UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII
Corso di Laurea Magistrale in Ingegneria Aerospaziale

Event-Driven 3D Reconstruction of Small Celestial Bodies in a Simulated Laboratory Environment

Relatore:
Prof. Sebastiano Chiodini

Studente:
Matteo Zanin
Matricola: 2123315

Anno Accademico 2024/2025

Abstract

This thesis investigates pose estimation algorithms for event cameras. Event cameras are innovative devices that detect motion faster than the update rate of traditional videos, and computer vision algorithms employ the new visual input to estimate trajectories in terrestrial or space applications. In particular, this work describes pose estimation in a simulated environment of deep space navigation, namely a spacecraft's circular orbit around a comet. From the simulation, the code must detect features, track their evolution, and establish how the camera moves without human intervention. Two main procedures are implemented to achieve this goal, one based entirely on event cameras and the other combining traditional and event videos. Together with the results, a complete analysis of the algorithms is provided: starting from the raw data captured from the scene, this thesis will describe how computer vision determines the relative motion between the camera and the features and how it represents the scene in three-dimensional coordinates.

Institutional Acknowledgments

The DVS/DAVIS technology was developed by the Sensors group of the Institute of Neuroinformatics (University of Zurich and ETH Zurich), which was funded by the EU FP7 SeeBetter project (grant 270324).

Acknowledgments

I would like to express my deepest gratitude to Professor Chiodini for his timely support, patience, and insightful guidance throughout this research. I am grateful to the Department of Industrial Engineering at the University of Padua for providing the environment, the knowledge, and the tools to carry out this work. I sincerely thank my friends and colleagues for making these years both enriching and joyful. A special thanks goes to Davide and Antonio, who have been my closest friends until the end of this journey. My heartfelt appreciation extends to all my childhood friends together with my professors, especially to my late Professor Monica Maran and to Diana Maschietto. Finally, I dedicate this work to my family: my Mom Stefania, my late father Roberto, my sister Federica, Diego, Alberto and the rest of my cousins, my uncles and aunts, and my granparents. Thank you all for your unwavering support and for giving me the strength to face every challenge.

Contents

Chapter 1

Introduction

This thesis investigates pose estimation algorithms for event cameras. Event cameras are an innovative device that detects motion in a scene, rather than collecting images, and the purpose of this work is to understand how the camera moves from its detections in a deep space navigation scenario.

This section will be dedicated to introducing pose estimation and deep space exploration, and to provide a description of the experiment conducted.

Even though the practical implementation will be adapted to the new acquisitions, the main steps required to find the motion of an event camera are similar in concept to those for traditional cameras:

1. Feature Detection

   The first operation is the characterization of the scene through the identification of peculiar points, called "features". These features differ from the rest of the scene, making them the easiest points to recognize.

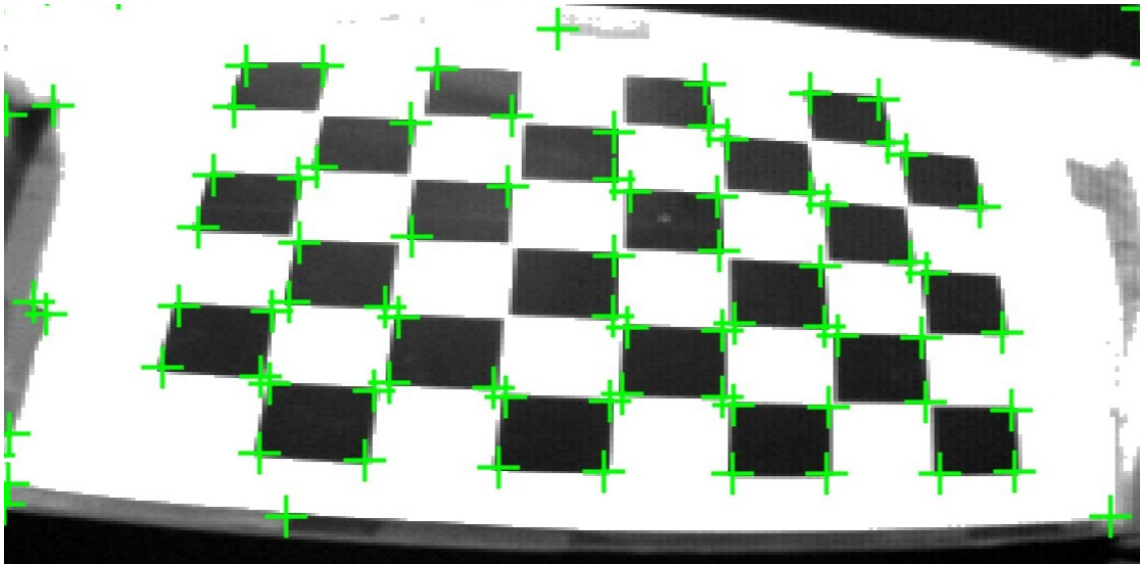


Figure 1.1: Image Feature Detection

2. Feature Tracking

   In the second step, the motion of these features is followed throughout the video. Again, since features are the most recognizable points, they are also the most convenient points to track.

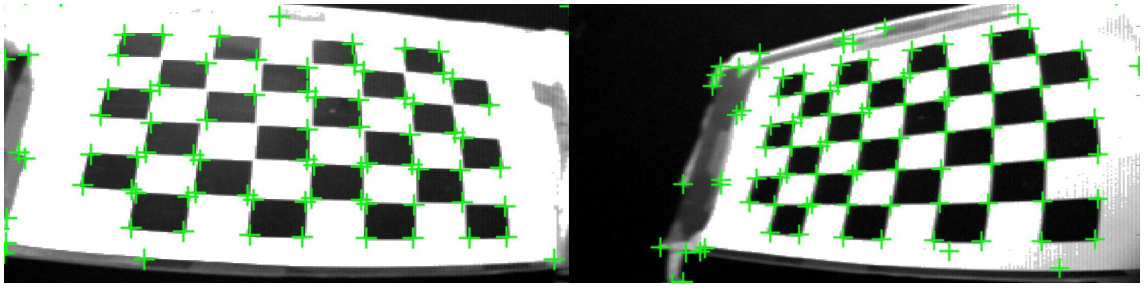


Figure 1.2: Image Feature Tracking

3. Pose Estimation and 3D Reconstruction

   In the third and final step, the tracked location of the features is used to understand the relative motion between the camera and the observed scene. In particular, the algorithms recover the camera motion together with a three-dimensional representation of the environment.

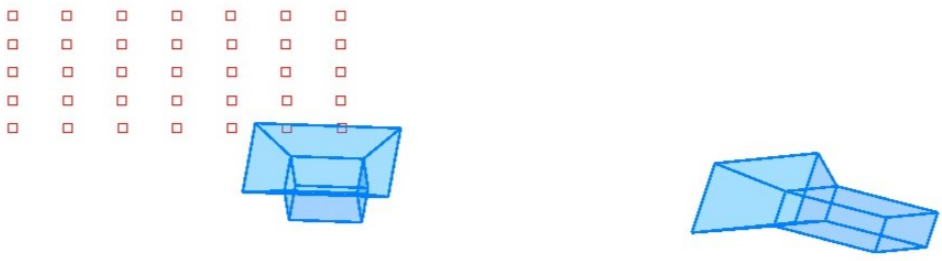


Figure 1.3: Image Pose Estimation

Event cameras are optimal in the context of deep space navigation, because they provide near real-time acquisitions (few μs) with very low power consumption (few mW) [8]. Furthermore, the device has a high dynamic range and works even in bright and dark environments, where traditional cameras are usually ineffective.

For these reasons, pose estimation with event cameras is investigated in the autonomous exploration of small celestial bodies. During the mission, a spacecraft approaches a celestial object and characterizes its shape, composition, and motion. The operation involves several sensors and phases, and therefore a brief description of the complete mission is provided [29].

1. Target Trajectory Estimation

   From Earth, the target ephemeris is estimated from radiometric and optical measurements. the trajectory of the spacecraft is computed from Earth observations, until its onboard instruments detect the target.

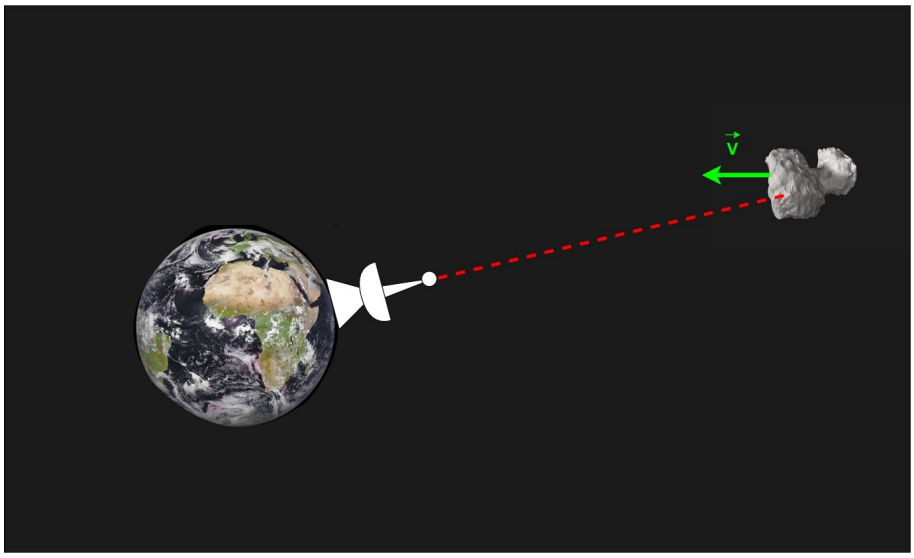


Figure 1.4: First Estimation of the Ephemeris of the Target

2. First Target Detection and Relative Navigation

   Visual observations occur at a distance of $10^7\,km$ from the target. During this phase, only a point is visible, spread on few pixels. A centroid-brightness moment algorithm predicts the relative target - spacecraft position and is used to correct the trajectory.

   Additionally, light-curve analysis estimates the spin rate of the moving target from the fluctuations of the light reflected on its rotating surface.

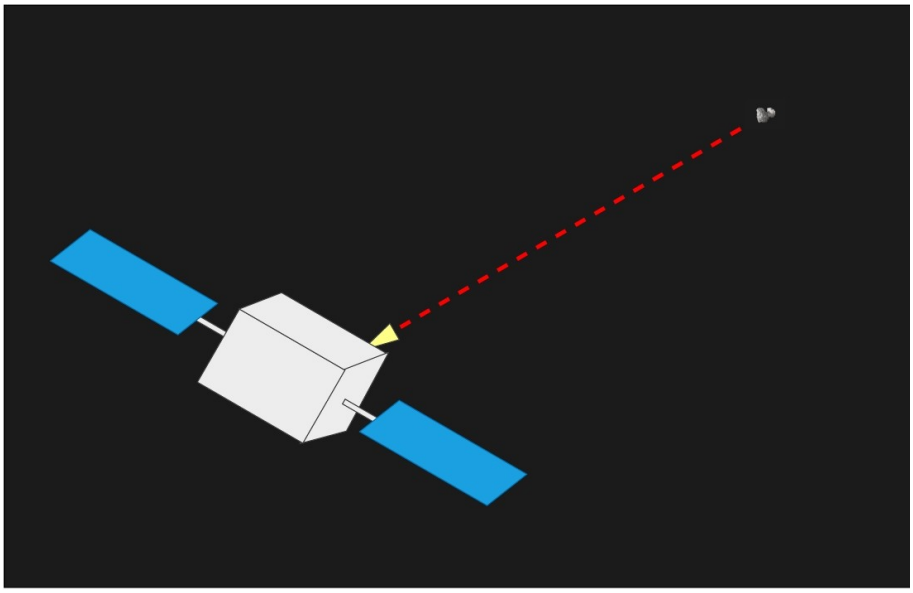


Figure 1.5: First Detection of the Body as a Point-Spread Function

3. Silhouette of the Body

   The silhouette of the body is visible once the target covers about ten pixels. The surface craters are still not visible, but the silhouette is sufficient to provide an initial estimation of the axis of spin and an upper bound to the shape of the object, called visual hull.

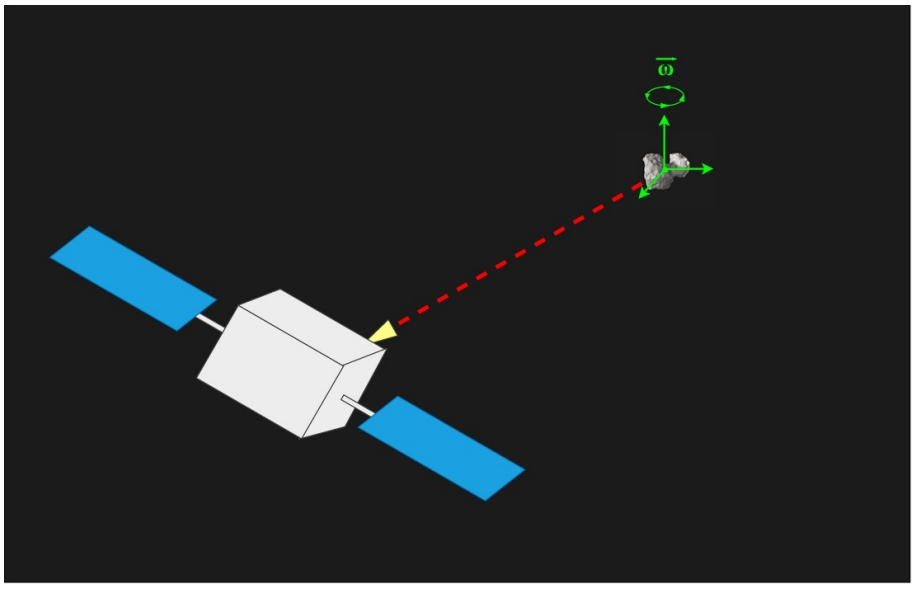


Figure 1.6: Silhouette of the Body

4. Feature Tracking of the Surface

   Once craters and landmarks on the surface are distinguishable, feature tracking is performed. Visual features are identified and tracked to estimate the satellite motion and to determine the shape of the surface. After many observations, a visual description of the object is obtained as well as an initial gravity model. The gathered informations are used to identify the ideal location for a landing site, considering thermal and surface stability and visibility from Earth and of the Sun.

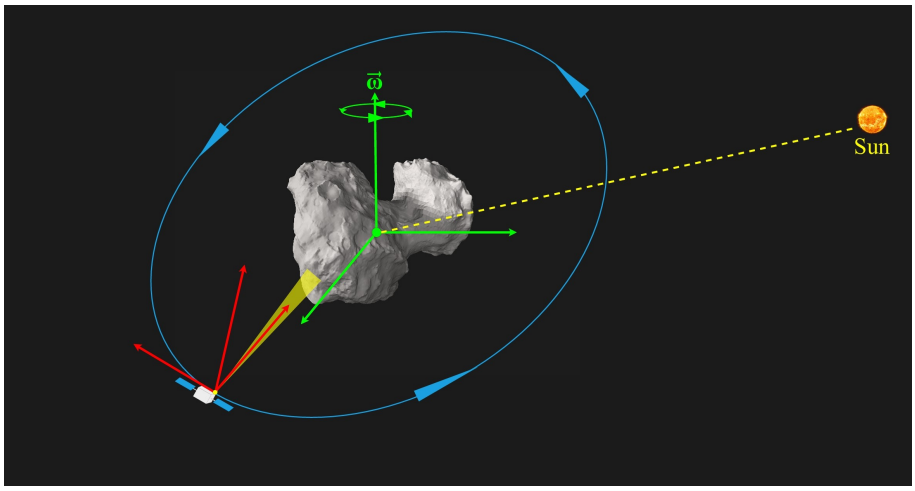


Figure 1.7: Visual Tracking of the Surface

5. Landing on the Surface

   Following the selection of a landing site, a lander descends on the surface of the target. The lander studies the composition and characteristics of the surface and may be used to deploy a rover to perform a fine estimation of the shape of the target. However, the rover cannot employ wheels due to micro-gravity, and therefore it may perform controlled hops to move [5].

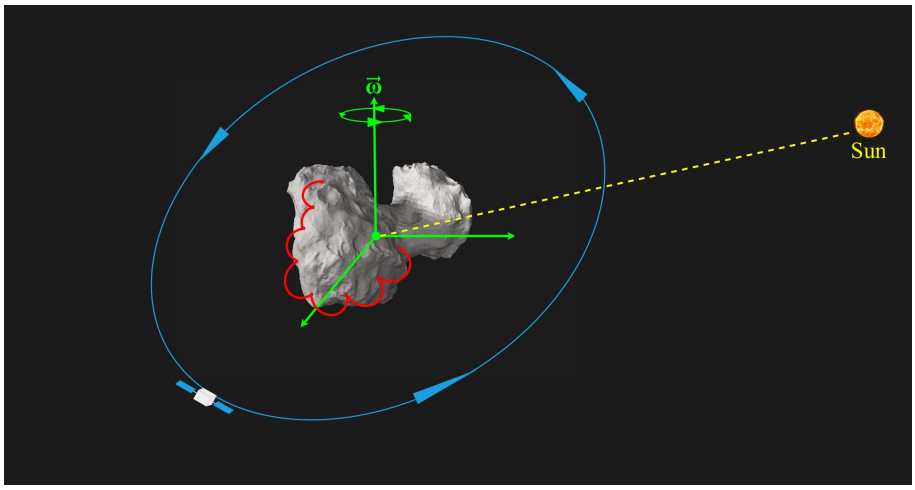


Figure 1.8: Hopping Rover

As mentioned in the first paragraph, this thesis will concentrate and experiment on feature tracking of small celestial objects. For this reason, a circular orbit of a spacecraft around a comet is simulated using an event camera. The camera reduces power consumption and increases temporal resolution and dynamic range, which are optimal properties during the observation of the surface of the target.

In the experiment, the celestial object is a 3D printed replica of the comet $67P/Churyumov - Geramisenko$, recorded on a iniVation DAVIS346 [16]. The DAVIS camera is a multisensory device capable of capturing traditional images and event data. In fact, it performs event acquisition (DVS dynamic vision sensor), traditional acquisition, and it possesses an IMU inertial measurement unit.

Figure 1.9: DAVIS346 Camera

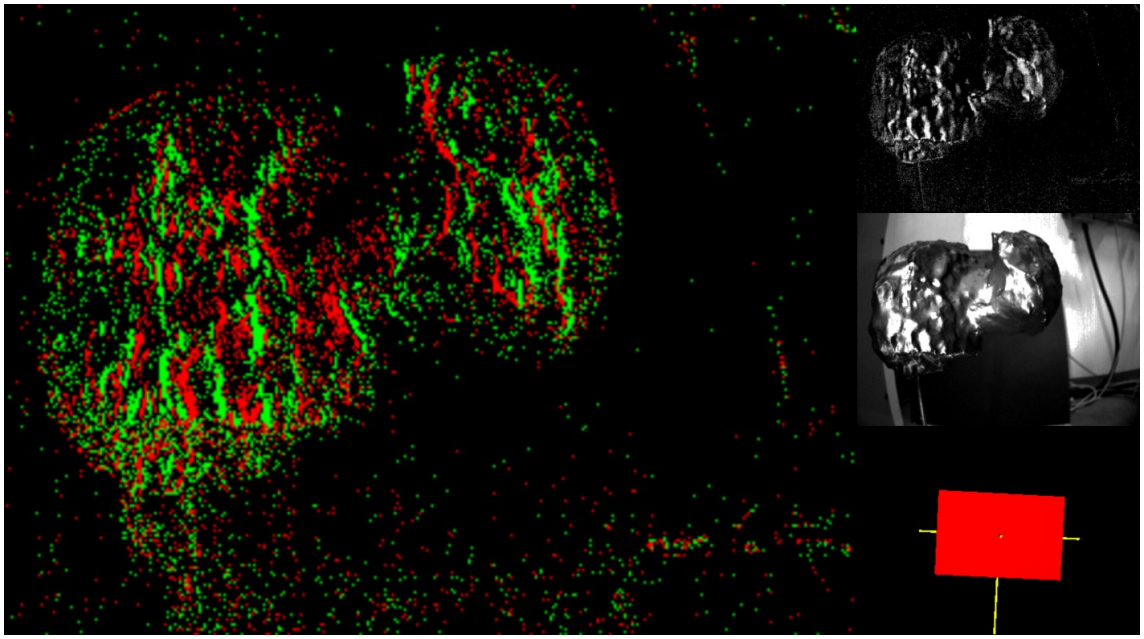


Figure 1.10: Acquisitions of a DAVIS346 Using IniVation DV [17]

The camera was positioned at a small distance from the object in a dark environment. Thanks to relative motion, the camera remained stationary while the comet was rotated around its principal axis.

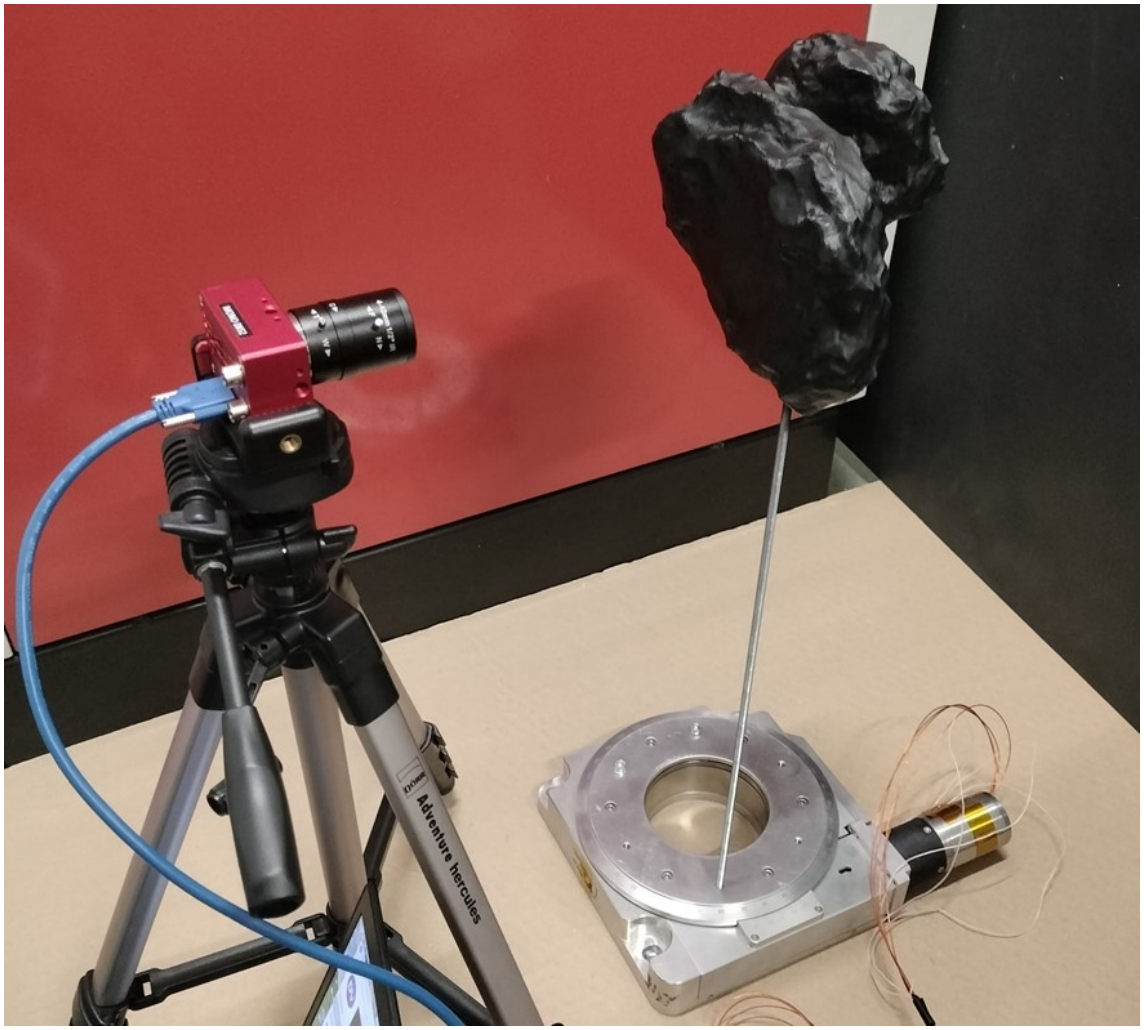


Figure 1.11: Experimental Set-Up

During the subsequent analysis, the recorded data were used to obtain an estimate of the camera trajectory. The steps required to complete the procedure were introduced in the first section: first, the algorithm identifies visual features, and then it defines their evolution. The motion of the camera is obtained by comparing the position of the said features in multiple views of the same scene. Finally, in practical implementations, the result is fused to the results of other sensors to obtain a final reconstruction of the scene and of the pose.

A visual representation of the main steps performed during the process is provided on the next page.

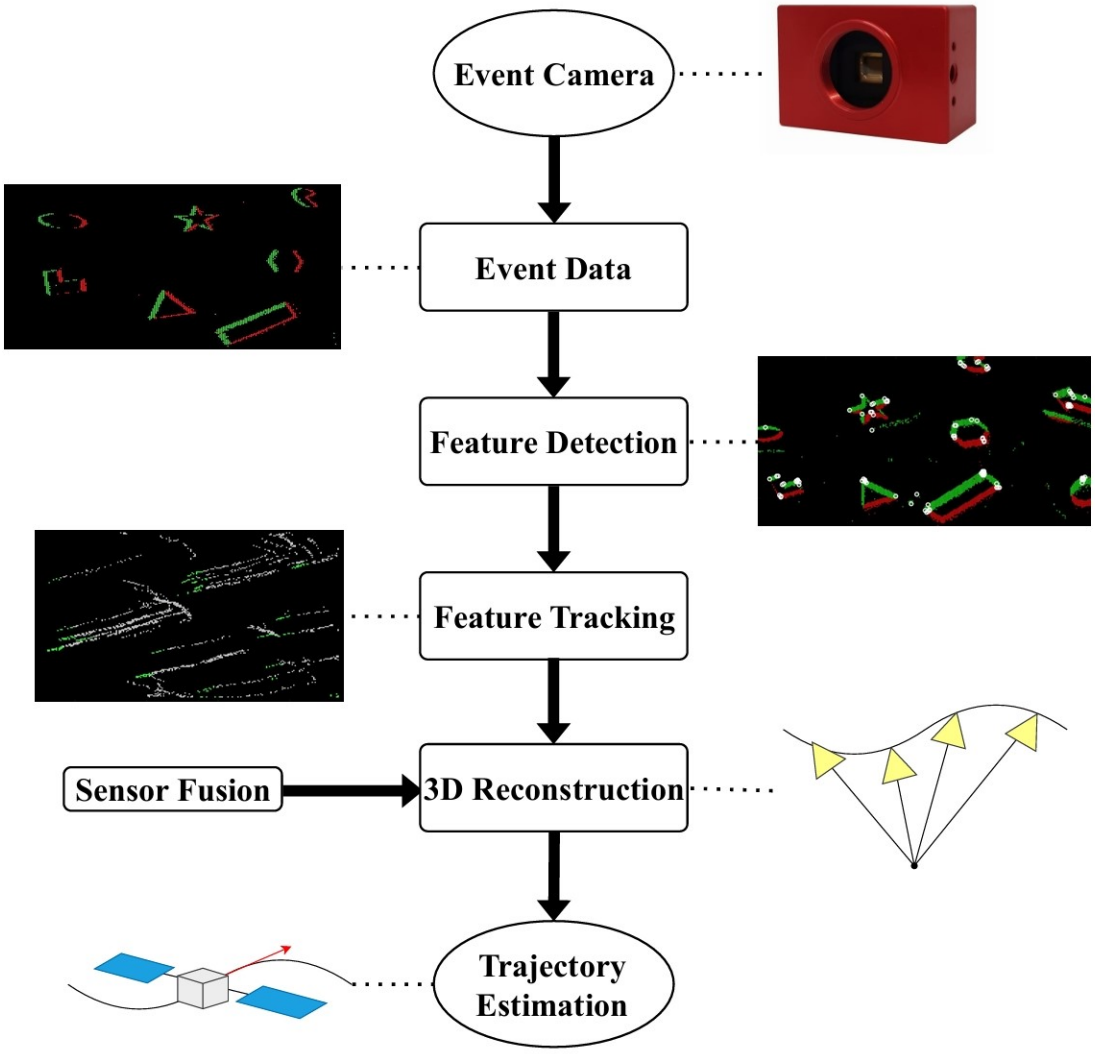
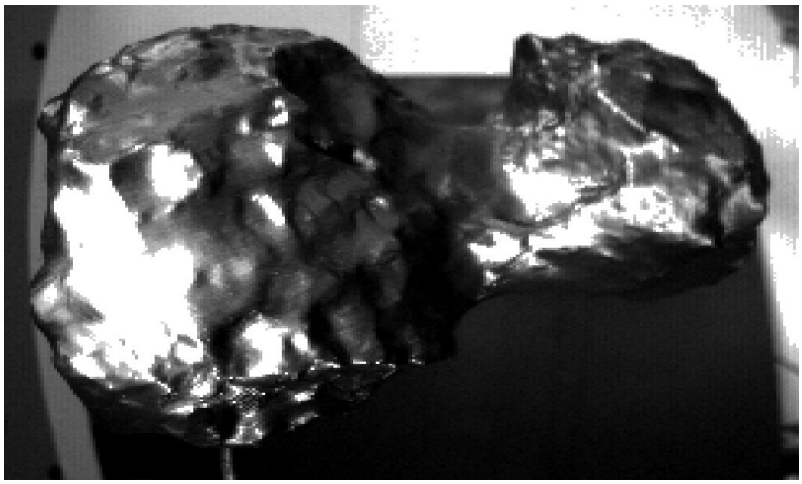


Figure 1.12: Operations in the Processing of Events

An additional video was used to assess the accuracy of the implementations. The video is *shapes_6dof*, provided in the Event Camera Dataset [27]. Additional videos were tested, but they will not be included in the representations to maintain the focus on the primary videos.

(a) shapes_6dof



(b) $67P/Churyumov - Geramisenko$

Figure 1.13: Original Videos Used in the Experiment

The remainder of this introductory section provides a brief description of the following chapters.

Chapter 1 is dedicated to introducing all the basic concepts required for the experiment. This chapter, together with the rest of this work, aims to provide a complete beginner guide to the field of computer vision.

Section 1.1 will provide a description of event cameras, how they measure motion, and some properties for later algorithms.

Section 1.2 will introduce the reader to the camera lens system and to homogeneous coordinates.The notation used in this work is greatly inspired by the work on Multiple View Geometry [15].

Section 1.3 explains how computers compute derivatives [33] and averages [6]. Due to the discretized acquisitions, computer vision opts for kernel operations, which involve the values surrounding the interest point. The kernel operations are applied in the practical implementation of the algorithms in chapter 2.

Chapter 2 will deal with the 1^{st} and 2^{nd} steps of pose estimation. Two procedures are described: one based only on events and the other based on event cameras and traditional ones. The algorithms in this section were implemented by the author in MATLAB, following the research provided by the original authors.

Section 2.1 describes the event-only procedure for feature detection and tracking. The features are detected using the FA-Harris algorithm [21], which is based on time surfaces SAE* [1]. FA-Harris is composed of ARC* [1] and a binarized version [36] of the Harris feature detection algorithm [14]. Meanwhile, the tracking algorithm is adapted from the method described in [1].

The detection and tracking algorithms are based on classic techniques, but adapted for event cameras. Due to the rigorous validation required in aerospace applications, no algorithm implements machine learning, as it could fail due to unforeseen conditions outside the known simulations used in training. In space applications, classic techniques are generally more robust and therefore preferred, although research is currently developing machine learning algorithms that may be used in the future.

Section 2.2 will provide a different approach to the problem, where both feature detection and tracking are performed in a single code, EKLT [12]. Inside EKLT, the stability of traditional images is fused with the reaction time of event cameras, resulting in a robust estimation of the evolution of each feature.

Chapter 3 concludes with the 3^{rd} step in pose estimation. In the chapter, the geometry theorems required to compute the camera motion are proposed. In particular, this work will focus mainly on the algorithms implemented in MATLAB's Computer Vision Toolbox.

This section contains the estimation of the essential matrix E [15] [30] [20] [35], the triangulation of three-dimensional features [15], the Perspective-n-Point problem [11] [19], bundle adjustment [15] and camera calibration [6] [37] [28] [32].

Finally, the conclusions will be summarized in the last section 4.

1.1 Event Cameras

1.1.1 Event Generation

This chapter describes how event cameras detect motion in the scene through variations in the illumination condition. To introduce the device and its properties, a brief introduction to the system is provided, along with a comparison to traditional cameras.

When a traditional camera records a video, its frames are collected uniformly in time. Each frame captures the brightness of the entire field of view, without distinguishing between static and dynamic scenes. As a result, the device fails to capture movements in between frames, and it records the same static background each time, even for dynamic scenes. Furthermore, motion blur may appear in dynamic scenarios, since the camera requires to be exposed to light for a discrete exposition time to collect the frame.

All these limitations in traditional cameras reduce the ability to track moving objects with high temporal resolution and accuracy.

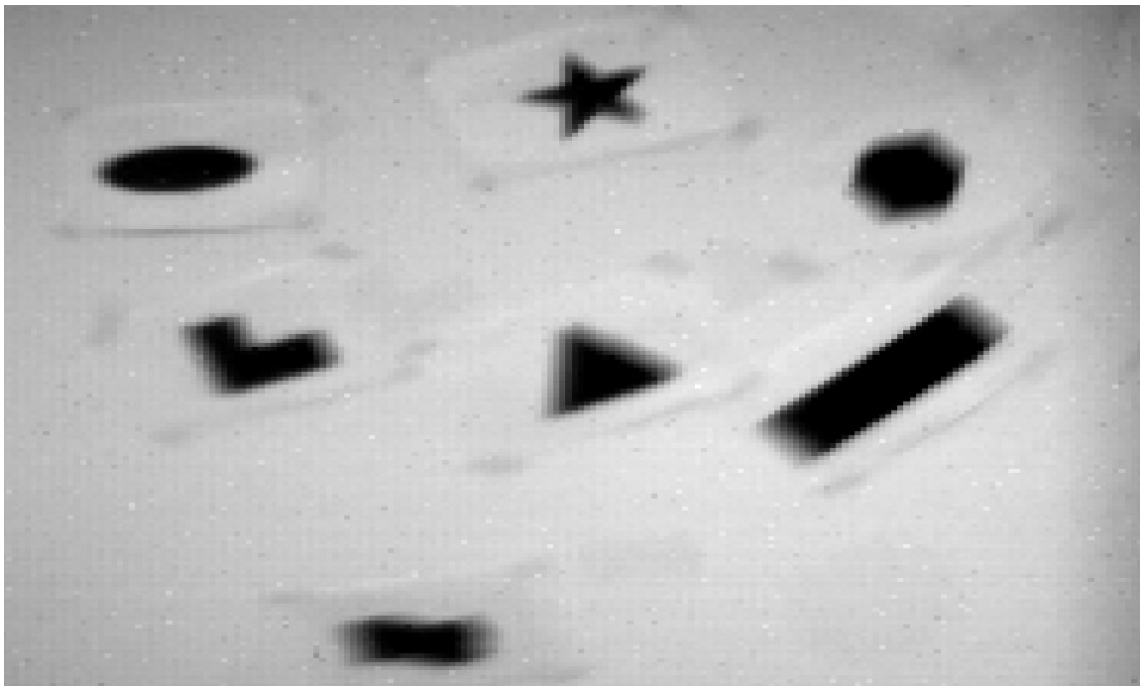


Figure 1.14: Motion Blur in a Traditional Camera

To address the issues of traditional cameras in fast applications, event cameras have been developed to collect information at a suitable update rate. Only useful data regarding motion are stored, while static backgrounds are ignored.

The event camera identifies variations in the illumination environment inside its field of view. Each pixel responds to sudden and localized changes in brightness and works independently from the others. As a result, the camera senses movements because of the reflected light that bounces on dynamic scenes.

Given the continuous and multivariate function $I(\mathbf{u}, t)$, which describes the brightness of the scene as seen by the lenses of the camera in space $\mathbf{u} = (x, y)^T$ and time t, typical event cameras respond to logarithmic variations of this function [12]:

$$L(\mathbf{u}, t) \doteq \log_e(I(\mathbf{u}, t)) \tag{1.1}$$

Ideally, an update is recorded if the logarithmic scale varies more than a certain threshold $\pm C, C > 0$ over a limited period of time Δt, which is typically of the order of microseconds $[\mu s]$. For example, considering that an update occurred at time t_k in the pixel with coordinates $\mathbf{u}_k = (x_k, y_k)^T$, it means that the logarithmic variation $\Delta L(\mathbf{u}_k, t_k)$ in that location is larger than the threshold:

$$\Delta L(\mathbf{u}_k, t_k) \doteq L(\mathbf{u}_k, t_k) - L(\mathbf{u}_k, t_k - \Delta t) > \pm C \tag{1.2}$$

The update takes the name of "event" e and is saved in the memory of the device. Each event e stores the coordinates $(x, y)^T$ where the variation occurred, the timestamp t, and the sign of the brightness variation $\{+1; -1\}$, which is called the polarity p. For example, for the k^{th} event e_k:

$$e_k = \{p_k, t_k, x_k, y_k\} \tag{1.3}$$

The data stored in an event camera are considerably different from that of a traditional camera. The latter has to collect one variable for each pixel in each frame, which means a total number of $height \times width \times frames$ variables, and store them as integer values between $[0; 255]$. In contrast, the collected events are asynchronous and independent of each other. Their number and distribution in space and time varies depending on the dynamic of the scene. Usually, there may be instances where the scene is full of events and other times in which everything is static.

1.1.2 Events Accumulation into Frames

A single event is an asynchronous and localized detection of motion and therefore does not provide enough information alone. For this reason, events are accumulated into frames to reproduce the dynamic scene and implement feature detection and tracking.

Each frame is produced by gathering all events that occurred in a time interval $\Delta\tau$, which is considerably lower than the frame rate of traditional cameras. The produced image represents the variation of brightness that occurred in the frame and is described as a function $F_m^E(x, y)$ of space $\mathbf{u} = (x, y)^T$ and of frame m. $F_m^E(x, y)$ is analytically defined as the accumulation of all N_e events in the frame m^{th}, namely $e_k = \{p_k, t_k, x_k, y_k\}, \forall t_k \in [(m-1)\Delta\tau; m\Delta\tau]$:

$$F_m^E(x, y) \doteq \sum_{k=1}^{N_e} p_k \, \delta(\mathbf{u} - \mathbf{u}_k) \tag{1.4}$$

Where δ is the Kronecker Delta, which is 1 if $\mathbf{u} = \mathbf{u}_k$ and 0 otherwise.

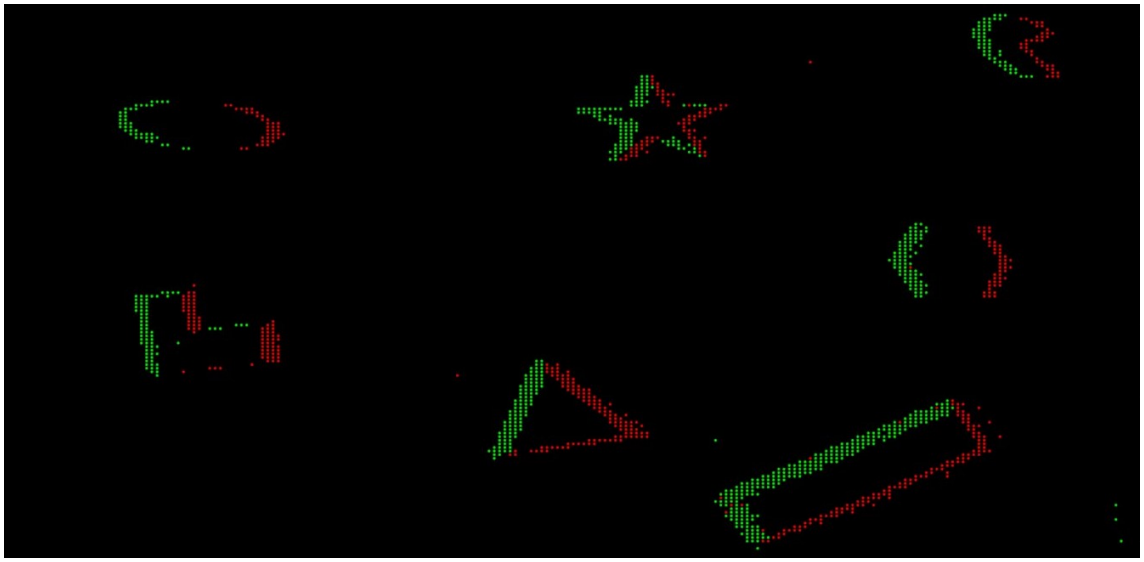


Figure 1.15: Event Frame

Small $\Delta\tau$ are essential to assume the brightness constancy [10]. Considering two successive event frames m and $(m+1)$, brightness constancy assumes that the same value of brightness is shared by pixels representing the same feature. For example, considering two corresponding pixels $(x, y)^T$ and $(x + \Delta x, y + \Delta y)^T$, respectively, in the frames m and $(m+1)$, the constancy of brightness along a point trajectory may be expressed as:

$$L(x, y, m\Delta\tau) = L(x + \Delta x, y + \Delta y, (m+1)\Delta\tau) \tag{1.5}$$

As a result, the derivative of the logarithmic brightness $L(\mathbf{u}, t)$ evaluated along the point trajectory $\mathbf{u}(t)$ is null, and it may be expressed in its spatial and temporal component to obtain the optical flow constraint:

$$\frac{dL(\mathbf{u}(t), t)}{dt} = \nabla L(\mathbf{u}(t), t) \cdot \dot{\mathbf{u}}(t) + \frac{\partial L(\mathbf{u}(t), t)}{\partial t} = 0 \tag{1.6}$$

Where $\nabla = (\partial/\partial x, \partial/\partial y)^T$ is the spatial derivative operator.

The optical flow constraint may be introduced into the equation for the generation of an event $e = \{p, t, x, y\}$ 1.2 . Using first-order Taylor's expansion on the latter equation:

$$\Delta L(\mathbf{u}, t) \doteq L(\mathbf{u}, t) - L(\mathbf{u}, t - \Delta t) \approx \frac{\partial L(\mathbf{u}, t)}{\partial t} \Delta t \tag{1.7}$$

Substituting equation 1.6 into equation 1.7:

$$\Delta L(\mathbf{u}, t) \approx \frac{\partial L(\mathbf{u}, t)}{\partial t} \Delta t \approx -\nabla L(\mathbf{u}, t) \cdot \dot{\mathbf{u}}(t) \Delta t \tag{1.8}$$

Which may also be expressed in terms of displacement $\mathbf{u}$:

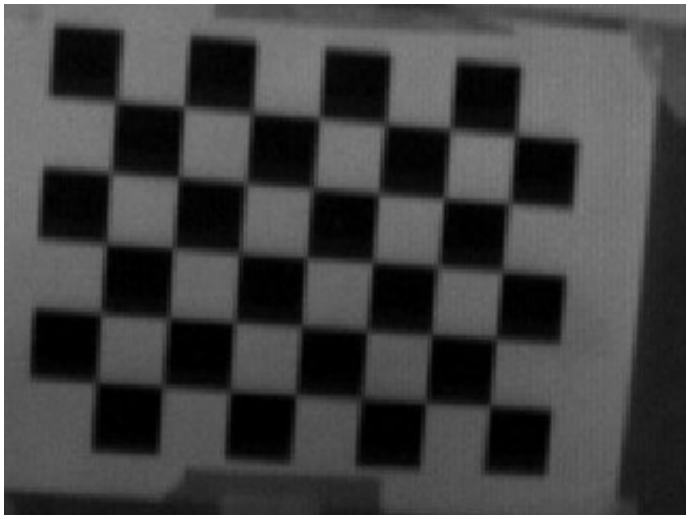
$$\Delta L(\mathbf{u}, t) \approx -\nabla L(\mathbf{u}, t) \cdot \mathbf{u}(t) \tag{1.9}$$

Given that peaks in the spatial gradient ∇L are produced by edges and features of moving objects, two observations can be made on the last equation 1.9:

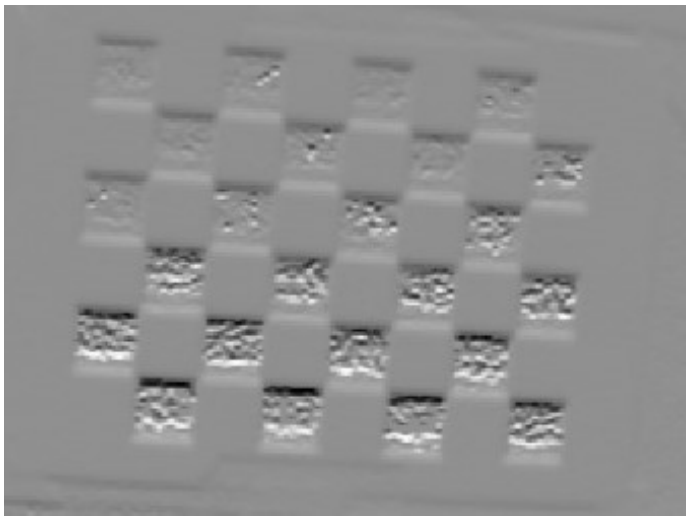
1. Events will form only on the edges perpendicular to motion, while no event is produced if the displacement is parallel to the edge (as shown in the image below).

2. Every event in a frame $F_m^E(x, y)$ is triggered by a spike in the spatial gradient of the logarithmic brightness function ∇L projected in the direction of motion $\mathbf{u}$. Therefore, F_m^E approximates the gradient of the logarithmic image of the same scene projected in the direction of motion.

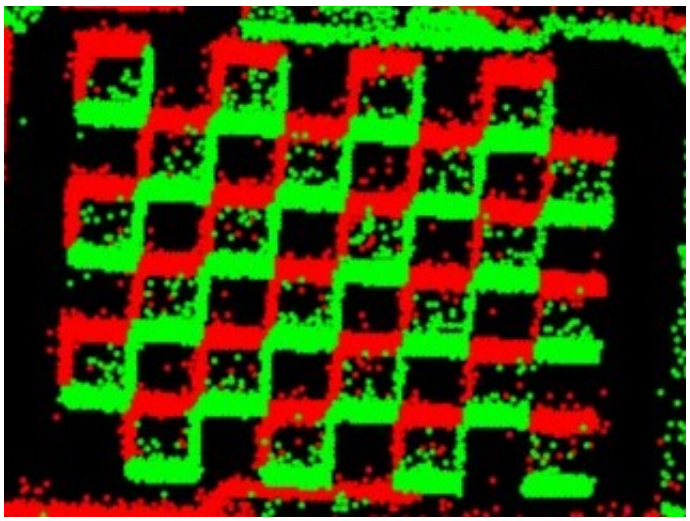
In the images reported in the next page, a checkerboard is depicted moving vertically. The same scene is shown using a standard camera, using the vertical gradient of the standard image in logarithmic scale, and using an event frame. As observed, the event frame approximates the gradient of the image.



(a) Standard Frame



(b) Image Gradient



(c) Event Frame

Figure 1.16: Checkerboard

1.2 Perspective Projection

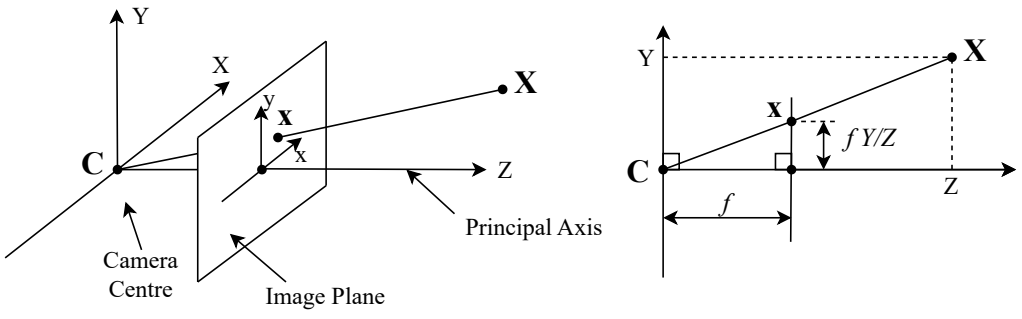
1.2.1 Single Perspective Camera

This section focuses on the geometry of a single perspective camera [15]. The camera lens system may be defined as a map that projects the coordinates of a three-dimensional point into the coordinates of its projection on the sensing element, called image plane. The analysis is valid regardless of whether an event camera or a traditional camera is used: both systems use lenses to capture the scene. In some cases, such as the DAVIS camera used in the experiment, the same system of lenses is used to capture both traditional frames and events.

Consider a target observed by the camera and described as a vector $\tilde{X} \in \mathbb{R}^3$ $\tilde{X} = (X, Y, Z)^T$. The camera centre $\tilde{C}$ is located in the origin of the current view, while the image plane is in front of it. Although the image plane is positioned at focal length f along the Z axis, the third component of the projection is generally ignored.

The camera $\tilde{P}$ maps the point $\tilde{X}$ to the corresponding projection in the image plane $\tilde{x} \in \mathbb{R}^2$, $\tilde{x} = (x, y)^T$. The projection $\tilde{x}$ is defined as the intersection of the image plane and the line that connects the point $\tilde{X}$ to the centre of the projection $\tilde{C}$.

$$\begin{aligned} \tilde{P}: \quad \mathbb{R}^3 &\longrightarrow \mathbb{R}^2 \\ \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} &\mapsto \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \end{pmatrix} \end{aligned} \tag{1.10}$$



(a) Axonometry (b) Orthogonal Projection

Figure 1.17: Single Perspective Camera (Adapted From [15])

1.2.2 Homogeneous Coordinates

The map $\tilde{P}$ between the point $\tilde{X}$ and the corresponding image $\tilde{x}$ is not linear. For this reason, it is convenient to describe the point and image vectors as homogeneous coordinates.

Given, for example, a generic vector $\tilde{X} \in \mathbb{R}^3$, $\tilde{X} = (X, Y, Z)^T$, the homogeneous representation of $\tilde{X}$ is defined as $\mathbf{X} \in \mathbb{R}^4$, $\mathbf{X} = (X_1, X_2, X_3, X_4)$, $X_4 \neq 0$ such that:

$$X = X_1/X_4,\ Y = X_2/X_4,\ Z = X_3/X_4 \tag{1.11}$$

Where $\mathbf{X}$ and $\tilde{X}$ are, respectively, the homogeneous and inhomogeneous vectors that describe the target. The same representation could be defined for a vector $\tilde{x} \in \mathbb{R}^2$.

Applying homogeneous coordinates to the point studied $\tilde{X}$ and the respective projection $\tilde{x}$ in the image plane:

$$\tilde{X} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X/1 \\ Y/1 \\ Z/1 \end{pmatrix} \mapsto \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{X} \tag{1.12}$$

$$\tilde{x} = \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \mathbf{x} \tag{1.13}$$

Now, the map $\tilde{P}$ 1.10 will be expressed in homogeneous coordinates. The new relationship $\mathbf{P}$ is a linear map and may be expressed in terms of matrix multiplication:

$$\begin{array}{rccc} \mathbf{P}: & \mathbb{R}^4 & \longrightarrow & \mathbb{R}^3 \\ & \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} & \longmapsto & \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \end{array} \tag{1.14}$$

This may also be expressed as:

$$\mathbf{x} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X} \doteq K \left[\mathbb{1}_3 | 0\right] \mathbf{X} \tag{1.15}$$

Where K is the camera calibration matrix and $[\mathbb{1}_3 \,|\, 0]$ is an augmented matrix composed of an identity matrix 3×3 and a column vector of 0.

The matrix K may be adapted according to different formulations of the camera lens system and to account for deformation. For example, considering the point $(p_x, p_y)^T$ as the origin of the image plane, the map $\tilde{P}$ between the point $\tilde{X}$ and the projection $\tilde{x}$ becomes:

$$(X, Y, Z)^T \mapsto (fX/Z + p_x, fY/Z + p_y)^T \tag{1.16}$$

Which, expressed in homogeneous coordinates, becomes:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.17}$$

As before:

$$\mathbf{x} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.18}$$

$$\mathbf{x} \doteq K \left[\mathbb{1}_3 | 0\right] \mathbf{X} \tag{1.19}$$

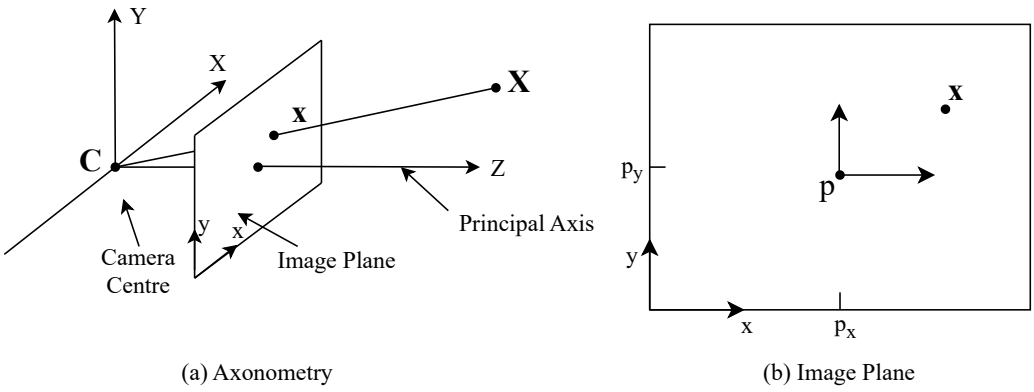


Figure 1.18: Offset in the Image's Coordinates (Adapted From [15])

1.2.3 Camera Rotation and Translation

The previous section assumed that the camera centre $\tilde{C}$ was located in the origin of the global reference system. However, in general, the global coordinate system may be positioned outside the camera, as seen in the image below.

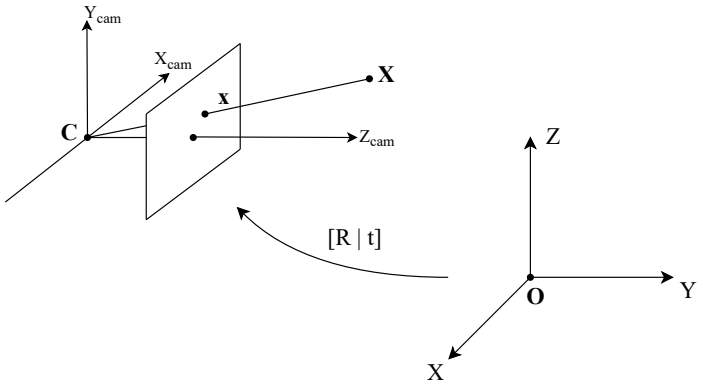


Figure 1.19: World and Camera Coordinate Systems (Adapted From [15])

Given a point represented by the inhomogeneous vector $\tilde{X}$ in global coordinates, the same vector is expressed in the camera system as $\tilde{X}_{cam} = R\tilde{X} + t$, where t represents the coordinates of the camera centre $\tilde{C}$ in the global system and R is a rotation matrix 3×3 representing the orientation of the camera system with respect to the global.

$$\begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \tag{1.20}$$

The same expression may be written in homogeneous coordinates:

$$\begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{pmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.21}$$

Or simply:

$$\mathbf{X}_{cam} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{1.22}$$

By introducing the last equation 1.22 into the previous formula 1.19, the complete map $\mathbf{P}$ of a single perspective camera in the global reference system becomes:

$$\mathbf{x} = K \left[\mathbb{1}_3 | 0\right] \mathbf{X}_{cam} = K \left[\mathbb{1}_3 \,|0\right] \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \mathbf{X} = K \left[R \mid t\right] \mathbf{X} = \mathbf{P}\mathbf{X} \tag{1.23}$$

The matrix $\mathbf{P}$ is divided into intrinsic parameters K, related to lenses, and extrinsic parameters $[R \,|\, t]$, related to the position and orientation of the camera in the global system.

1.2.4 Loss of Depth

Projecting $\mathbf{X}$ to the image point $\mathbf{x}$ causes to lose the sense of depth. In fact, the matrix $\mathbf{P}$ has rank 3 despite having 4 columns.

The loss of information may be verified by considering the line that connects any point to the centre of the camera. Given the line $\mathbf{X}(\lambda)$ that contains any homogeneous point $\mathbf{A}$ and the centre of the camera $\mathbf{C}$, the points on this line may be represented as:

$$\mathbf{X}(\lambda) = \lambda\mathbf{A} + (1-\lambda)\mathbf{C}, \lambda \in \mathbb{R} \tag{1.24}$$

Applying the map $\mathbf{x} = \mathbf{PX}$, the points on this line are projected to:

$$\mathbf{x} = \mathbf{PX}(\lambda) = \lambda\mathbf{PA} + (1-\lambda)\mathbf{PC} \tag{1.25}$$

Being $\mathbf{C}$ the centre of the camera, it is the null space of the camera projection matrix $\mathbf{PC} = \mathbf{0}$. As a consequence:

$$\mathbf{x} = \mathbf{PX}(\lambda) = \mathbf{P}(\lambda\mathbf{A}) \tag{1.26}$$

This means that all the points on the line $\mathbf{X}(\lambda)$ are mapped to the same image point $\mathbf{x}$.

Therefore, given an image point $\mathbf{x}$, its back-projection may be found only up to a constant. The back-projection is computed by applying the pseudo-inverse matrix $\mathbf{P}^+ = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}$ to the map $\mathbf{x} = \mathbf{PX}$.

$$\mathbf{P}^+\mathbf{x} = \mathbf{P}\mathbf{P}^+\mathbf{X} = \mathbf{P}\mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}\mathbf{X} = \mathbf{X} \tag{1.27}$$

The entire segment between the pseudo-inverse point $\mathbf{P}^+\mathbf{x}$ and the camera centre $\mathbf{C}$ may be defined using a parameter $\lambda \in \mathbb{R}^+$:

$$\mathbf{X}(\lambda) = \mathbf{P}^+\mathbf{x} + \lambda\mathbf{C} \tag{1.28}$$

Where, if $\lambda = 0 \Rightarrow \mathbf{X} = \mathbf{P}^+\mathbf{x}$ and if $\lambda \to \infty \Rightarrow \mathbf{X} = \mathbf{C}$.

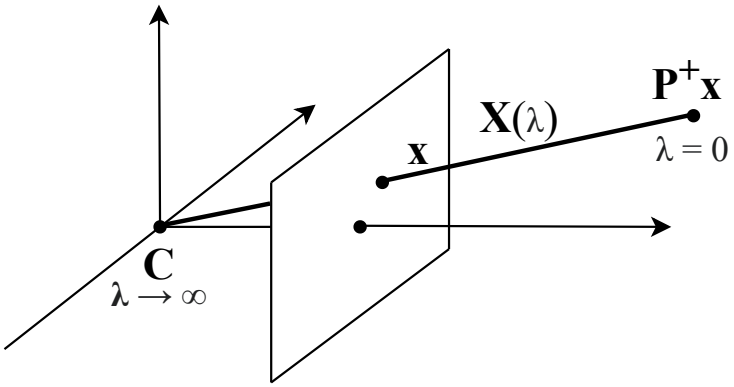


Figure 1.20: Back-Projection

1.2.5 Points and Lines in Homogeneous Coordinates

Homogeneous coordinates allow for an easier representation of vectors and points involved in perspective geometry. In addiction, they also possess several useful properties that will become relevant in the pose estimation procedure:

1. Given a generic inhomogeneous vector $(a/c, b/c)^T$, its homogeneous representation is not unique. In fact, the homogeneous vector $(a, b, c)^T$ is equivalent to $(ka, kb, kc)^T = k(a, b, c)^T \, \forall \, k \neq 0$, since they both satisfy the definition of homogeneous vector of $(a/c, b/c)^T$. In homogeneous coordinates, a vector is represented up to a non-zero constant.

2. Given a line l represented by the equation $l : ax + by + c = 0$, it becomes natural to represent it as a vector $(a, b, c)^T$. A point $\tilde{x} = (x_o, y_o)^T$ lies on the line $l \iff ax_o + by_o + c = 0$. By expressing the point $\tilde{x}$ in homogeneous coordinates $\mathbf{x} = (x_o, y_o, 1)^T$, the condition may be expressed as the inner product $(x_o, y_o, 1)(a, b, c)^T = (x_o, y_o, 1)l = 0$. Therefore:

$$\mathbf{x} \in l \iff \mathbf{x}^T l = 0 \tag{1.29}$$

3. Given two lines $l : (a, b, c)^T$ and $l' : (a', b', c')^T$, the homogeneous vector $\mathbf{x} = l \times l'$ defines their intersection. This is easily verified by demonstrating that $\mathbf{x}^T l = 0 \wedge \mathbf{x}^T l' = 0$. Using, respectively, the definition of $\mathbf{x}$ as a cross product, circular permutation of the triple scalar product $((a \times b) \cdot c = (c \times a) \cdot b)$ and knowing that $a \times a = 0$:

$$\mathbf{x}^T l = (l \times l')^T l = (l \times l)^T l' = 0 \tag{1.30}$$

   And the same for $\mathbf{x}^T l' = 0$.

4. Given two points $\mathbf{x}$ and $\mathbf{x}'$ expressed in homogeneous coordinates, the line that passes though the two points is $l = \mathbf{x} \times \mathbf{x}'$. This can be verified using the second property, because $\mathbf{x}^T l = \mathbf{x}^T(\mathbf{x} \times \mathbf{x}') = \mathbf{x}'^T(\mathbf{x} \times \mathbf{x}) = 0$ and the same for $\mathbf{x}$'.

1.3 Operators in Computer Vision

1.3.1 Smoothing Kernel

When a scene is observed, the data arriving from the environment and passing through the camera lens system are collected only at specific locations, called pixels. The captured frame is a discrete signal and must be evaluated as such.

Given a frame, computer vision does not work with continuous functions $f(\mathbf{u}, t) : \mathbb{R}^3 \to \mathbb{R}$ to describe the scene in space $\mathbf{u} = (x, y)^T$ and time t. Instead, the function $f(\mathbf{u}, t)$ is captured only at the specific pixel coordinates and at a specific time $\bar{t}$:

$$g = f|_\Omega[\mathbf{u}, t], \quad g : \Omega \to \mathbb{R},\ \Omega = \{1, 2, ..., Width\} \times \{1, 2, ..., height\} \times \{\,\bar{t}\,\} \quad (1.31)$$

In signal processing, the brackets $[\mathbf{u}, t]$ are used to emphasize that the function is a discrete signal. Given that the scene is captured at a specific time $t = \bar{t}$, which is constant for each frame, the function representing a single frame will be expressed as $f|_\Omega[x, y]$ from now on.

Several operations may be performed to retrieve information about the observed scene from the captured frame $f|_\Omega[x, y]$. This section describes the most common operators in computer vision and in particular those that will be used in the experiments.

The first operator is the smoothing filter [6] and it is used to reduce localized noise by performing an averaged sum. The smoothed function $f_s|_\Omega$ is computed as a discrete convolution of the original function $f|_\Omega[x, y]$ and averaging weights $\tilde{w}[x, y]$.

$$f_s|_\Omega[x, y] = \tilde{w} * f|_\Omega[x, y] := \sum_{i,j=-n}^{n} \tilde{w}[i, j]\ f|_\Omega[x - i, y - j] \quad (1.32)$$

From the definition of the discrete convolution product, the operator assigns to each pixel the sum of the values in its neighbourhood weighted by $\tilde{w}[x, y]$. For computational reasons, the weighted average is taken only in a matrix $(2n + 1) \times (2n + 1)$ around the central pixel, where n is the half-size of the patch:

The most common weights are the values of the Gaussian function $G(x, y)$ because they naturally favour the central pixels over the external ones. The Gaussian function is centred around the point of interest $\mu = 0$ *pixel* and with the standard deviation $\sigma = 1$ *pixel*:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, (x, y) \in \mathbb{R}^2 \quad (1.33)$$

As mentioned above, functions in computer vision can only be evaluated at specific locations. Therefore, the weights must be described as a discrete function, which takes the name of Gaussian kernel or smoothing kernel:

$$G[x, y] = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad x, y \in \{-n, ..., n\} \quad (1.34)$$

Furthermore, it is common practice to normalize the kernel so that the total sum of the weights is 1 and the total average value of the function $f|_\Omega$ is preserved:

$$\tilde{w}_G[x, y] = \tilde{G}[x, y] = \frac{G[x, y]}{\sum_{i,j=-n}^{n} G[i, j]} \quad (1.35)$$

In many cases, the Gaussian formula $\tilde{G}[x, y]$ may require too much computational cost to be efficiently implemented several times within an algorithm. In those situations, it is usually replaced by the binomial distribution, which approximates the normal distribution as demonstrated by the central limit theorem.

The binomial distribution could be defined in two dimensions, like the Gaussian kernel; however, in this experiment, the smoothing filter is actually applied only in one direction. In fact, the univariate filter is commonly applied in the direction perpendicular to the desired gradient, to reduce localized noise.

The binomial approximation of the univariate Gaussian distribution $\tilde{G}[x]$ is presented here. As mentioned above, the $2n + 1$ weights are normalized so that their total sum is 1 and the total mean value of the original function is preserved.

$$w_B[k - n] = \binom{2n}{k} = \frac{2n!}{(2n - k)!k!}, k = 0, 1, ..., 2n \tag{1.36}$$

$$\tilde{w}_B[k - n] = \frac{w_B[k - n]}{\sum_{i=-0}^{2n} w_B[i - n]} \tag{1.37}$$

For example, using a kernel size $2n+1 = 5$ ($\Rightarrow n = 2$) in the univariate distribution, the normalized binomial weights $\tilde{w}_B$ and the Gaussian weights $\tilde{w}_G$ are:

$$\tilde{w}_B[-2] = \frac{1}{16},\ \tilde{w}_B[-1] = \frac{4}{16},\ \tilde{w}_B[0] = \frac{6}{16},\ \tilde{w}_B[1] = \frac{4}{16},\ \tilde{w}_B[2] = \frac{1}{16} \tag{1.38}$$

$$w_G[-2] \approx 0.0540,\ w_G[-1] \approx 0.2420,\ w_G[0] \approx 0.3989,\ w_G[1] \approx 0.2420,\ w_G[2] \approx 0.0540 \tag{1.39}$$

To verify the accuracy of the approximation, the image below reports the binomial distribution compared to the Gaussian function.

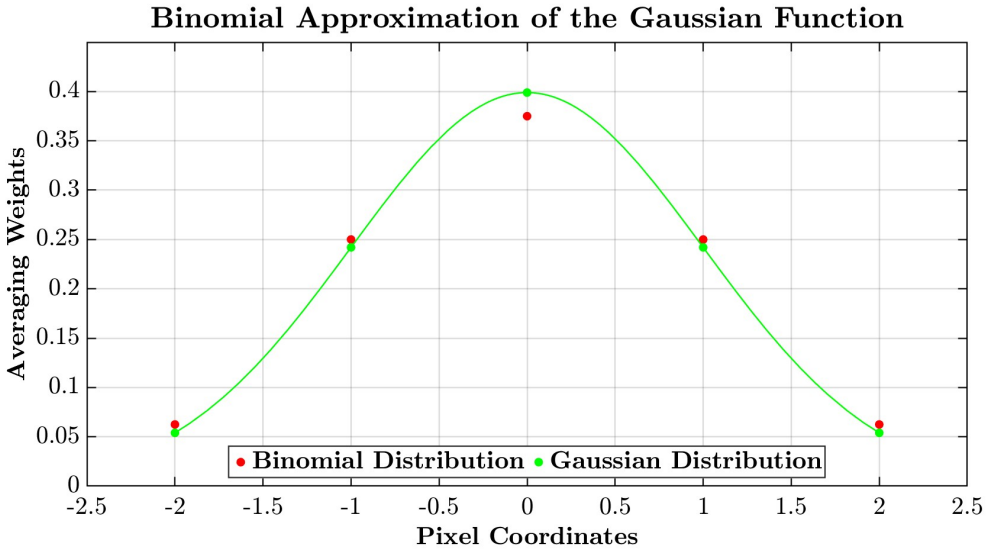


Figure 1.21: Binomial Approximation of the Univariate Gaussian Function

1.3.2 Derivative Kernel

Several among the algorithms implemented require gradients of the captured functions. However, since the collected data are discrete signals, derivatives cannot be computed analytically. Therefore, the gradients must be approximated from the discrete values of $f|_\Omega[x, y]$ using a derivative operator called derivative kernel. This section is dedicated to the definition of common kernels used in computer vision.

Given a continuous univariate function $f(x), x \in \mathbb{R}$ and its discrete representation $f|_\Omega(x), x \in \Omega \subset \mathbb{N}$, the easiest approximation of the first-order derivative of the discrete function $f|'_\Omega(x_o)$ evaluated for $x = x_o$ can be defined as the incremental ratio:

$$f'|_\Omega(x_o) \approx f|_\Omega(x_o) - f|_\Omega(x_o - 1) \tag{1.40}$$

Or similarly, using symmetrical first-order difference:

$$f'|_\Omega(x_o) \approx \frac{f|_\Omega(x_o + 1) - f|_\Omega(x_o - 1)}{2} \tag{1.41}$$

The last operation may be expressed equivalently using the discrete convolution product:

$$f'|_\Omega(x_o) \approx \tilde{K} * f|_\Omega(x_o) := \sum_{i=-1}^{1} \tilde{K}[i]\ f|_\Omega(x_o - i) \tag{1.42}$$

Where $\tilde{K}[x]$ is the derivative kernel or convolution matrix for an univariate function:

$$\tilde{K}[-1] = \frac{1}{2}, \tilde{K}[0] = 0, \tilde{K}[1] = -\frac{1}{2}, \tag{1.43}$$

The same principle is applied to evaluate the gradient of functions with two variables.

In the image below, the most common sequence of operations for calculating the vertical gradient at the point C is presented. Given, for example, a patch 5×5 around the central pixel C, a univariate smoothing filter is applied in the perpendicular direction to reduce noise. Then, a derivative kernel computes the derivative in the vertical direction.

		↓			↓ 2) *Vertical Gradient*
O	O	O	O	O	← 1) *Compute Average*
O	O	O	O	O	← 1) *Compute Average*
O	O	C	O	O	← 1) *Compute Average*
O	O	O	O	O	← 1) *Compute Average*
O	O	O	O	O	← 1) *Compute Average*

Figure 1.22: Computation of the Vertical Gradient at C

This sequence of operations is formalized as follows. For a continuous bivariate function $f(x, y)$ and its discrete representation $f|_\Omega(x, y), (x, y) \in \Omega \subset \mathbb{N}^2$, the horizontal and vertical gradients $f_x|_\Omega(x_o, y_o)$ and $f_y|_\Omega(x_o, y_o)$ can be approximated as

the discrete convolution product with a derivative kernel:

$$f_x|_\Omega(x_o, y_o) = \tilde{K}_x * f|_\Omega(x_o, y_o) := \sum_{i,j=-n}^{n} \tilde{K}_x[i,j]\ f|_\Omega(x_o - i, y_o - j) \tag{1.44}$$

$$f_y|_\Omega(x_o, y_o) = \tilde{K}_y * f|_\Omega(x_o, y_o) := \sum_{i,j=-n}^{n} \tilde{K}_y[i,j]\ f|_\Omega(x_o - i, y_o - j) \tag{1.45}$$

The most common derivative kernel is the $(2n+1) \times (2n+1) = 3 \times 3$ Sobel kernel [6]:

$$\tilde{K}_x = \begin{bmatrix} 0.1250 & 0 & -0.1250 \\ 0.2500 & 0 & -0.2500 \\ 0.1250 & 0 & -0.1250 \end{bmatrix}, \tilde{K}_y = \begin{bmatrix} 0.1250 & 0.2500 & 0.1250 \\ 0 & 0 & 0 \\ -0.1250 & -0.2500 & -0.1250 \end{bmatrix}, \tilde{K}_y = \tilde{K}_x^T \tag{1.46}$$

The Sobel kernel is the generalization of the univariate kernel $\tilde{K}$ defined before 1.43. In fact, it is the result of the matrix product of the binominal smoothing weights $\tilde{w}_B$ and the derivative kernel $\tilde{K}$. This operation smoothens the function $f|_\Omega$ before taking the derivative, reducing the sensibility to local noise:

$$\tilde{K}_x = \tilde{w}_B\ \tilde{K}^T = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ \frac{1}{4} \end{bmatrix} \left[\tfrac{1}{2}, 0, -\tfrac{1}{2}\right] = \begin{bmatrix} 0.1250 & 0 & -0.1250 \\ 0.2500 & 0 & -0.2500 \\ 0.1250 & 0 & -0.1250 \end{bmatrix} \tag{1.47}$$

Despite being the most common derivative kernel, the algorithms implemented in this experiment use the 5×5 Savitzky-Golay kernel [33]. As in Sobel, the Savitzky-Golay kernel is composed of a univariate derivative kernel and a smoothing filter, acting in the perpendicular direction. The implemented filter is the binomial smoothing weights w_B, while the derivative kernel will be derived in the following pages.

Given a function $h(x)$, the Savitzky-Golay method aims to approximate its derivative $h^s(0)$ from an odd set of known points $h(k), k \in \{-n, ..., n\}$. The values $h(k)$ are used to define a polynomial function $h_p(x), x \in [-n; n]$ of order m that will be used to approximate $h(x)$ in a neighbourhood of $x = 0$. The order $m < 2n + 1$ ensures that there is at least one polynomial function that passes through all points.

$$h_p(x) := \sum_{j=0}^{m} b_j\ x^j = b_o + b_1 x^1 + ... + b_m x^m, x \in [-n, n] \tag{1.48}$$

The first derivative of the polynomial function $h_p(x)$ is:

$$h'_p(x) = 0 + b_1 + 2b_2 x + ... + m b_m x^{m-1} \tag{1.49}$$

The second derivative is:

$$h''_p(x) = 0 + 0 + 2b_2 + ... + m(m-1) b_m x^{m-2} \tag{1.50}$$

And so on until the m^{th} derivative:

$$h_p^m(x) = m!\, b_m \tag{1.51}$$

Hence, the s^{th} derivative calculated for $x = 0$, is:

$$h_p(0) = s!\, b_s \tag{1.52}$$

The coefficients $b_j, \forall j \in \{0, ..., m\}$ are found using the least squares criterion. The correct values minimize the sum of the squared differences between the polynomial function $h_p(x)$ and the original function $h(x)$, evaluated at the $2n+1$ known points:

$$\frac{d}{db_j}\left[\sum_{k=-n}^{n}\left(h_p(k) - h(k)\right)^2\right] = 0,\ j \in \{0, ..., m\} \tag{1.53}$$

Which is a set of $m+1$ equations for the $m+1$ unknowns b_j. For example, minimizing with respect to b_0:

$$\frac{d}{db_0}\left[\sum_{k=-n}^{n}\left(h_p(k) - h(k)\right)^2\right] = 0 \tag{1.54}$$

Introducing the definition of $h_p(x)$:

$$\frac{d}{db_0}\left[\sum_{k=-n}^{n}\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)^2\right] = 0 \tag{1.55}$$

The derivative of a linear combination of functions is the linear combination of the derivatives:

$$\sum_{k=-n}^{n}\left[\frac{d}{db_0}\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)^2\right] = 0 \tag{1.56}$$

The derivative of a squared function is $\frac{d(f^2)}{dx} = 2ff'$:

$$\sum_{k=-n}^{n} 2\left[\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)\frac{d}{db_0}\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)\right] = 0 \tag{1.57}$$

The derivatives of the coefficients are $\frac{db_j}{db_0} = \delta_{0j}$, where δ_{0j} is the Kronecker delta, which is 1 if the two indices coincide and 0 otherwise. Therefore, only $\frac{db_0}{db_0} = 1$ while the other terms go to zero:

$$2\sum_{k=-n}^{n}\left[\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)\quad 1\right] = 0 \tag{1.58}$$

Similarly, minimizing with respect to a generic b_r:

$$\frac{d}{db_r}\left[\sum_{k=-n}^{n}\left(h_p(k) - h(k)\right)^2\right] = 0 \tag{1.59}$$

Following the same steps, this time $\frac{d(b_rk^r)}{db_r} = k^r$ and the other terms in the derivative are 0:

$$2\sum_{k=-n}^{n}\left[\left(b_o + b_1k^1 + ... + b_mk^m - h(k)\right)\quad k^r\right] = 0 \tag{1.60}$$

Introducing the sum operator:

$$2 \sum_{k=-n}^{n} \left[\left(\sum_{j=0}^{m} b_j \; k^j - h(k) \right) \;\; k^r \right] = 0 \tag{1.61}$$

Distributing k^r:

$$2 \sum_{k=-n}^{n} \left[\left(\sum_{j=0}^{m} b_j \; k^{j+r} - h(k) \; k^r \right) \right] = 0 \tag{1.62}$$

Bringing $h(k) \; k^r$ to the right side and dividing by 2:

$$\sum_{k=-n}^{n} \sum_{j=0}^{m} \left(b_j \; k^{j+r} \right) = \sum_{k=-n}^{n} \left(h(k) \; k^r \right) \tag{1.63}$$

Interchange the summations on the left:

$$\sum_{j=0}^{m} \sum_{k=-n}^{n} \left(b_j \; k^{j+r} \right) = \sum_{k=-n}^{n} \left(h(k) \; k^r \right) \tag{1.64}$$

b_j is independent of k:

$$\sum_{j=0}^{m} b_j \left(\sum_{k=-n}^{n} (k^{j+r}) \right) = \sum_{k=-n}^{n} \left(h(k) \; k^r \right) \tag{1.65}$$

$\sum_{k=-n}^{n} (k^{j+r})$ and $\sum_{k=-n}^{n} \left(h(k) \; k^r \right)$ are composed only of known quantities, since they are formed by $h(k)$, by the coefficient r and by integers k. Imposing:

$$S_{j+r} = \sum_{k=-n}^{n} (k^{j+r}) \tag{1.66}$$

$$H_r = \sum_{k=-n}^{n} \left(h(k) \; k^r \right) \tag{1.67}$$

An equation with $b_j, j \in \{0, ..., m\}$ as unknowns is obtained:

$$\sum_{j=0}^{m} b_j S_{j+r} = H_r \tag{1.68}$$

The same minimization is repeated for every coefficient $b_j, \forall j \in \{0, ..., m\}$ in order to obtain an equal number of equations and unknowns. After solving the minimization problem, the first derivative $h'_p(0)$ can be computed. Taking into account $2n+1 = 5$ known points, the numerical solution is equivalent to a discrete convolution product with the derivative kernel:

$$h'(0) \approx K_{sg} * h(0) = \sum_{i=-n}^{n} K_{sg}[i] \; h(0 - i) \tag{1.69}$$

$$K_{sg}[-2] = 1, K_{sg}[-1] = 2, K_{sg}[0] = 0, K_{sg}[1] = -2, K_{sg}[2] = -1 \tag{1.70}$$

The derivative kernel may be extended to gradients in two variables, using the binomial smoothing kernel w_B as filter. The complete Savitzky-Golay kernel to compute the horizontal gradients becomes:

$$K_{SG} = w_B \; K_{sg}^T = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} \tag{1.71}$$

The kernel is normalized with respect to its maximum value:

$$\tilde{K}_{SG} = \frac{K_{SG}}{\|K_{SG}\|_\infty}, \;\; \|K_{SG}\|_\infty = \max_{i,j \in \{-n,\ldots,n\}} |K_{SG}| \tag{1.72}$$

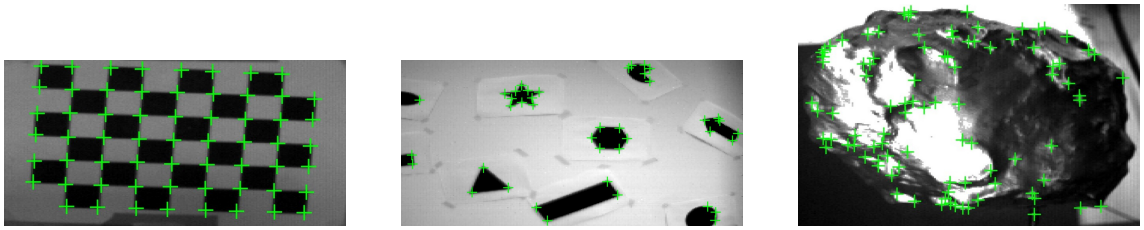
Chapter 2

Feature Detection and Tracking

This chapter is dedicated to explaining how a system can be commanded to autonomously detect and track features on a video recorded by an event camera. As mentioned in the Introduction 1, these two operations are the 1^{st} and 2^{nd} steps that must be performed to determine how a camera moves in space. In fact, these codes have the important task of transforming raw observations into data that may be used to gain knowledge regarding motion.

As mentioned in the chapter Event Generation 1.1.1, events are asynchronous and localized detections of motion, but not all detections are useful for understanding the rigid motion of a dynamic object. In fact, the majority of pose estimation algorithms concentrate on studying only peculiar points in the structure, called "features".

By definition, features are points in a scene that are very different from their surroundings. This characteristic makes them the easiest points to follow in a video, either made with a traditional or event camera. Every algorithm in computer vision takes advantage of the motion of these points to understand how the camera moves. In the following image, a visual representation of features in a scene is presented.



(a) Checkerboard (b) Geometric Shapes (c) $67P/C-G$

Figure 2.1: Examples of Features in an Image

In particular, a feature detector is an algorithm that automatically identifies the position of peculiar points in the scene, while a tracker or matcher is responsible for connecting detections of the same point in various frames. Feature tracking and matching are not synonyms, although they have the same objective of connecting detections: feature matchers compare the surroundings of each detection to understand if they represent the same feature, while feature trackers propagate the motion of a feature across successive views. This procedure is standardized for traditional cameras, but due to the different nature of acquisitions, the algorithms need to be completely re-invented to be adapted to a stream of events.

This section will describe several feature detection and tracking algorithms for events. Although feasible in theory, feature matching is not a common practice due to the very nature of events. Being asynchronous and localized detections, it is easier to track events instead of matching them.

The algorithms will be applied to a couple of videos captured by an event camera. One video contains geometric shapes, while the other contains the objective of this experiment, that is, the realistic scenario of a satellite orbiting around a small celestial object, which in this case is the comet $67P/Churyumov - Geramisenko$. The results provided by the feature trackers will be used to determine the pose of the camera and the point-cloud distribution of the scene in the third chapter 3.

On a final note, a warning. Detecting and tracking features might appear obvious the first time. This is because humans distinguish shapes and movements without any particular difficulties. However, the device must be able to perform the operations on its own, and that is where difficulties arise.

.

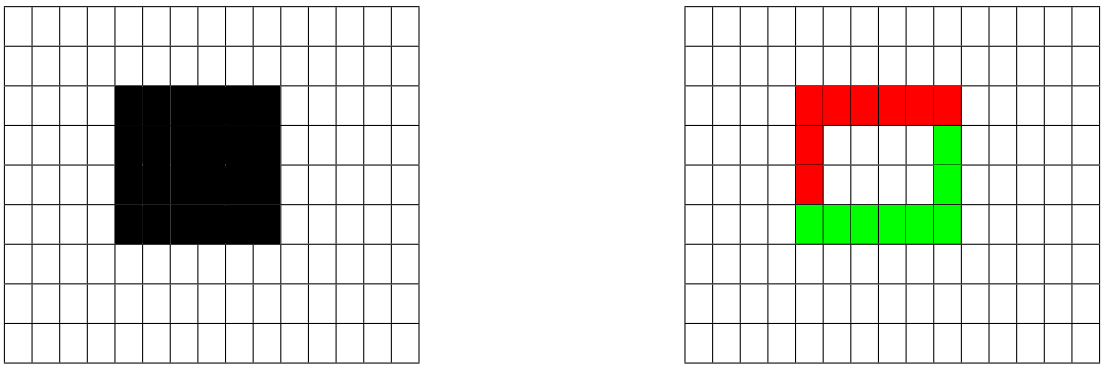
2.1 Event Only Procedure

2.1.1 SAE* Surfaces of Active Events

As mentioned above 2, features are identified because they are different from their surroundings. However, to distinguish features from generic events, there must be a structure that allows one to compare the two. This structure will act similarly to frames in a traditional video while still maintaining the asynchronous nature of events.

The most common structure in event cameras is the time surface, or SAE surface of active events [1]. Several algorithms, including those in this first step of the procedure, use time surfaces as a baseline to compare events and determine which event is a feature and which is not. In this section, the formal definition of SAE* is presented, where " * " represents an improved variant. Prior to the formal definition, an intuitive demonstration of the working principle of a generic SAE will be presented to explain how these surfaces help to transform a stream of events into reliable structures for feature detectors.

Consider, for example, the image of a black rectangle on a white background; this rectangle is moving from the top left corner of the frame to the bottom right. Events $e_k = \{p_k, t_k, x_k, y_k\}, k \in \{1, ..., n\}$ will occur due to the motion of this object, because pixels change from pointing a white background to pointing an object with a different index of reflectivity. In the image, green represents a positive variation in brightness ($p_k = +1$) while red represents a negative variation ($p_k = -1$). Given that the object is moving to the bottom right corner, the edges in the direction of motion will be green, while the remaining will be red.



(a) Representation of the Image (b) Representation of the Events

Figure 2.2: Visual Representation of the Scene Created by a Moving Rectangle

As the rectangle continues to move, it will continue to trigger new events. The time surfaces collect the time t_k of the latest event that was triggered in each pixel. In particular, two surfaces have to be created: one for positive polarity and one for negative polarity. This distinction will help avoid erasing features. The image below presents what time surfaces look like. In each pixel, the latest time t_k [μs] is stored:

2	3	4	4	4	4	4	4	0	0	0	0	0	0	0
2	3	4	5	5	5	5	5	5	0	0	0	0	0	0
0	3	4	5	6	6	6	6	6	6	0	0	0	0	0
0	0	4	5	6	0	0	0	0	0	0	0	0	0	0
0	0	0	5	6	0	0	0	0	0	0	0	0	0	0
0	0	0	0	6	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) Negative Time Surface

1	1	1	1	1	2	3	4	0	0	0	0	0	0	0
2	2	2	2	2	2	3	4	5	0	0	0	0	0	0
0	3	3	3	3	3	3	4	5	6	0	0	0	0	0
0	0	4	4	4	4	4	4	5	6	0	0	0	0	0
0	0	0	5	5	5	5	5	5	6	0	0	0	0	0
0	0	0	0	6	6	6	6	6	6	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) Positive Time Surface

Figure 2.3: Visual Representation of the Time Surfaces

From the image, it is clear which are the events that represent peculiar points in the field of view. Even if this is an ideal example, it is useful to understand how different valid corners are with respect to other points in the scene:

4	4	4	4	4
4	5	5	5	5
4	5	6	6	6
4	5	6	0	0
4	5	6	0	0

4	5	6	0	0
5	5	6	0	0
6	6	6	0	0
0	0	0	0	0
0	0	0	0	0

Figure 2.4: Possible Shapes of Corners

The remainder of this section is dedicated to the formal definition of SAE*. The base SAE is a particular case of SAE*, therefore, the definition will be provided directly for the latter.

Given a stream of events $e_k = \{p_k, t_k, x_k, y_k\}, k \in \{1, ..., n\}$, the two time surfaces $S^*_{\oplus}$ and $S^*_{\ominus}$ are built to iteratively collect the values t_r $[\mu s]$, where t_r represents the time in which the latest important event occurred in each pixel. As mentioned above, two surfaces are used for events of different polarities; the distinction reveals two scenarios otherwise unclear: a bright corner in a dark environment produces mostly positive events, whereas the opposite occurs for dark corners.

$$\begin{aligned} S^*_{\oplus k} : \Omega \subset \mathbb{N}^2 &\to \mathbb{R} \\ (x, y)^T &\mapsto t_r \end{aligned} \tag{2.1}$$

$$\begin{aligned} S^*_{\ominus k} : \Omega \subset \mathbb{N}^2 &\to \mathbb{R} \\ (x, y)^T &\mapsto t_r \end{aligned} \tag{2.2}$$

The surfaces $S^*_{\oplus}$ and $S^*_{\ominus}$ are updated after each event $e_k, \forall k$. Therefore, they represent two sequences of functions $S^*_{\oplus k}$ and $S^*_{\ominus k}$. As the name suggests, a sequence of

functions is a succession in which every term is defined as a function. Every time a new event is found from the event camera, the time surface is updated.

$$\forall k \in \{1, ...n\} :$$

$$\begin{cases} S^*_{\oplus\, k}(x,y) = S^*_{\oplus\, k-1}(x,y), \forall (x,y)^T \neq (x_k, y_k)^T \\ \\ S^*_{\oplus\, k}(x_k,y_k) = \begin{cases} t_k & if \;\; p_k = \oplus \;\wedge\; t_k > S^*_{\oplus\, k-1}(x_k,y_k) + \Delta t \\ t_k & if \;\; p_k = \oplus \;\wedge\; S^*_{\ominus\, k-1}(x_k,y_k) > S^*_{\oplus\, k-1}(x_k,y_k) \\ S^*_{\oplus\, k-1}(x_k,y_k) & otherwise \end{cases} \end{cases}$$

$$\begin{cases} S^*_{\ominus\, k}(x,y) = S^*_{\ominus\, k-1}(x,y), \forall (x,y)^T \neq (x_k, y_k)^T \\ \\ S^*_{\ominus\, k}(x_k,y_k) = \begin{cases} t_k & if \;\; p_k = \ominus \;\wedge\; t_k > S^*_{\ominus\, k-1}(x_k,y_k) + \Delta t \\ t_k & if \;\; p_k = \ominus \;\wedge\; S^*_{\oplus\, k-1}(x_k,y_k) > S^*_{\ominus\, k-1}(x_k,y_k) \\ S^*_{\ominus\, k-1}(x_k,y_k) & otherwise \end{cases} \end{cases}$$

$$Where\; S^*_{\oplus\, 0} = S^*_{\ominus\, 0} = -\Delta t \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \tag{2.3}$$

Despite the complex formula, the idea behind 2.3 is actually quite simple. Consider a new and generic event $e_{\bar{k}} = \{p_{\bar{k}}, t_{\bar{k}}, x_{\bar{k}}, y_{\bar{k}}\}, k = \bar{k}$ detected by the event camera. Consider, for example, that the polarity of this event is positive ($p_{\bar{k}} = 1$). Therefore, this event could update the positive surface $S^*_{\oplus}$ (the same happens with a negative event and $S^*_{\ominus}$).

There are three conditions that may occur:

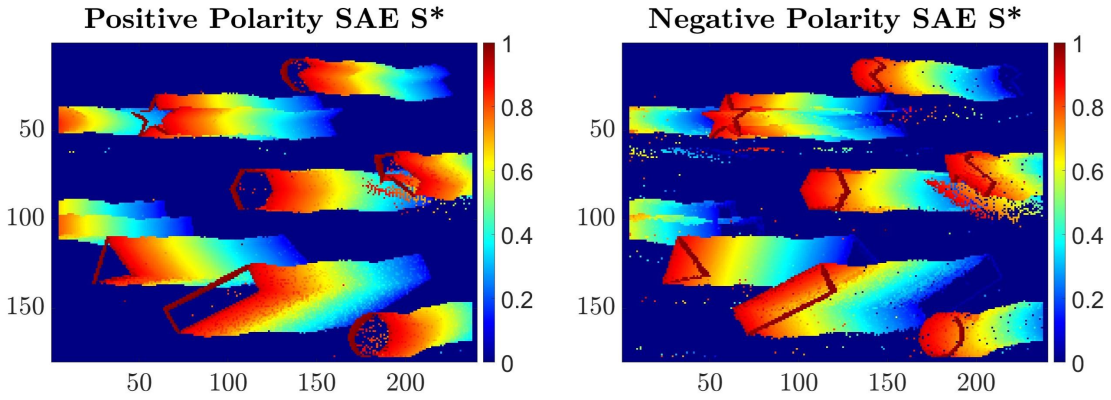
1. If the time of appearance $t_{\bar{k}}$ exceeds the value on the surface by a certain threshold Δt, which is $50ms$ for SAE* and $0ms$ for SAE, the surface is updated with the new value $t_{\bar{k}}$ at the location $(x_k, y_k)^T$. SAE* improves on the base SAE thanks to the threshold Δt, which helps to avoid redundant and noisy events.

   The definition of $S^*_{\oplus\, 0}$ and $S^*_{\ominus\, 0}$ for $k = 0$ implies that all events with $t_k > 0$ can be stored.

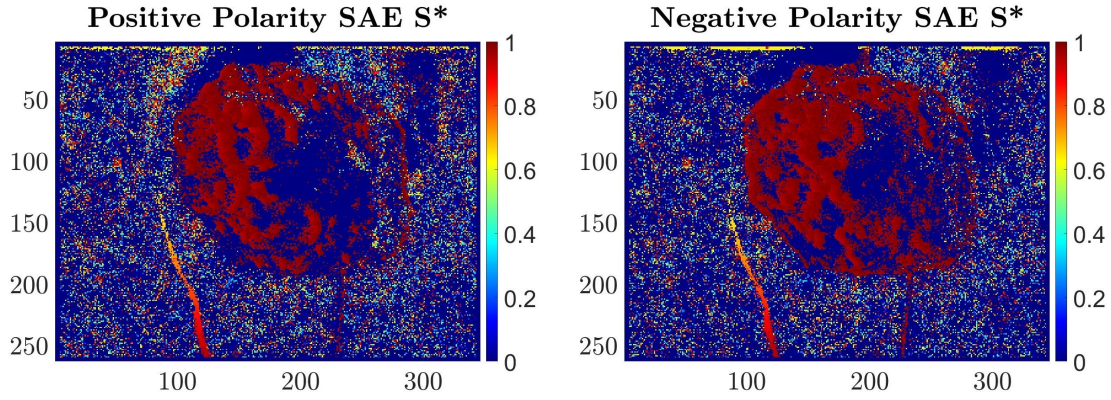
2. If the last event that occurred at the same coordinates $(x_k, y_k)^T$ was negative ($S^*_{\ominus\, k-1}(x_k, y_k) > S^*_{\oplus\, k-1}(x_k, y_k)$), the new event is completely unrelated to the previous events, and the new time $t_{\bar{k}}$ is stored at $(x_k, y_k)^T$.

3. If none of the conditions are satisfied, the time surface remains unchanged.

Examples of time surfaces are presented in the image below. The scene represents videos of geometric shapes and of the rotating comet $67P/Churyumov - Geramisenko$. In the image, red represents recent events, while blue represents older events.



(a) Geometric Shapes



(b) $67P/Churyumov - Geramisenko$

Figure 2.5: Examples of Time Surfaces

2.1.2 FA-Harris: ARC* Candidates' Detection

Among the feature detectors for events, FA-Harris [21] is one of the most complete and reliable among the current research. In fact, FA-Harris is constituted by the union of two prominent detectors for events, united as one to balance performance and accuracy. In the following sections, the two key components of FA-Harris are explained together with additional insights regarding their MATLAB implementation.

The codes implemented by FA-Harris are the ARC* and binarized Harris detectors. The first is dedicated to rapidly selecting possible features among the entire stream of events; this selection is performed with no particular accuracy, as it helps reduce the computational cost of the second part of the algorithm, which is much more accurate. Both ARC* and binarized Harris are adaptations to time surfaces of existing techniques based on traditional images. Respectively, ARC* adapts the FAST detector [1], while binarized Harris [36] adapts the Harris corner detector [14].

This section is dedicated to ARC*, which is based on a geometric evaluation of the time surfaces. As explained above 2.1.1, and as reported in the image below, a valid feature is surrounded by different values that may be divided into two regions: the values of time t_r in the first region are younger / higher than all the elements in the second. In the context of this code, these two regions will be referred to as the "younger arc" and the "older arc". ARC* studies each incoming event and establishes whether the time surface around these points can be divided into these arcs. Events that pass this test are marked as possible candidates and are evaluated by the binarized Harris.

4	4	4	4	4
4	5	5	5	5
4	5	6	6	6
4	5	6	0	0
0	5	6	0	0

4	5	6	0	0
5	5	6	0	0
6	6	6	0	0
0	0	0	0	0
0	0	0	0	0

Figure 2.6: Possible Shapes of Corners

However, ARC* does not employ an entire patch of the time surface to carry out this investigation. In fact, to balance accuracy and computational cost, only Bresenham circles of radius 3 and 4 are used. The image below represents these two sets of points around the central event in red. The yellow points represent the Bresenham circle (also called discrete circle mask or pixel circle) of radius 3 while the green ones represent the circle of radius 4. A more detailed explanation of why these structures are selected is provided at the end of this section.

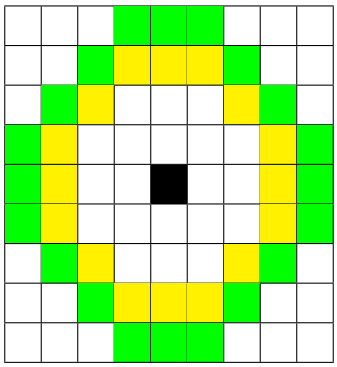


Figure 2.7: Bresenham Circles Around C

Regardless of the shape, the idea of the code remains the same. The code will try to divide each of these circles into the younger arc and the older arc. In the case that it succeeds for both circles, the test is considered complete.

For example, considering a Bresenham circle of radius 3, the image below reports a valid corner candidate. In fact, the group in red ($1\mu s, 2\mu s, 3\mu s$ and $4\mu s$) is the older arc, while the remaining section is the younger arc:

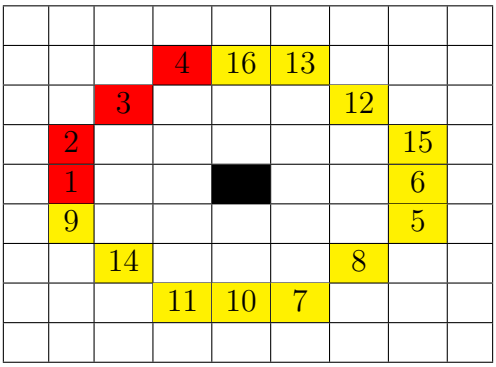


Figure 2.8: Example of a Valid Corner

The remainder of this section is dedicated to the working principle of ARC*.

After a new event $e_{\bar{k}} = \{p_{\bar{k}}, t_{\bar{k}}, x_{\bar{k}}, y_{\bar{k}}\}, k = \bar{k}$ has been identified, the algorithm interrogates the two Bresenham circles around the coordinates $(x_{\bar{k}}, y_{\bar{k}})^T$ and tries to divide both circles into two arcs. At the beginning, the algorithm studies the small circle, because it requires less computational cost. Then, if the event $e_{\bar{k}}$ passes the first test, the second circle is studied. In the end, an event $e_{\bar{k}}$ that passed both tests is considered a corner candidate and is studied by the binarized Harris.

The two tests share the same core concept; therefore, the images will depict the procedure only on the smaller circle. As said, to pass a test, the code must be able to divide the circle into two arcs. However, the size of these two arcs must be in a specific range to consider the distinction valid. For the small circle, which consists of 16 elements, the smallest arc between the younger and older arc must be in the range $[3, 6]$, while for the large circle, which consists of 20 elements, the range is $[4, 8]$. The lower threshold distinguishes valid detections from noise, whereas the upper limit ensures that edges are not identified as corners.

At the beginning of the test, the algorithm identifies the youngest / highest value in the circle and introduces it into the younger arc. In the previous example, $t = 16\mu s$ was the most recent event:

			4	16	13			
		3				12		
	2						15	
	1						6	
	9						5	
		14				8		
			11	10	7			

Figure 2.9: First Element in the Younger Arc

Once the youngest element has been identified, the code examines the two elements next to the first one. In the example, $13\mu s$ and $4\mu s$. The youngest value between the two is introduced inside the younger arc. The same operation is repeated again until the younger arc contains the minimum value that distinguishes noise and valid corners, which is 3 for the small circle and 4 for the large circle. In the example, $t = 13\mu s$ is the younger value between $4\mu s$ and $13\mu s$, while $t = 12\mu s$ is the younger value in the second iteration, between $4\mu s$ and $12\mu s$.

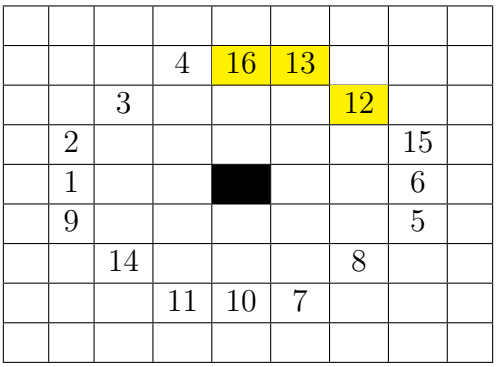


Figure 2.10: Min n.° of Elements in the Younger Arc

Now that the minimum number of elements has been reached, inclusion in the younger arc cannot be so direct. From now on, the code continues to examine the two elements next to the arc but does not automatically insert the higher value in the younger arc. Instead, two scenarios may occur:

1. If the higher value between the two is younger than the oldest value in the younger arc, it can be safely introduced into the younger arc. The successive iterations continue in the same manner, using the two elements next to the arc.

   In the example, $t = 15\mu s$ is the younger value between $4\mu s$ and $15\mu s$. In addition, $15\mu s$ is higher than the oldest element in the arc, which is $12\mu s$, and therefore $15\mu s$ can be added to the younger arc:

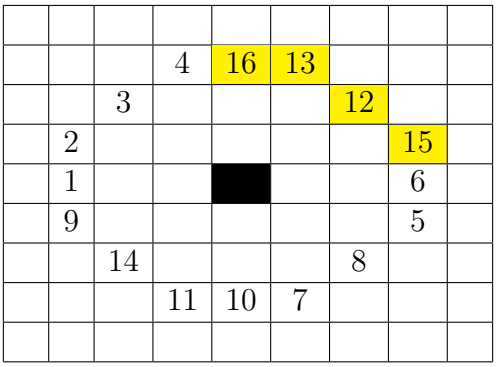


Figure 2.11: Case 1: Event Immediately Added

2. If the younger value between the two is older than the oldest value in the younger arc, the element is not added to the arc. The algorithm skips the addition and continues with a new couple of elements. For example, $t = 6\mu s$ is the younger value between $4\mu s$ and $6\mu s$, but it is older than the oldest time in the younger arc, which is $12\mu s$:

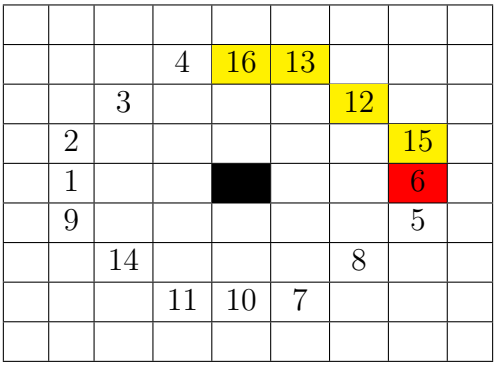


Figure 2.12: Case 2: Event Skipped

One of these two cases may follow the second scenario:

(a) If the younger value between the two is younger than the oldest value in the younger arc for at least one of the successive iterations: all the elements on the side of this younger value are added to the younger arc. The oldest value in the younger arc is updated to be the oldest value added to the group, and the cycle restarts. In the example, $6, 5, 8, 7, 10, 11\mu s$ are all older than the oldest value $12\mu s$, so they are included in the younger arc only when $14\mu s$ appears. The new oldest value in the arc becomes $5\mu s$.

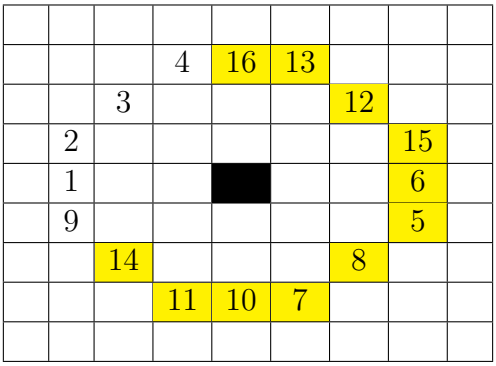


Figure 2.13: Case 2a: Events Added Later

(b) If the younger value between the two is older than the oldest value in the younger arc for all the remaining iterations, the cycle ends. In the example, $9\mu s$ is introduced in the younger arc without problems, but the remaining values $4, 3, 2, 1\mu s$ are all older than the oldest value in the arc, which is $5\mu s$. The remaining elements outside the younger arc constitute the older arc.

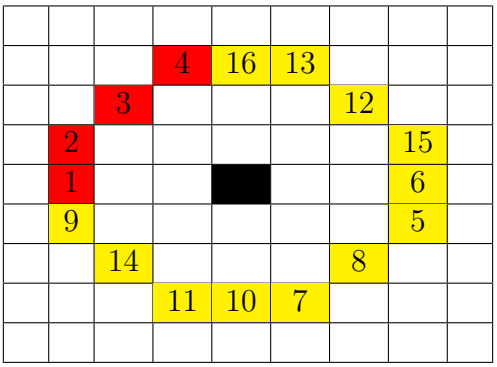
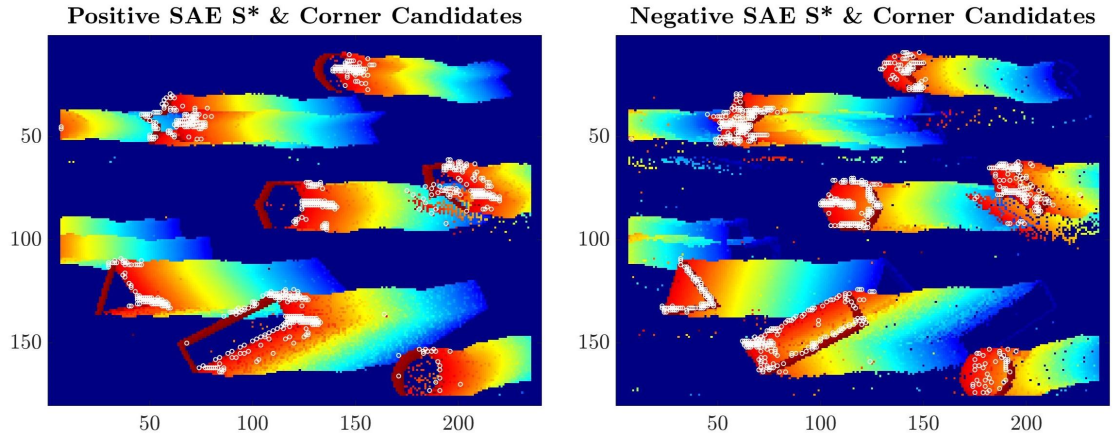


Figure 2.14: Case 2b: End of the Loop

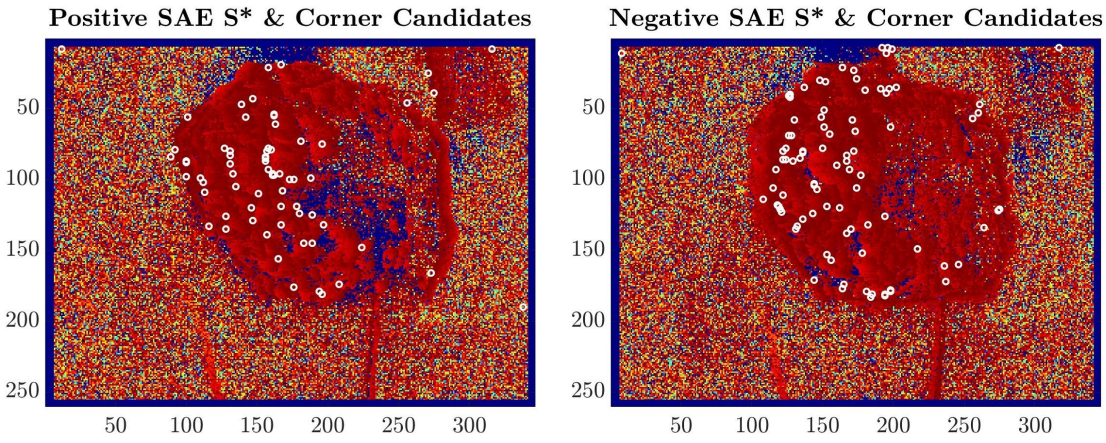
At the end of the loop, the event $e_{\bar{k}}$ is classified as a possible corner or discarded. As mentioned above, the distinction is based on the smaller number of elements in the arcs. In the example, the event was indeed a possible corner because the smaller arc contained 4 elements, which is in the range $4 \in [3, 6]$.

The ARC* candidate selection results in a simple and fast detector in both MATLAB and other programming languages. In fact, it relies only on geometric evaluations, which do not require many calculations. However, it is not as precise as other methods and there are false positives very often. For this reason, it is paired to binarized Harris, which is the opposite.

In the following images, ARC* is applied to the videos of the geometric shapes and the comet $67P/Churyumov-Geramisenko$. The results show the time surfaces of the videos as background. White circles are placed around the identified candidate corners. As the images show, the algorithm performs well in both scenes, especially in the first because of its distinct features.



(a) Geometric Shapes



(b) $67P/Churyumov-Geramisenko$

Figure 2.15: Time Surfaces and Corner Candidates

Two additional observations must be made regarding the algorithm:

1. Why does the algorithm employ a discrete circle of radius 3 and 4?

   The Bresenham circles are selected to balance accuracy with computational cost. A smaller set of points would be too noisy for a precise estimate, while a larger one would require higher computational costs without improving accuracy.

2. How does the code examine the two elements next to the previous ones?

   This operation is peculiar and deserves to be mentioned. When Bresenham circles are represented visually, they appear as a discrete circle in two dimensions. However, the code stores its values in a vector because it is simpler to access and reduces the required allocated memory. By definition, the values must be extracted so that the first value corresponds to the top of the circle, while the others follow in the clockwise direction. In the example, the resulting vector is:

   $$t = \big[16, 13, 12, 9, 6, 5, 8, 7, 10, 11, 14, 15, 1, 2, 3, 4\big] \tag{2.4}$$

   The youngest value t_{max} among the set is identified using max in MATLAB. Once the highest value is known, its position in the vector can be identified using the command $find\,(t = t_{max})$.

   The two elements next to the first one are elegantly computed using circular permutation. Given the size of the circle n and the position of an element a in the circle, the position of the two elements a_{CCW} and a_{CW} (respectively the first element next to a in the counter-clockwise and clockwise directions) can be found using the remainder with the same sign of the divisor (mod):

   $$\begin{gathered} mod(x, y) := x - y\lfloor\tfrac{x}{y}\rfloor, \;\; Where\lfloor\tfrac{x}{y}\rfloor = floor(\tfrac{x}{y}) \\ a_{CW} = mod(a, n) + 1 \\ a_{CCW} = mod(a - 2, n) + 1 \end{gathered} \tag{2.5}$$

   Once the position of the two elements is known, the values can be retrieved from the vector t. Taking, for example, $a = 1$ and $n = 16 \implies a_{CW} = mod(1, 16)+1 = 1+1 = 2$ and $a_{CCW} = mod(1-2, 16)+1 = mod(-1, 16)+1 = 15 + 1 = 16$

NB: These formulations are valid only for MATLAB, due to its indexing starting from 1 instead of 0. In other programming languages:

$$\begin{gathered} a_{CW} = mod(a + 1, n) \\ a_{CCW} = mod(a - 1 + n, n) \end{gathered} \tag{2.6}$$

2.1.3 FA-Harris: Binarized Harris Corners' Detection

This section describes the second algorithm contained in the FA-Harris detector [21]. The code receives the ARC* candidates as input and finalizes the identification of corners through the binarized Harris detector. The Harris approach evaluates candidates based on similarity [14]: by definition, features are very different from their surroundings 2. In the code, the Harris method is applied to a binarized version of the time surface, hence the name. The chapter is divided into two parts: the first part contains an example to describe the idea behind the code, while the second part gives a formal description of the approach.

Consider an event $e_{\bar{k}} = \{p_{\bar{k}}, t_{\bar{k}}, x_{\bar{k}}, y_{\bar{k}}\}, k = \bar{k}$ that passed the ARC* test. Consider also a $(2n+1) \times (2n+1)$ patch of the time surface $S^*_{\bar{k}}$ with the same polarity as $e_{\bar{k}}$ around its coordinates $(x_{\bar{k}}, y_{\bar{k}})^T$:

$$S^*_{\bar{k}}[x, y], [x, y]^T \in \{x_{\bar{k}} - n, ..., x_{\bar{k}} + n\} \times \{y_{\bar{k}} - n, ..., y_{\bar{k}} + n\} \tag{2.7}$$

In the table below, a 9×9 patch of the time surface $[\mu s]$ is presented. The central element of this patch, which occurred at $5630246\mu s$, was identified during the experiments as a valid feature.

0000000	0000000	4761404	3802765	1145022	5260907	3802882	4892323	5273312
5624318	0000000	0000000	0000000	4970385	0000000	4230114	4273009	4184253
4412498	4681606	4462060	2983270	4023288	4063320	4293309	4142478	5630144
5560729	5459528	4359635	3963020	5581503	5582232	4550935	4362907	5619975
5549060	5561620	5624589	5581167	5630246	5520734	5577118	5552208	5580797
5561087	5442105	4424701	5560591	5521649	5521521	5547288	5520452	5548001
5368703	5338643	5309840	5458392	5598546	5359322	5341395	5361468	5478856
5446310	5301790	5261757	5476797	5416801	5273450	5456167	5418244	5240234
5265689	5264298	5195189	5366061	5341913	5309950	5342627	5320742	5322937

Figure 2.16: Example of Time Surface Around a Valid Corner

The algorithm computes a binary surface starting from this patch. A value of 1 is assigned to the $m = 25$ highest values, while the remaining elements are defined as 0. This sharp distinction highlights the presence of patterns in the patch.

In the example, it is clear that the central part of the patch is different from the rest. In fact, the patch is divided into two, and most of the events are condensed in the center.

0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figure 2.17: Example of Binarized Surface Around a Valid Corner

In the image below, two other examples of binarized surfaces are presented: one related to a generic point, while the other associated to a corner. In the former surface, events are randomly distributed, while in the latter most events gather on the right side.

1	1	1	0	0	0	0	0	1
1	0	1	0	0	0	0	1	0
0	1	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	1	0
1	1	0	1	0	0	0	0	0

(a) Discarded Event

0	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0
0	0	1	0	0	0	1	1	1
0	0	0	0	1	1	0	1	1
0	0	0	1	0	1	1	1	1
0	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0

(b) Valid Feature

Figure 2.18: Additional Examples of Binarized Surfaces

The remainder of this section provides the formal procedure implemented in the code.

Given an event $e_{\bar{k}} = \{p_{\bar{k}}, t_{\bar{k}}, x_{\bar{k}}, y_{\bar{k}}\}, k = \bar{k}$ and the $(2n+1) \times (2n+1)$ patch of the time surface $S^*_{\bar{k}}$ with the same polarity $p_{\bar{k}}$ around $(x_{\bar{k}}, y_{\bar{k}})^T$:

$$S^*_{\bar{k}}[x, y], [x, y]^T \in \{x_{\bar{k}} - n, ..., x_{\bar{k}} + n\} \times \{y_{\bar{k}} - n, ..., y_{\bar{k}} + n\} \tag{2.8}$$

For simplicity, from now on $S^*_{\bar{k}}$ will be referred to as $A = a_{ij}, A \in \mathbb{R}^{n \times n}$.

The binarized surface $B = b_{ij},\ b_{ij} \in \{0, 1\}\ \forall i, j$ is defined as follows:

Let the vector $C = c_i,\ C \in \mathbb{R}^{n^2}$ be the vector that contains all the elements of A sorted from the largest to the smallest: $c_1 \geq c_2 \geq ... \geq c_{n^2}$. Select the element c_m of the vector (for a 9×9 patch, $m = 25$). This element becomes the threshold for the definition of B:

$$\forall i, j \in \{1, ..., n\} \times \{1, ..., n\} :$$

$$b_{ij} = \begin{cases} 1 & if\ a_{ij} \geq c_m \\ 0 & if\ a_{ij} < c_m \end{cases} \tag{2.9}$$

A similar but equivalent operation is performed in MATLAB using $maxk$. $maxk$ identifies the location of the m highest values in a matrix and returns the linear indices that represent them. A linear index represents the position of an element in a matrix using only one coordinate, which is equivalent to flattening the matrix to a column vector.

To determine whether the event $e_{\bar{k}}$ is a feature or not, the Harris algorithm implements a similarity function $s(x, y, \delta x, \delta y)$, which describes how similar the local patch around the point $(x_{\bar{k}}, y_{\bar{k}})^T$ is to other patches. The similarity function is defined as:

$$s(x, y, \delta x, \delta y) = \sum_{i,j=-n}^{n} G[i, j]\ (B[x + \delta x - i, y + \delta y - j] - B[x - i, y - j])^2 \tag{2.10}$$

In the formula:

1. B is the binarized surface.
2. $G[x, y]$ is the Gaussian filter that emphasizes the central pixel values.
3. $\sum_{i,j}(...)[x - i, y - j]$ represents a sum of all the values in a $(2n+1) \times (2n+1)$ patch around the point $[x, y]$.
4. $\sum_{i,j}(...)[x+\delta x-i, y+\delta y-j]$ represents the sum of all values in a patch around the point $[x + \delta x, y + \delta y]$.

Therefore, the function evaluates how similar the current patch is to a patch shifted by $[\delta x, \delta y]$. Large values of $s\ \forall[\delta x, \delta y]$ imply the uniqueness of the feature.

The term $B[x+\delta x-i, y+\delta y-j]$ may be expanded around $[x-i, y-j]$ using the first term of the Taylor series:

$$B[x+\delta x-i, y+\delta y-j] \approx B[x-i, y-j]+B_x[x-i, y-j]\delta x+B_y[x-i, y-j]\delta y \quad (2.11)$$

Where B_x and B_y are, respectively, the x and y gradients of B. Therefore:

$$s(x, y, \delta x, \delta y) \approx \sum_{i,j=-n}^{n} G[i, j]\ (B_x[x-i, y-j]\delta x + B_y[x-i, y-j]\delta y)^2 \quad (2.12)$$

Expanding the square:

$$\begin{aligned} s(x, y, \delta x, \delta y) \approx \textstyle\sum_{i,j=-n}^{n} G[i, j]\ (B_x[x-i, y-j]^2\delta x^2 + \\ +2B_x[x-i, y-j]B_y[x-i, y-j]\delta x\delta y + B_y[x-i, y-j]^2\delta y^2) \end{aligned} \quad (2.13)$$

Applying the distributive property of the sum and product:

$$\begin{aligned} s(x, y, \delta x, \delta y) \approx \textstyle\sum_{i,j=-n}^{n} G[i, j]\ B_x[x-i, y-j]^2\delta x^2 + \\ +2\textstyle\sum_{i,j=-n}^{n} G[i, j]\ B_x[x-i, y-j]B_y[x-i, y-j]\delta x\delta y + \\ +\textstyle\sum_{i,j=-n}^{n} G[i, j]\ B_y[x-i, y-j]^2\delta y^2 \end{aligned} \quad (2.14)$$

This could be expressed in quadratic form as follows:

$$s(x, y, \delta x, \delta y) \approx (\delta x, \delta y) M \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} \quad (2.15)$$

Where M is the autocorrelation or second-moment matrix:

$$M = \begin{bmatrix} \sum_{i,j=-n}^{n} G[i, j]\ (B_x[x-i, y-j]^2 & \sum_{i,j=-n}^{n} G[i, j]\ B_x[x-i, y-j]B_y[x-i, y-j] \\ \sum_{i,j=-n}^{n} G[i, j]\ B_x[x-i, y-j]B_y[x-i, y-j] & \sum_{i,j=-n}^{n} G[i, j]\ B_y[x-i, y-j]^2 \end{bmatrix} \quad (2.16)$$

M could also be expressed using the definition of discrete convolution:

$$\begin{aligned} F * H &:= \textstyle\sum_{i,j} F[i, j]H[x-i, y-j] \\ M &= \begin{bmatrix} G * B_x^2 & G * B_xB_y \\ G * B_xB_y & G * B_y^2 \end{bmatrix} \end{aligned} \quad (2.17)$$

Therefore, M is calculated as a convolution of the Gaussian kernel G and the gradients of B. The x and y gradients of B are calculated using the Savitzky-Golay kernel as follows:

$$\begin{aligned} B_x[x, y] &= \tilde{K}_{SG} * B[x, y] \\ B_y[x, y] &= \tilde{K}_{SG}^T * B[x, y] \end{aligned} \quad (2.18)$$

As mentioned above, a valid corner $(x, y)^T$ is characterized by a large value of $s(x, y, \delta x, \delta y), \forall(\delta x, \delta y)^T$. Instead of computing the similarity function in any direction, the Harris algorithm studies the eigenvalues of the autocorrelation matrix M. In fact:

1. The similarity function s is described as a quadratic form of the matrix M.
2. M is symmetric.
3. M is semidefinite positive (from Sylvester's criterion, a 2×2 symmetric matrix with $M_{11} \geq 0$ and $det(M) \geq 0$ is semidefinite positive).

Thus, M may be decomposed as follows:

$$M = Q \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q^T \tag{2.19}$$

Where Q is the orthogonal matrix of the eigenvectors $Q^T = Q^{-1}$, and λ_1 and λ_2 are the eigenvalues of M, with $\lambda_1, \lambda_2 \geq 0$.

Changing coordinates:

$$\begin{pmatrix} \delta x' \\ \delta y' \end{pmatrix} = Q^T \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} \tag{2.20}$$

s becomes:

$$s(x, y, \delta x', \delta y') \approx (\delta x', \delta y') \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{pmatrix} \delta x' \\ \delta y' \end{pmatrix} \tag{2.21}$$

Which implies:

$$s(x, y, \delta x', \delta y') \approx \lambda_1 \delta x'^2 + \lambda_2 \delta y'^2 \tag{2.22}$$

Therefore, λ_1 and λ_2 represent the principal curvatures of the similarity function s. From the last expression, the following observations can be made:

1. If both λ_1 and λ_2 are small, the similarity s is small for any arbitrary shift $\forall[\delta x', \delta y']$, implying that the event $e_{\bar{k}}$ is not a feature.
2. If one eigenvalue is small and the other is large, only shifts in a certain direction produce a variation. Therefore, the event $e_{\bar{k}}$ is an edge, as large values of s are produced only perpendicular to the edge.
3. if both λ_1 and λ_2 are large, any shift produces a large similarity $\forall[\delta x', \delta y']$, implying that the event $e_{\bar{k}}$ is a corner.

Solving an eigenvalue problem for every event would require too much computational cost. Instead, the Harris corner detector defines the "corner strength" to establish if a candidate $(x, y)^T$ is a feature. The Harris strength is defined as:

$$C^H(x, y) = det(M) - k\, tr(M),\ k = 0.04 \tag{2.23}$$

The operator elegantly avoids solving the eigenvalue problem because:

1. $det(M) = \lambda_1 \lambda_2$
2. $tr(M) = \lambda_1 + \lambda_2$

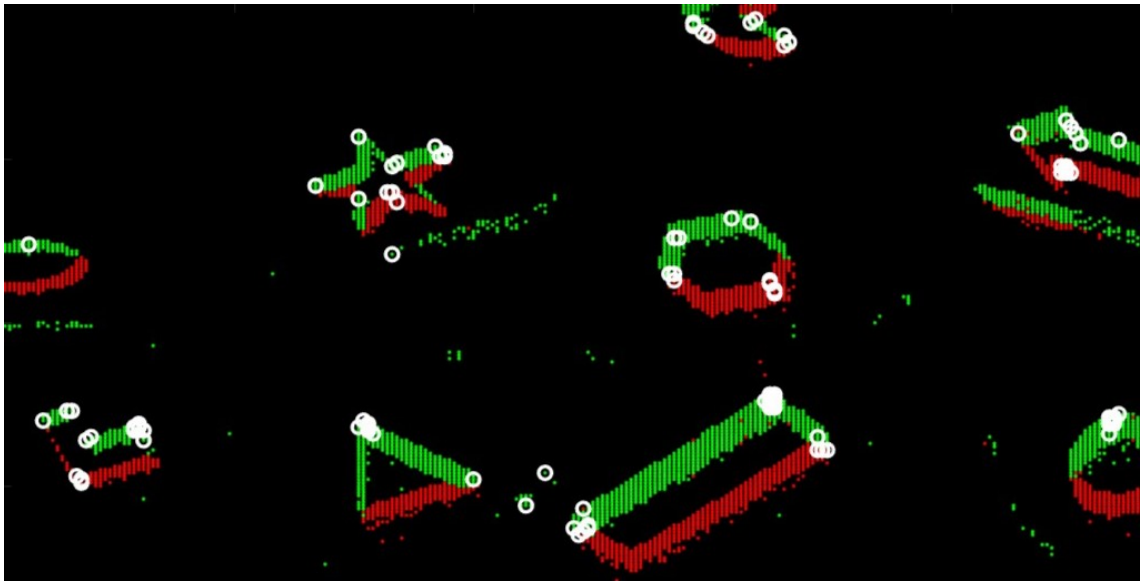
$\implies C^H(x, y)$ has a large value $\iff$ both λ_1 and λ_2 are large.

In common practice, the threshold for identifying features is set to $C^H(x, y) > 8.0$, which is an empirical number. In the past, this heuristic value has been proven to provide the best results, and this experiment confirms this. $k = 0.04$ is another empirical parameter called the Harris magic number.

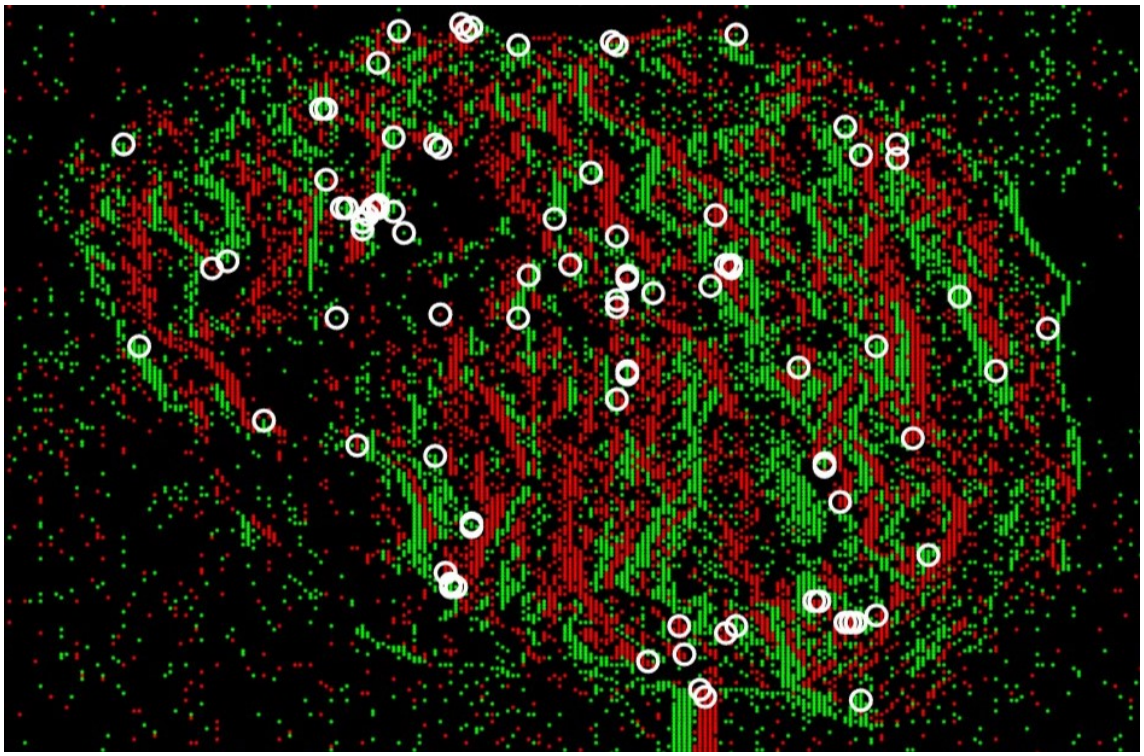
The algorithm was applied to both videos studied in the experiment. The result is reported here. The algorithm is capable of detecting features in both videos; however, due to the complex shape and motion of the rotating comet $67P/Churyumov-Geramisenko$, the code struggles to repeatedly identify the same features, which is necessary to perform feature tracking.

This limitation might also be due to the inherent noise captured in the video of the comet, which was recorded in a darkroom to mimic the space environment. In contrast to the dark space void, the darkroom was affected by tiny variations in illumination. These fluctuations resulted in many false triggers, as logarithmic oscillations 1.1 are particularly strong in dark environments. The noise obscured many of the features and covered a good chunk of the comet, resulting in the scene presented in the image.

In addition, a rotating comet is a much more complex scene than terrestrial scenarios. In such environments, detections from traditional images may still provide much more stable results, at the cost of being restricted to a fixed frame rate.



(a) Geometric Shapes



(b) $67P/Churyumov - Geramisenko$

Figure 2.19: Harris Valid Corners

2.1.4 Event-Tree Feature Tracking

After feature detection is performed, the second step in estimating camera pose is to describe the motion of the detected features across the video. This operation is performed using feature trackers, which are preferred over feature matchers due to the asynchronous and localized representation of events, as described above 2.

This section is dedicated to one of the most common strategies for tracking features [1]. Assuming good detections, the strategy assumes that each feature describes a continuous trajectory from their origin to the point at which they vanish. Therefore, a tree graph may be created to collect detections close to each other. The tree contains multiple paths to account for noise, but in the end the longest ramification is selected to describe the feature. The code is divided into two sections: the first part grows the trees, while the second selects the longest ramification of each tree.

In the image below, a possible representation of a tree is presented. The tree originates at the bottom and progresses upward as time passes. Inside, events are connected through parent-child connections, which means that each feature has only one parent while a parent may have many children. The blue point is the first point identified in a tree (root), while the red and green points represent, respectively, points with children (non-leaves) and points with no children (leaves).

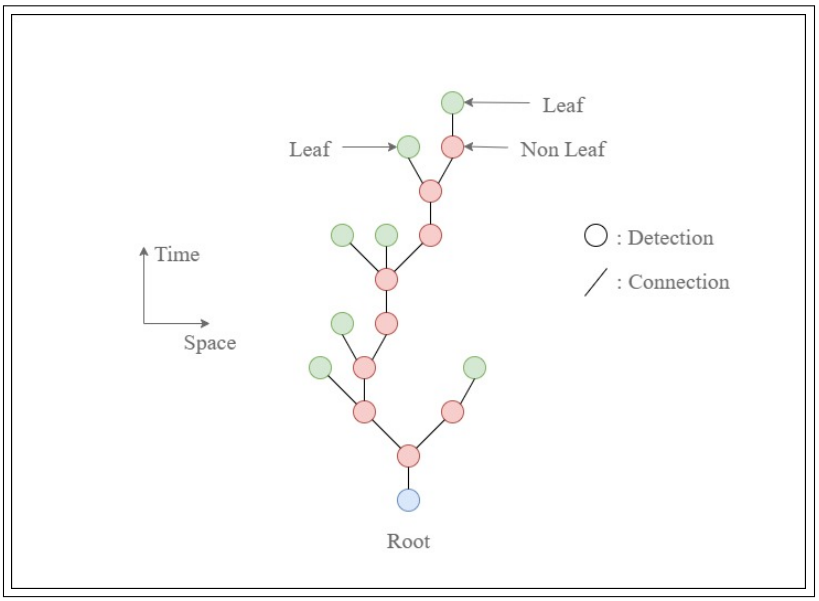


Figure 2.20: Example of a Tree Structure

Every time a new event passes the feature detector test, it must be established whether it belongs to an existing tree or not. If a connection is established, it becomes a new leaf of the tree, whereas if no connections are established it becomes the root of a new tree. Connections may be established only with features within a spatio-temporal neighbourhood around the new event. This neighbourhood consists

of a circle of radius R around the incoming event and a temporal window Δt. In the code, $R = 5pixel$ and $\Delta t = 0.1s$. The code implemented in this experiment improves particularly from the reference paper [1]. The original code did not directly check the neighbourhood of the event; instead it performed a last-in-first-out $LIFO$ recursive call to check if any of the elements satisfied the neighbouring condition. The new logic provides the same results as the recursive function in a fraction of the time.

Two scenarios may occur after the neighbourhood search:

1. If there are no neighbours, the incoming event becomes the root of a new tree.

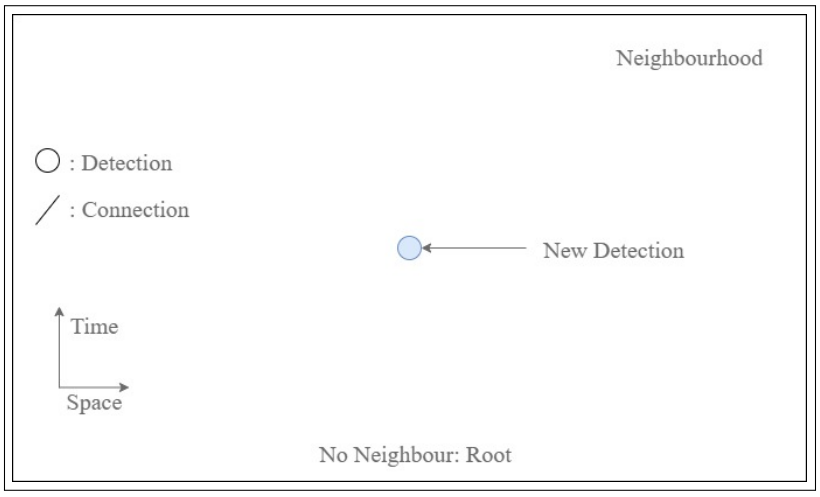


Figure 2.21: No Neighbours in Proximity of the New Event

2. If there are neighbours, they are first divided into leaves and non-leaves. As mentioned, the distinction is based on the number of children each element possesses: no children $\Longrightarrow$ leaf, children $\Longrightarrow$ non-leaf.

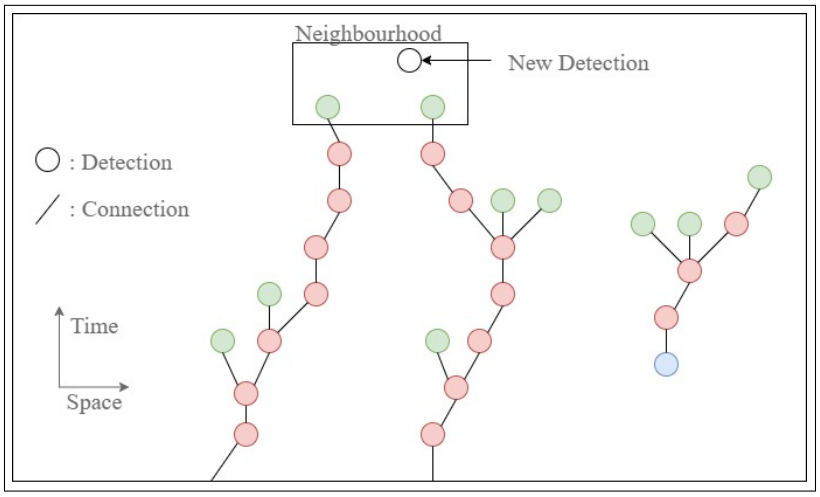


Figure 2.22: Example of a Neighbourhood Around the New Event

Since leaves are the most recently active elements in a tree, they most likely represent the previous point in the trajectory. For this reason, a connection is

always established with a leaf if any are present. Instead, non-leaves are used to define new routes in case a previous ramification was the result of a faulty detection. Therefore, two cases arise:

(a) If any leaf is present, the new connection is established with the closest leaf in space and time. To define this leaf, the algorithm compares the distance of every leaf in the neighbourhood and updates the best candidate every time a leaf is closer than the previous ones.

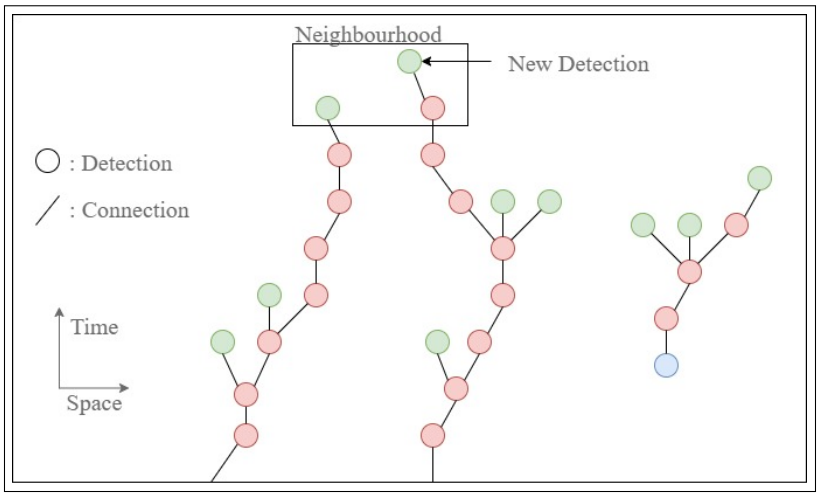


Figure 2.23: Connection Established with a Leaf

(b) If no active leaves are present, the closest non-leaf is used to define a new ramification. As mentioned above, growing the graph in multiple directions allows the code to filter out noisy events in the trajectory, since only the valid route will be selected in the end.

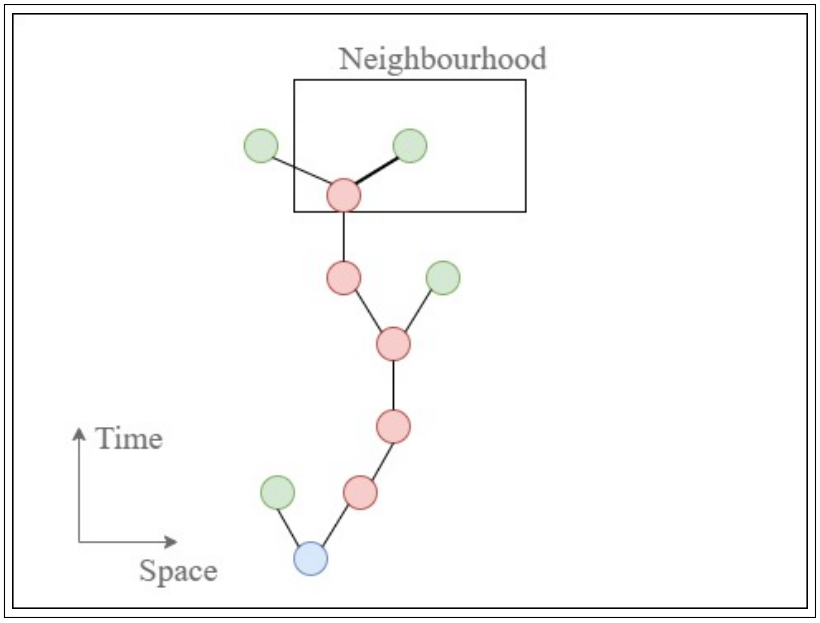


Figure 2.24: Connection Established with a Non-Leaf

The trees continue to grow until they are deactivated due to inactivity, which occurs after the threshold $\Delta\tau = 0.5s$. Additionally, old events in active trees can no longer form new connections after reaching a certain relative depth with respect to the furthest element from the root. In the code, this relative depth is 5.

After successfully building a tree, the second part of the algorithm processes the evolution of each trajectory. The selection starts from the furthest leaf of the tree and proceeds backward. Every time an element is saved, the code finds the parent of that element, and the process continues until the root is reached. For example, the image below displays a representation of the process.

Since the experiment was not conducted in real time, the process was performed at the end. However, in a real scenario it would happen during the growth of the trees.

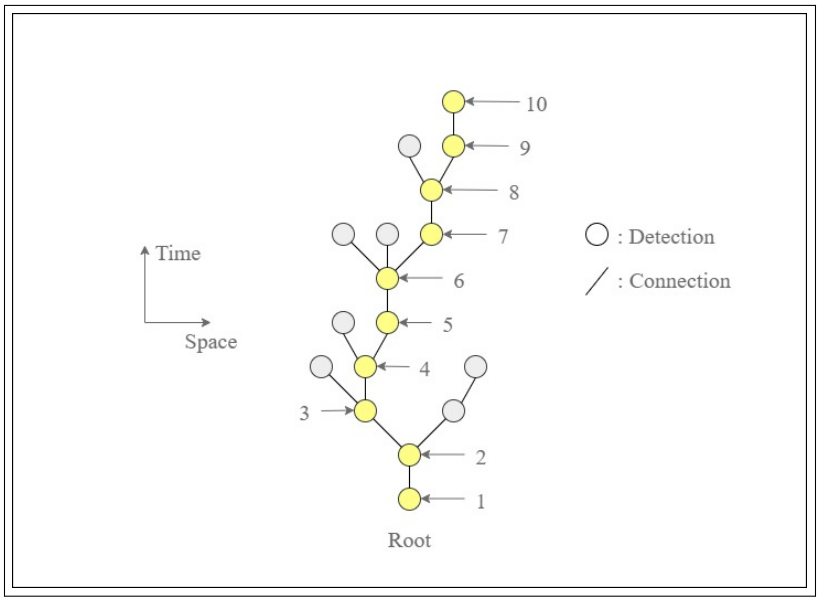
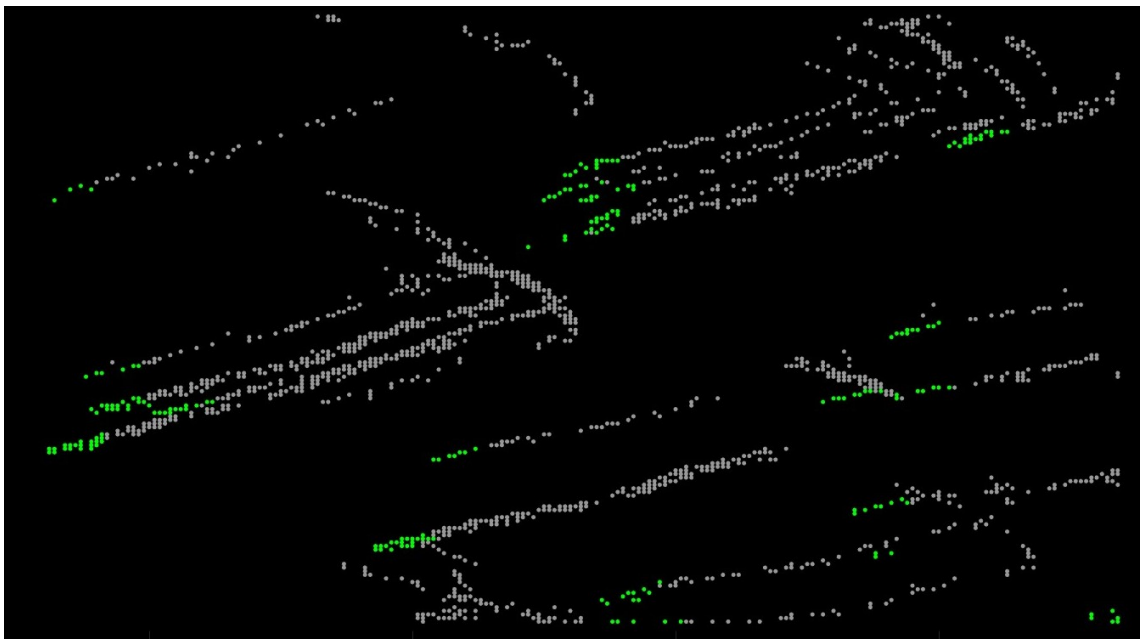


Figure 2.25: Save the Correct Trajectory

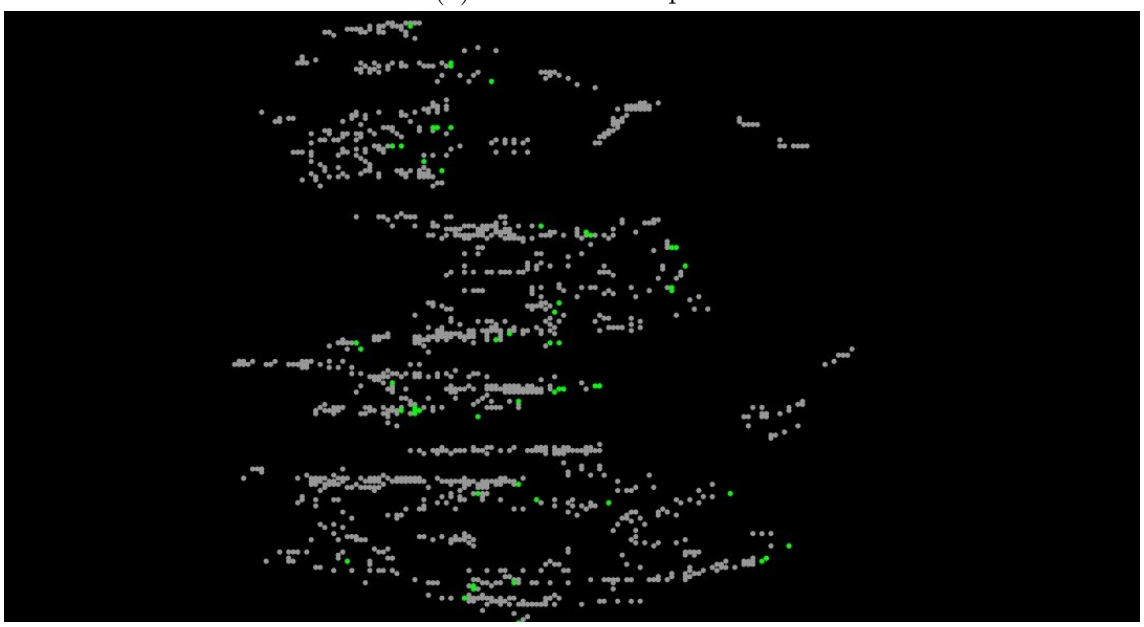
In the end, only trajectories with a minimum of 5 elements are saved to avoid useless information.

The code was applied to the two videos used in the experiment. The results are reported in the following images. Inside, green points represent the current position of a feature, while previous locations are represented in gray.

The results confirm the predictions made in the previous chapter: only videos with high contrasts, no particular noise, and representing terrestrial environments (such as the video of the geometric shapes) produce good estimates for the trajectory of the features. Meanwhile, the video of the rotating comet $67P/Churyumov - Geramisenko$ depicts a complex scenario and was performed in a dark environment, resulting in detections that were too sporadic to allow for a good feature tracking.



(a) Geometric Shapes



(b) $67P/Churyumov - Geramisenko$

Figure 2.26: Tracking of the Features

2.2 Hybrid Procedure

2.2.1 EKLT

From the results of the Event Only Procedure 2.1.4, it is concluded that event cameras provide better temporal resolution than traditional frames. However, traditional images continue to provide better stability in feature recognition of complex scenarios. For this reason, an algorithm that combines the stability of traditional images and the reaction time of events may result in an accurate and reactive tracking, even with complex environments. Fortunately, several event cameras, including the DAVIS346, are also capable of capturing traditional images. The dual acquisition is performed through the same camera lens system, giving the opportunity to implement such an algorithm.

This section introduces the working principle of one of the few algorithms that performs this fusion, EKLT [12]. As mentioned above, EKLT processes both events $e_k = (p_k, t_k, x_k, y_k), \forall k$ and image frames $F_n[x, y], \ \forall n$. The code performs both feature detection and tracking and is supported by two key principles:

1. Link between events and frames.

   As mentioned in the previous chapter 1.1.2, events e_k can be accumulated into frames $F_m^E[x, y], \forall m$ through the equation:

   $$F_m^E[x, y] \doteq \sum_{k=1}^{N_e} p_k \, \delta(\mathbf{u} - \mathbf{u}_k) \tag{2.24}$$

   Additionally, given a generic event frame $F_{\bar{m}}^E[x, y], m = \bar{m}$, it was concluded that this frame approximates the gradient in the direction of motion of its corresponding image $F_{\bar{n}}[x, y]$ expressed in the logarithmic scale. Therefore, given the optical flow $\alpha_{\bar{m}}[x, y]$, which expresses the direction of motion of each point in the frame $\bar{m}$, the relationship may be expressed as:

   $$\begin{gathered} F_{\bar{m}}^E[x, y] \approx -\nabla F_{\bar{n}}^L[x, y] \cdot \alpha_{\bar{m}}[x, y] \\ Where \ F_{\bar{n}}^L[x, y] \doteq log_e(F_{\bar{n}}[x, y]/255) \end{gathered} \tag{2.25}$$

   In the equation, 255 normalizes the intensity of the image to be inside $[0; 1]$ (instead of $[0; 255]$). As the equation suggests, corresponding quantities do not share the same index $\bar{m} \neq \bar{n}$, because many event frames can be created between two image frames.

   The equation provides a method to connect the two representations. Therefore, the code can successfully detect features using traditional frames $F_n[x, y]$ and study their evolution using event frames $F_m^E[x, y]$.

In particular, the equation above can provide an estimation of the number of events N_e required to successfully build an event frame. In fact, given that $F_m^E[x, y]$ is represented as a sum of N_e elements of value $p_k = \pm 1$:

$$F_m^E[x, y] \doteq \sum_{k=1}^{N_e} p_k \, \delta(\mathbf{u} - \mathbf{u}_k) \tag{2.26}$$

It follows that the Frobenius norm, which is the sum of the absolute values in the event frame, is exactly the number of events N_e inside the frame m:

$$\|F_m^E[x, y]\|_F \doteq \sqrt{\sum_{i,j} |F_m^E[i, j]|^2} = N_e \tag{2.27}$$

Therefore, the number of events $N_e^{\bar{m}}$ required to build an event frame $F_{\bar{m}}^E[x, y]$ may be found using traditional frames as:

$$N_e^{\bar{m}} \approx \|\nabla F_{\bar{n}}^L[x, y] \cdot \alpha_{\bar{m}}[x, y]\|_F \tag{2.28}$$

The same equations are obtained for small 25×25 patches P around the features in the scene. This adaptation is performed because the code does not actually evaluate the entire frame each time; instead, it extracts a patch around the feature and compares it to the original patch of the first scene. Therefore, assuming that the direction of motion inside each patch is constant and equal to the value obtained from the feature $\alpha_m[x, y] = \alpha_{i,m}, \forall (x, y)^T \in P$, the two relationships may be adapted to the smaller patch. Considering the 25×25 corresponding patches $P_{i,\bar{m}}^E[x, y]$ and $P_{i,\bar{n}}^I[x, y]$, respectively the event and image patch around the i^{th} feature in the frames $\bar{m}$ and $\bar{n}$, the two equations become:

$$P_{i,\bar{m}}^E[x, y] \approx -\nabla P_{i,\bar{n}}^L[x, y] \cdot \alpha_{i,\bar{m}} \tag{2.29}$$

$$N_e^{i,\bar{m}} \approx \|\nabla P_{i,\bar{n}}^L[x, y] \cdot \alpha_{i,\bar{m}}\|_F \times d \tag{2.30}$$

Where $d = 0.6$ is a scaling factor, used to reduce the estimated number of events since in practice the equation provides an overestimation.

2. How the code retrieves the motion of the features.

   Given a generic patch P_i around the feature i, EKLT assumes that its motion may be described using only translations and rotations from its origin to its current point. Therefore, given a generic point $(x, y)^T$ contained in the current patch P_i, the point is mapped to its original location in the first image $(x', y')^T$ using the following affine transformation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} cos(\gamma) & -sin(\gamma) \\ sin(\gamma) & cos(\gamma) \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2.31}$$

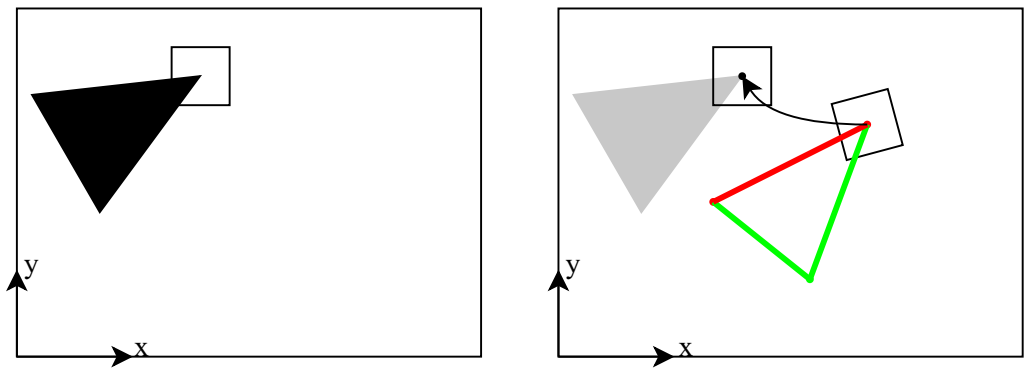
   Where $(t_x, t_y)^T$ represents the translation of the central feature from the current location to the original, while γ represents the rotation angle of the patch around its centre between the two views. The same relationship may be expressed using homogeneous coordinates:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & t_x \\ sin(\gamma) & cos(\gamma) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = W \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = W(x, y) \tag{2.32}$$

   Where W is called warp matrix.

   During the run, t_x, t_y, γ and the direction of motion α are unknown and must be computed for every event frame of every feature i. Their determination is achieved by solving a minimization problem: the solver iterates over the four unknowns until the affine transformation superimposes the current event frame $P^E_{i,\bar{m}}[x, y]$ around the feature i to the original image patch $P^I_{i,\bar{n}}[x, y]$ in which the feature i was first detected and with the image expressed through the equation 2.29.

$$\min_{t_x, t_y, \gamma, \alpha} \left\| \frac{P^E_{i,\bar{m}}[x, y]}{\|P^E_{i,\bar{m}}[x, y]\|_F} - \frac{-\nabla P^L_{i,\bar{n}}[W(x, y)] \cdot \alpha_{i,\bar{m}}}{\|\nabla P^L_{i,\bar{n}}[W(x, y)] \cdot \alpha_{i,\bar{m}}\|_F} \right\|^2 \tag{2.33}$$



(a) Original Image (b) Current Event Frame

Figure 2.27: Minimization of the Affine Transformation

The two principles are applied to the algorithm every time a new event frame is built. The purpose of the code is to track a number of features in the range $[m_f; M_f]$. In the code, this range is set to $[60, 100]$. During any point in the run, if the total number of tracked features falls below m_f, the algorithm interrogates the first two successive images $F_n[x, y]$ and $F_{n+1}[x, y]$ that appear right after the loss of the features. These two frames are used, respectively, to detect new features and to obtain an initial estimation of the parameters t_x, t_y, γ, α. After this procedure, the total number of features is again M_f.

This procedure is also performed at the beginning of the code, when the number of features tracked is 0.

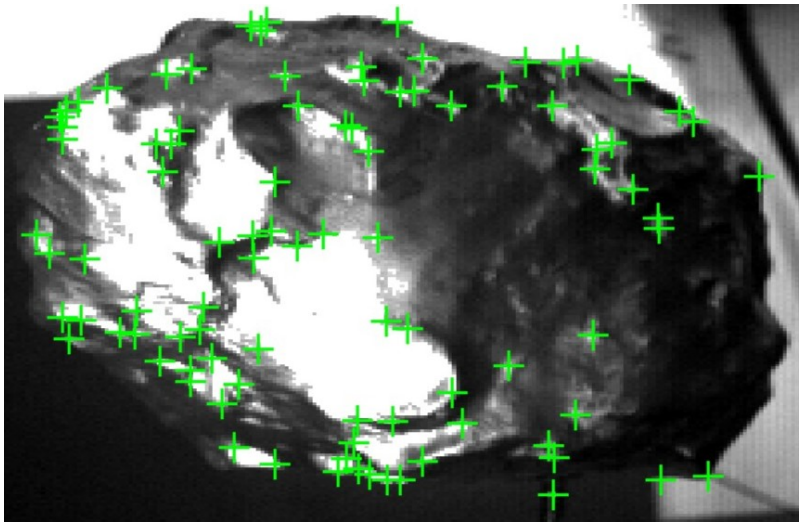


Figure 2.28: Example of Feature Detection

The two operations are performed, respectively, using the Harris feature detector [14] and the Lucas-Kanade KLT tracker [23], [4]. The formulation of the Harris detector for images is completely identical to the one derived in section 2.1.3: the only difference is that this time the method is applied to an image instead of a binarized time surface. Meanwhile, KLT solves a minimization problem similar to the one implemented in EKLT: as before, the only difference is that it compares two images instead of an event frame and a gradient.

After the two images $F_n[x, y]$ and $F_{n+1}[x, y]$, the traditional video is no longer used to track the motion of the features. Instead, each feature is tracked only with frames built from events $F_m^E[x, y]$. The code collects every event that occurs within the patch around each feature i, until the correct number N_e^i is reached to build the event patch around i. Then, the code minimizes the difference between the new event patch and the original image patch to obtain the motion parameters. Finally, the code starts again to accumulate events to build a new patch. This process continues until either every event is studied or too many features are lost, in which case the images are used to obtain new ones.

Chapter 3

Pose Estimation & Reconstruction

This chapter is dedicated to the third and final step in the estimation of the camera pose. Starting from the tracked features, the purpose of the algorithms in this section is to describe the rigid motion of the camera and to obtain a three-dimensional representation of the observed scene. The procedure is performed using projective geometry theorems, which are valid for both event cameras and traditional ones. In particular, several algorithms are required to complete the task and are implemented to work in a complementary manner. Therefore, to clarify the objectives of each code, a brief introduction is provided in the remainder of this section.

After feature detection and tracking, the evolution of each feature i is represented by a set of coordinates $(x_i^j, y_i^j)^T$, which represent the location of the i^{th} feature in the j^{th} view of the video. The set of locations $(x_i^j, y_i^j)^T$ represents the projections onto the image plane $\tilde{x}_i^j$ of the feature $\tilde{X}_i$, expressed in inhomogeneous coordinates.
The projections $\tilde{x}_i^j$ are gathered into frames, as projective theorems require multiple correspondences between views. The operation is trivial for traditional videos, as they are already recorded in discrete frames, while event cameras require the additional step. After gathering the projections $\tilde{x}_i^j$ in frames, the geometry theorems no longer depend on the nature of the video, as the description of the projections is now the same. The advantage of event cameras over traditional ones is the time interval between frames, thanks to faster detections.

The projective theorems require the camera to be calibrated, i.e., to estimate how the scene is projected onto the image plane. After calibration, the first algorithm begins to describe the motion of the camera using a couple of frames at the beginning of the video. The description involves estimating the essential matrix E and requires that the two views are sufficiently distant from each other. After detecting the first motion, the projections $\tilde{x}_i^j$ in the two frames are also used to triangulate their three-dimensional coordinates $\tilde{X}_i$.

Then, the coordinates $\tilde{X}_i$ are used to solve the Perspective-n-Point problem. The PnP problem estimates the position of a camera in a global reference system by comparing the three-dimensional representation of known features with their correspondent image projections. In particular, the operation is combined to the trian-

gulation theorem: PnP describes the location of the camera every time a new frame is studied, while triangulation estimates the three-dimensional coordinates of new features. The new coordinates are then used in the following solution of PnP, and so on.

The procedure continues until the end of the video, where all errors in the results are minimized using bundle adjustment. At the end of the process, a description of the motion of the camera is obtained, together with a set of three-dimensional points that visually represent the scene. However, as will be discussed in the estimation of the essential matrix E, projective geometry theorems detect motion only up to a scale, implying the need for an additional device, commonly an accelerometer, to define the absolute value of translation.

3.1 Essential Matrix and Epipolar Geometry

This chapter is dedicated to describing the geometry of multiple views of the same scene, the epipolar geometry [15]. In this section, epipolar geometry is applied to retrieve the essential matrix E, which is related to 5 of the 6 degrees of freedom in the camera motion. As mentioned above, a couple of frames are selected at the beginning of the video, or whenever there are not enough correspondences between three-dimensional features $\tilde{X}_i$ and projections $\tilde{x}_i^j$. The frames must contain a sufficient number of matching features, and there should be a good angular separation between the two views. At the end of the procedure, the first motion of the camera is recovered and the PnP+triangulation sequence starts.

To describe how a point is seen by two different perspectives, let $\tilde{X} = (X, Y, Z)^T$ be a three-dimensional point in a global reference system. As introduced in the first chapter 1.2.2, $\tilde{X}$ is expressed as an inhomogeneous vector, while it is more common to represent it in homogeneous coordinates as $\mathbf{X} = (X, Y, Z, 1)^T$.

In epipolar geometry, $\mathbf{X}$ is mapped onto two image planes of two cameras. The first camera is centred in $\mathbf{C}$, and the first image projection $\mathbf{x}$ is defined as the intersection of the first image plane and the segment that connects $\mathbf{C}$ and $\mathbf{X}$. The second camera is centred in $\mathbf{C}'$ and $\mathbf{x}'$ is defined accordingly. The point $\mathbf{X}$ and the two camera centres $\mathbf{C}$ and $\mathbf{C}'$ define the epipolar plane π. The intersections of the epipolar plane π with the two image planes produce two lines, respectively l and l'. Specifically, l contains the image projection $\mathbf{x}$ and the projection $\mathbf{c}'$ of the second camera centre $\mathbf{C}'$, while l' contains $\mathbf{x}'$ and $\mathbf{c}$.

In particular, the projections $\mathbf{c}$ and $\mathbf{c}'$ are contained in every epipolar plane π for any three-dimensional point $\forall\mathbf{X}$. In fact, the entire line connecting the centres $\mathbf{C}$ and $\mathbf{C}'$ is always contained in π, since $\mathbf{C}$ and $\mathbf{C}'$ define the plane. For this reason, the two projections of the centres $\mathbf{c}$ and $\mathbf{c}'$ are also called epipolar points.

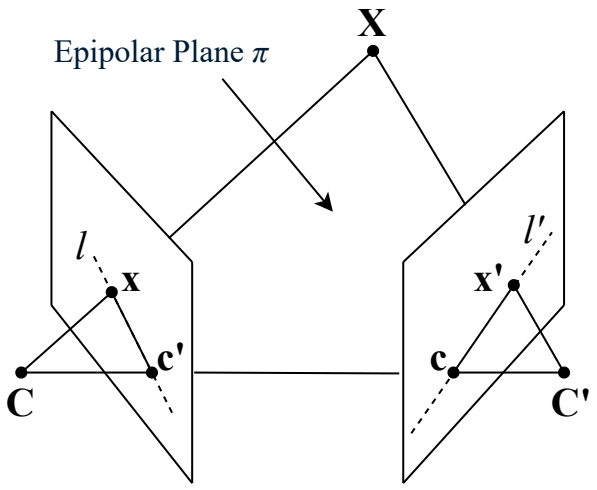


Figure 3.1: Epipolar Geometry (Adapted from [15])

In this chapter, $\mathbf{X}$ is unknown and will be derived only later from triangulation. The purpose of the following algorithm is to obtain the motion of the second camera centre $\mathbf{C}'$ with respect to the first camera $\mathbf{C}$, which is located at the origin of the global reference system. Estimation involves defining either the fundamental matrix $\mathbf{F}$ or the essential matrix $\mathbf{E}$ from a set of n corresponding projections $\mathbf{x}_i$ and $\mathbf{x}'_i$ of the same features $i \in \{1, ..., n\}$. In particular, the fundamental and essential matrices are the map that connects one projection $\mathbf{x}$ to its correspondent $\mathbf{x}'$, for any three-dimensional point seen by the two views $\forall \mathbf{X}$.

$$\mathbf{x} \longleftrightarrow \mathbf{x}' \tag{3.1}$$

To define either matrix, consider the projections $\mathbf{x}$ and $\mathbf{x}'$ of a generic three-dimensional point $\mathbf{X}$. These projections are obtained from the camera matrices $\mathbf{P}$ and $\mathbf{P}'$:

$$\mathbf{x} = \mathbf{P}\mathbf{X}, \quad \mathbf{x}' = \mathbf{P}'\mathbf{X} \tag{3.2}$$

The camera matrices $\mathbf{P}$ and $\mathbf{P}'$ are the combination of the intrinsic matrix and the extrinsic matrix. The former describes how the scene is projected onto the image plane and has been estimated during calibration. The latter describes the motion of the camera in terms of translation t and rotation R with respect to the global reference system. As mentioned, the first camera is assumed to be at the origin of the reference system $\mathbf{C} = (0, 0, 0, 1)^T = (\mathbf{0}, 1)^T$ and with the same orientation, since the main interest is to define the relative position of the second camera after a rigid transformation.

$$\mathbf{P} = K \left[\mathbb{1}_3 \mid 0\right], \quad \mathbf{P}' = K' \left[R \mid t\right] \tag{3.3}$$

The first projection $\mathbf{x}$ and the camera matrix $\mathbf{P}$ are used to estimate the point $\mathbf{X}$. As described in chapter 1.2.4, $\mathbf{X}$ is found only up to a constant $\lambda \in \mathbb{R}^+$:

$$\mathbf{X}(\lambda) = \mathbf{P}^+\mathbf{x} + \lambda\mathbf{C} \tag{3.4}$$

Where $\mathbf{P}^+$ is the pseudo-inverse of $\mathbf{P}$, $\lambda = 0 \Rightarrow \mathbf{X} = \mathbf{P}^+\mathbf{x}$ and $\lambda \to \infty \Rightarrow \mathbf{X} = \mathbf{C}$. Although not equal to $\mathbf{X}$, $\mathbf{P}^+\mathbf{x}$ is contained in the segment between $\mathbf{C}$ and $\mathbf{X}$. This condition implies that $\mathbf{P}^+\mathbf{x}$ is contained in the epipolar plane π, since every plane that contains two points must contain the line passing through those two. As a consequence, since $\mathbf{P}^+\mathbf{x}$ is contained in π, the projection of $\mathbf{P}^+\mathbf{x}$ onto the second image plane is contained in the same line l' that contains the projections $\mathbf{c}$ and $\mathbf{x}'$.

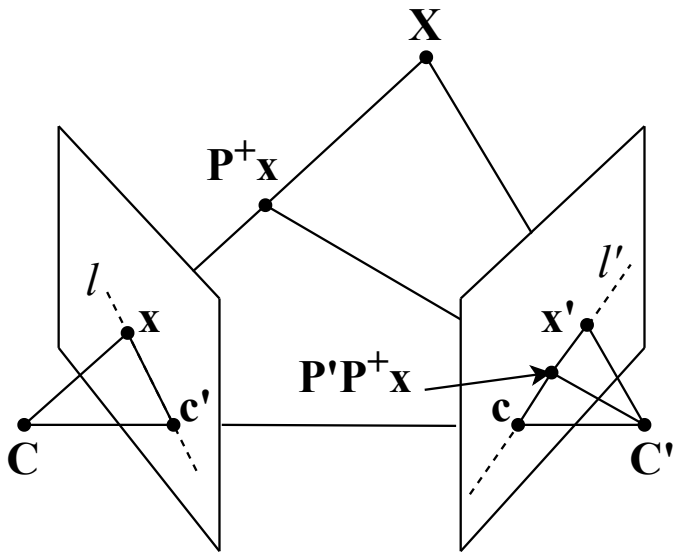


Figure 3.2: Back-Projection of $\mathbf{x}$ (Adapted from [15])

As seen in chapter 1.2.5, a line passing through two points is defined as the cross product of their homogeneous coordinates. Since l' contains the projection of $\mathbf{P}^+\mathbf{x}$ onto the second image plane and the projection $\mathbf{c}$ of the first camera centre $\mathbf{C}$, it may be defined as follows.

$$\mathbf{x}'_+ = \mathbf{P}'\mathbf{P}^+\mathbf{x}, \quad \mathbf{c} = \mathbf{P}'\mathbf{C} \tag{3.5}$$

$$l' = \mathbf{c} \times \mathbf{x}'_+ = (\mathbf{P}'\mathbf{C}) \times (\mathbf{P}'\mathbf{P}^+\mathbf{x}) \tag{3.6}$$

The cross product may be represented as the product with the skew-symmetric matrix $[\mathbf{c}]_x$:

$$l' = [\mathbf{c}]_x\mathbf{P}'\mathbf{P}^+\mathbf{x}, \quad [\mathbf{c}]_x = \begin{bmatrix} 0 & -\mathbf{c}_z & \mathbf{c}_y \\ \mathbf{c}_z & 0 & -\mathbf{c}_x \\ -\mathbf{c}_y & \mathbf{c}_x & 0 \end{bmatrix} \tag{3.7}$$

The equation defines a relationship between the first projection $\mathbf{x}$ and the second image plane. Here, the fundamental matrix $\mathbf{F}$ may be defined as:

$$\mathbf{F} \doteq [\mathbf{c}]_x\mathbf{P}'\mathbf{P}^+, \quad l' = \mathbf{F}\mathbf{x} \tag{3.8}$$

As mentioned above, also the projection $\mathbf{x}'$ is contained in l'. Therefore, as seen in chapter 1.2.5:

$$\mathbf{x}'^T l' = 0 \tag{3.9}$$

Finally, the map between the points $\mathbf{x} \longleftrightarrow \mathbf{x}'$ is found from the fundamental matrix:

$$\mathbf{x}'^T\mathbf{F}\mathbf{x} = 0 \tag{3.10}$$

Given two image projections $\mathbf{x}$ and $\mathbf{x}'$ of any three-dimensional point $\mathbf{X}$, the equation states that $\mathbf{x}$ and $\mathbf{x}'$ are linked by the same 3×3 matrix, because $\mathbf{F}$ depends only on

the camera matrices $\mathbf{P}$ and $\mathbf{P}'$ and on the projection $\mathbf{c}$. Therefore, given a sufficient number of corresponding projections $\mathbf{x}_i$ and $\mathbf{x}'_i$ of the same features $i \in \{1, ..., n\}$, this condition becomes a system of equations with the components of $\mathbf{F}$ as unknowns.

The fundamental matrix $\mathbf{F}$ describes the motion of the camera and its intrinsics, which may be expressed explicitly. From the definition of the camera matrices:

$$\mathbf{P} = K\left[\mathbb{1}_3 \mid 0\right] \Longrightarrow \mathbf{P}^+ = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1} = \begin{bmatrix} K^{-1} \\ \mathbf{0}^T \end{bmatrix} = \begin{bmatrix} \mathbb{1}_3 \\ \mathbf{0}^T \end{bmatrix} K^{-1} \tag{3.11}$$

$$Where: \ \mathbf{P}\mathbf{P}^+ = K\left[\mathbb{1}_3 \mid 0\right] \begin{bmatrix} \mathbb{1}_3 \\ \mathbf{0}^T \end{bmatrix} K^{-1} = K \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} K^{-1} = KK^{-1} = \mathbb{1}_3 \tag{3.12}$$

In the equation, $\mathbf{P}^+ \in \mathbb{R}^{4\times3}$ and is made up of the inverse of K and an additional null row. Therefore, $\mathbf{F}$ may be expressed as:

$$\begin{aligned} \mathbf{F} &\doteq [\mathbf{c}]_x \mathbf{P}'\mathbf{P}^+ = [\mathbf{P}'\mathbf{C}]_x \mathbf{P}'\mathbf{P}^+ \\ &= \left[K'[R \mid t] \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}\right]_x K'[R \mid t] \begin{bmatrix} \mathbb{1}_3 \\ \mathbf{0}^T \end{bmatrix} K^{-1} \\ &= \left[K't\right]_x K'RK^{-1} \end{aligned} \tag{3.13}$$

In the equation, $\mathbf{F}$ is related to the motion of the camera. However, it also depends on the intrinsics, and for this reason the essential matrix $\mathbf{E}$ is introduced. After calibration, the calibration matrices K and K' are known and are used to express the two projections $\mathbf{x}$ and $\mathbf{x}'$ as normalized coordinates:

$$\hat{\mathbf{x}} = K^{-1}\mathbf{x}, \quad \hat{\mathbf{x}}' = K'^{-1}\mathbf{x} \tag{3.14}$$

As a consequence of the change in coordinates, the camera matrices must also be normalized $\hat{\mathbf{P}}$ and $\hat{\mathbf{P}}'$. The new equations that map the three-dimensional point $\mathbf{X}$ and its projections may be found as follows:

$$\begin{aligned} \hat{\mathbf{x}} = \hat{\mathbf{P}}\mathbf{X}, \qquad \hat{\mathbf{x}}' = \hat{\mathbf{P}}'\mathbf{X} \\ \hat{\mathbf{P}} = \left[\mathbb{1}_3 \mid 0\right], \quad \hat{\mathbf{P}}' = \left[R \mid t\right] \end{aligned} \tag{3.15}$$

Following the same procedure, the map between the points $\hat{\mathbf{x}} \longleftrightarrow \hat{\mathbf{x}}'$ is found from the essential matrix $\mathbf{E}$:

$$\hat{\mathbf{x}}'^T \mathbf{E} \hat{\mathbf{x}} = 0 \tag{3.16}$$

Where $\mathbf{E}$ is defined as follows:

$$\begin{aligned} \mathbf{E} &\doteq [\mathbf{c}]_x \hat{\mathbf{P}}'\hat{\mathbf{P}}^+ = [\hat{\mathbf{P}}'\mathbf{C}]_x \hat{\mathbf{P}}'\hat{\mathbf{P}}^+ \\ &= \left[[R \mid t] \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}\right]_x [R \mid t] \begin{bmatrix} \mathbb{1}_3 \\ \mathbf{0}^T \end{bmatrix} \\ &= \left[t\right]_x R \end{aligned} \tag{3.17}$$

As the equation states, the effect of the camera intrinsics was removed. Again, given a sufficient number of corresponding projections $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$ of the same features $i \in \{1, ..., n\}$, this condition becomes a system of equations with the components of $\mathbf{E}$ as unknowns. The components of $\mathbf{E}$ are usually found using Nistér 5-points algorithm [30] or the normalized 8-point algorithm [15]. The former exploits an additional constraint on $\mathbf{E}$: every essential matrix has two equal singular values and a third singular value equal to zero. Therefore:

$$\mathbf{E}\mathbf{E}^T\mathbf{E} - \frac{1}{2}tr(\mathbf{E}\mathbf{E}^T)\mathbf{E} = 0 \tag{3.18}$$

If there are more points than the minimum required, the solution is found using a least-square approach or using $MSAC$ [35]. $MSAC$ computes the essential matrix multiple times using the minimum number of projections required, and then computes a precise estimation from all the trials. Using the precise estimation of $\mathbf{E}$, the algorithm then verifies which projections do not satisfy the equation $\hat{\mathbf{x}}'^T\mathbf{E}\hat{\mathbf{x}} = 0$, and marks them as outliers.

The components of t and R may be computed from SVD decomposition of $\mathbf{E}$. In particular, the translation t may be found only up to a scale. In fact, even if t is scaled by a scalar $\lambda \in \mathbb{R}$, the resulting essential matrix still satisfies the equation $\hat{\mathbf{x}}'^T\mathbf{E}\hat{\mathbf{x}} = 0$.

$$\begin{aligned} &\left[\lambda t\right]_x R = \lambda t \times R = \lambda \left[t\right]_x R \\ \implies &\hat{\mathbf{x}}'^T \left[\lambda t\right]_x R\hat{\mathbf{x}} = \lambda(\hat{\mathbf{x}}'^T \left[t\right]_x R\hat{\mathbf{x}}) = 0 \end{aligned} \tag{3.19}$$

This is a symptom of the loss of depth in the camera projections. As a result, $\mathbf{E}$ describes only 5 degrees of freedom of camera motion: three rotations and two translations.

3.2 Triangulation

This section explains how to determine the location of three-dimensional features $\mathbf{X}_i$ in a global reference system [15]. The determination requires at least two projections $\mathbf{x}_i^j$ for each feature $i = \{1, ..., n\}$ and is performed every time a new pose of the camera is computed. Therefore, it occurs after the first determination of the essential matrix $\mathbf{E}$ and after each step of PnP.

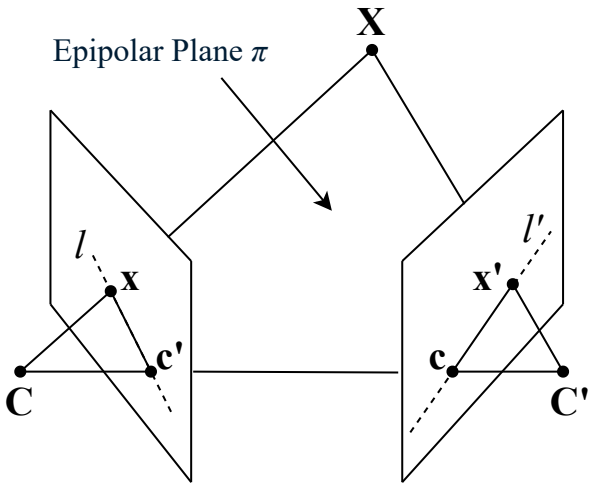


Figure 3.3: Triangulation (Adapted from [15])

Given a single feature $\mathbf{X}$ and its projections $\mathbf{x}$ and $\mathbf{x}'$, the camera matrices $\mathbf{P}$ and $\mathbf{P}'$ are computed from the newly found pose and from the calibration. In general:

$$\mathbf{P} = K\left[R \mid t\right], \quad \mathbf{P}' = K'\left[R' \mid t'\right] \tag{3.20}$$

Where K and K' are the calibration matrices and R,t and R',t' represent two generic poses of the camera in the global reference system. The camera matrices map the feature $\mathbf{X}$ to its projections as follows:

$$\mathbf{x} = \mathbf{P}\mathbf{X}, \quad \mathbf{x}' = \mathbf{P}'\mathbf{X} \tag{3.21}$$

As mentioned 1.2.4, a single equation $\mathbf{x} = \mathbf{P}\mathbf{X}$ can compute $\mathbf{X}$ only up to a constant $\lambda \in \mathbb{R}$. For this reason, at least two projections must be combined into a linear system $A\mathbf{X} = \mathbf{0}$ to find $\mathbf{X}$. The linear system is obtained from the cross product of the projective maps with the respective projections.

$$\begin{aligned} \mathbf{x} \times \mathbf{P}\mathbf{X} &= \mathbf{x} \times \mathbf{x} = \mathbf{0} \\ \mathbf{x}' \times \mathbf{P}'\mathbf{X} &= \mathbf{x}' \times \mathbf{x}' = \mathbf{0} \end{aligned} \tag{3.22}$$

Where $a \times a = 0, \ \forall a$.

The cross products provide three equations each.

$$\begin{aligned}\mathbf{x}\times\mathbf{PX} &= \begin{pmatrix} y(\sum_{i=1}^{4} p_{3i}X_i) - (\sum_{i=1}^{4} p_{2i}X_i) \\ (\sum_{i=1}^{4} p_{1i}X_i) - x(\sum_{i=1}^{4} p_{3i}X_i) \\ x(\sum_{i=1}^{4} p_{2i}X_i) - y(\sum_{i=1}^{4} p_{1i}X_i) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \\ \mathbf{x}'\times\mathbf{P}'\mathbf{X} &= \begin{pmatrix} y'(\sum_{i=1}^{4} p'_{3i}X_i) - (\sum_{i=1}^{4} p'_{2i}X_i) \\ (\sum_{i=1}^{4} p'_{1i}X_i) - x'(\sum_{i=1}^{4} p'_{3i}X_i) \\ x'(\sum_{i=1}^{4} p'_{2i}X_i) - y'(\sum_{i=1}^{4} p'_{1i}X_i) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\end{aligned} \tag{3.23}$$

Where $p_{ij} \in \mathbb{R}$ are the components of the matrix $\mathbf{P}$ and $\mathbf{X} = (X_1, X_2, X_3, X_4)^T$. The third and sixth equations are linear combinations of the others (for example, $-y$ II $- x$ I $=$ III, and the same for the sixth). Therefore, only 4 equations are linearly independent, and the resulting system becomes:

$$A\mathbf{X} = \begin{bmatrix} y\mathbf{p}_3 - \mathbf{p}_2 \\ \mathbf{p}_1 - x\mathbf{p}_3 \\ y'\mathbf{p}'_3 - \mathbf{p}'_2 \\ \mathbf{p}'_1 - x'\mathbf{p}'_3 \end{bmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.24}$$

With $\mathbf{p}_i = [p_{i1}, p_{i2}, p_{i3}, p_{i4}], \mathbf{p}_i \in \mathbb{R}^{1\times 4}$ being the i^{th} row of the matrix $\mathbf{P}$. $A\mathbf{X} = \mathbf{0}$ is a homogeneous system of 4 equations and 4 unknowns that represent a homogeneous vector. The system may be solved in two ways: using a direct linear transformation DLT or using the inhomogeneous method. The latter sets $\mathbf{X} = (X, Y, Z, 1)^T$, where $(X, Y, Z)^T = \tilde{X}$ is the inhomogeneous representation of the three-dimensional point. Therefore, the inhomogeneous method reduces the unknowns to 3 and solves the system using the least-squares approach. Meanwhile, the former solves the system in homogeneous coordinates by searching for the smallest singular value of the matrix A.

The method takes the name of linear triangulation from measured correspondences. It is a direct method that naively triangulates the three-dimensional point $\mathbf{X}$ by finding the intersection of the back-projections of $\mathbf{x}$ and $\mathbf{x}'$. However, due to the accuracy of the measurements, it is most likely that the two back-projections do not intersect, thus preventing to find the solution. Therefore, a correction must be made to the measured values affected by noise. Assuming that the measured projections are mostly affected by Gaussian noise, the most common algorithm is the two-view optimal solution.

Given a measured correspondence $\mathbf{x} \longleftrightarrow \mathbf{x}'$ affected by noise, this couple does not satisfy the constraint of the fundamental matrix $\mathbf{x}'^T\mathbf{F}\mathbf{x} \neq 0$. Therefore, the optimal solution seeks to find the closest points $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ that satisfy $\bar{\mathbf{x}}'^T\mathbf{F}\bar{\mathbf{x}} = 0$. Therefore, the points $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ minimize the function:

$$\min_{\bar{\mathbf{x}},\bar{\mathbf{x}}'} C(\mathbf{x},\mathbf{x}') = d(\mathbf{x}, \bar{\mathbf{x}})^2 + d(\mathbf{x}', \bar{\mathbf{x}}')^2, \ \textit{Subjected to}\ \bar{\mathbf{x}}'^T\mathbf{F}\bar{\mathbf{x}} = 0 \tag{3.25}$$

Where $d(\cdot, \cdot)$ is the Euclidean distance between two points.

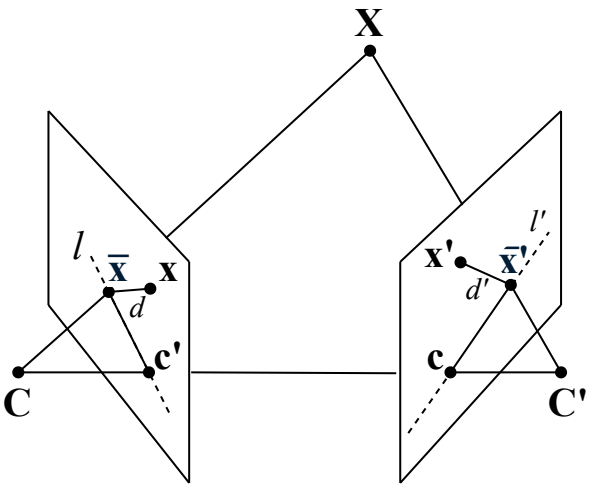


Figure 3.4: Optimal Solution (Adapted from [15])

The two points $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ must lie on the two lines l and l' 1.2.5 that are found as the intersection of the image planes with the epipolar plane π of the point $\mathbf{X}$ 3.1. In fact, since $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ satisfy the constraint $\bar{\mathbf{x}}'^T\mathbf{F}\bar{\mathbf{x}} = 0$:

$$\begin{aligned} \bar{\mathbf{x}}'^T\mathbf{F}\bar{\mathbf{x}} = \bar{\mathbf{x}}'^T l' = 0, \quad Where\ l' = \mathbf{F}\bar{\mathbf{x}} \quad \Longrightarrow \quad \bar{\mathbf{x}}' \in l' \\ \bar{\mathbf{x}}'^T\mathbf{F}\bar{\mathbf{x}} = l^T\bar{\mathbf{x}} = 0, \quad Where\ l = \mathbf{F}^T\bar{\mathbf{x}}' \quad \Longrightarrow \quad \bar{\mathbf{x}} \in l \end{aligned} \tag{3.26}$$

However, every couple of points on l and l' satisfies the constraint, and for this reason the closest points are searched. From geometry, the shortest distance from a point to a line corresponds to the perpendicular distance.

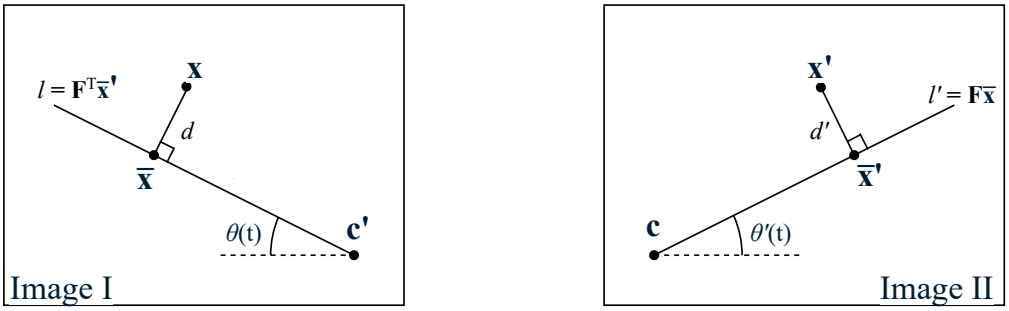


Figure 3.5: Image Planes (Adapted from [15])

Consequently, the minimization function may be reformulated to find the unknown lines l and l' that produce the minimum perpendicular distance.

$$\min_{l,l'} C = d(\mathbf{x}, l)^2 + d(\mathbf{x}', l')^2 \tag{3.27}$$

Where $d(\mathbf{x}, l)$ represents the perpendicular distance from the point $\mathbf{x}$ to the line l.

The lines l and l' are unknown; however, as mentioned in chapter 3.1, the projections $\mathbf{c}'$ and $\mathbf{c}$ of the camera centres $\mathbf{C}'$ and $\mathbf{C}$ are always contained, respectively, in l and l'. Therefore, a parametrization $l = l(t)$ and $l' = l'(t)$ may be used to represent the pencil of lines passing through $\mathbf{c}'$ and $\mathbf{c}$. As a consequence, the optimization problem becomes the minimization of a function in a single variable.

$$\min_{\bar{\mathbf{x}},\bar{\mathbf{x}}'} C(\mathbf{x},\mathbf{x'}) = \min_{t} C = d(\mathbf{x}, l(t))^2 + d(\mathbf{x}', l(t)')^2 \tag{3.28}$$

To parametrize l and l' as functions of t, an assumption is made: neither projection $\mathbf{x}$ nor $\mathbf{x}'$ lies in the segment between the two camera centres $\mathbf{C}$ and $\mathbf{C}'$. The condition is equivalent to $\mathbf{x} \neq \mathbf{c}' \wedge \mathbf{x}' \neq \mathbf{c}$. In the degenerate case in which $\mathbf{x} = \mathbf{c}' \wedge \mathbf{x}' = \mathbf{c}$, the three-dimensional feature $\mathbf{X}$ would be between the two camera centres $\mathbf{C}$ and $\mathbf{C}'$. In this case, it would be impossible to estimate its distance from the two cameras because the equations of the two projections would be identical and therefore the system would be rank deficient.

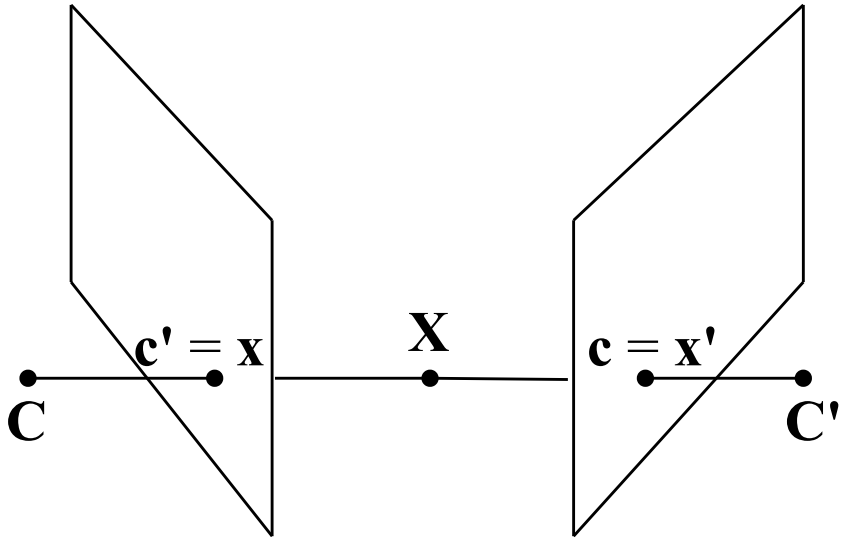


Figure 3.6: Degenerate Case in Triangulation

Meanwhile, if $\mathbf{x} \neq \mathbf{c}' \wedge \mathbf{x}' \neq \mathbf{c}$, a rigid transformation may be applied to the image planes of the two cameras. In fact, rigid transformations maintain the distances in the view, while providing a simpler description of the equations. To understand how distances are maintained in rigid transformations of the image plane, think about rotating and translating a camera while keeping the same distance from the scene. As a consequence, the objects in the scene maintain their dimensions, but are placed in different locations in the image.

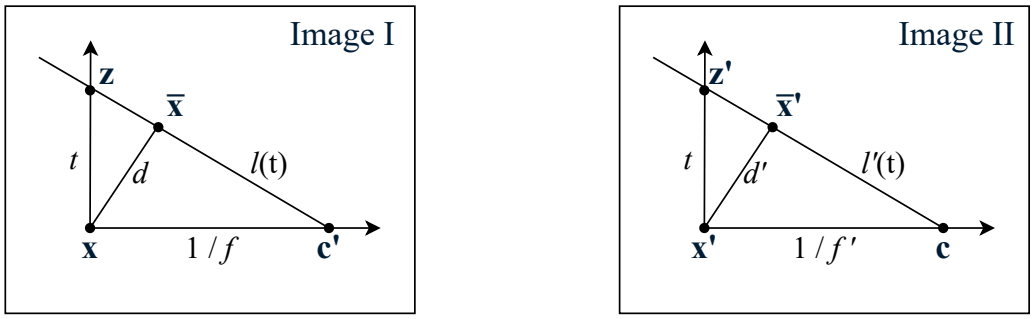


Figure 3.7: Rigid Translation of the Views

The rigid transformation places both measured projections $\mathbf{x}$ and $\mathbf{x}'$ at the origin of their respective image planes. In homogeneous coordinates:

$$\tilde{x} = \tilde{x}' = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \implies \mathbf{x} = \mathbf{x}' = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.29}$$

Meanwhile, the projections of the camera centres $\mathbf{c}$ and $\mathbf{c}'$ are placed on the positive side of the x axis. Taking into account $1/f$ and $1/f'$ the original distances of $\mathbf{c}$ and $\mathbf{c}'$ from, respectively, $\mathbf{x}$ and $\mathbf{x}'$, the resulting inhomogeneous and homogeneous coordinates are:

$$\begin{aligned} \tilde{c} = \begin{pmatrix} \frac{1}{f} \\ 0 \end{pmatrix} &\implies \mathbf{c} = \begin{pmatrix} 1 \\ 0 \\ f \end{pmatrix} \\ \tilde{c}' = \begin{pmatrix} \frac{1}{f'} \\ 0 \end{pmatrix} &\implies \mathbf{c}' = \begin{pmatrix} 1 \\ 0 \\ f' \end{pmatrix} \end{aligned} \tag{3.30}$$

As a result, the line l may be parametrized as the pencil of lines containing $\mathbf{c}'$ and an additional point $\mathbf{z}$ on the y axis, which depends on the variable t.

$$\tilde{z}(t) = \begin{pmatrix} 0 \\ t \end{pmatrix} \implies \mathbf{z}(t) = \begin{pmatrix} 0 \\ t \\ 1 \end{pmatrix} \tag{3.31}$$

As described in chapter 1.2.5, l is the cross product of $\mathbf{z}$ and $\mathbf{c}'$:

$$l(t) = \mathbf{z}(t) \times \mathbf{c}' = \begin{pmatrix} 0 \\ t \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ f \end{pmatrix} = \begin{pmatrix} tf \\ 1 \\ -t \end{pmatrix} \tag{3.32}$$

Therefore, the perpendicular distance from the point $\mathbf{x}$ to the pencil of lines $l(t)$ may be computed. For example, given a point $\tilde{x}_o = (x_o, y_o)^T$ and a line $s : ax + by + c = 0 \implies s = (a, b, c)^T$, the minimum distance is:

$$d(\tilde{x}_o, s)^2 = \frac{(ax_o + by_o + c)^2}{a^2 + b^2} \tag{3.33}$$

Therefore:

$$d(\mathbf{x}, l(t))^2 = \frac{t^2}{1 + (tf)^2} \tag{3.34}$$

Similarly to l, $l'(t)$ contains the projection $\mathbf{c}$ of the first camera centre $\mathbf{C}$ and an additional point $\mathbf{z}'(t)$ on the y axis.

$$l'(t) = \mathbf{c} \times \mathbf{z}'(t), \quad \mathbf{z}'(t) = \begin{pmatrix} 0 \\ t \\ 1 \end{pmatrix} \tag{3.35}$$

Following the same passages described in chapter 3.1, l' may be expressed using the fundamental matrix $\mathbf{F}$.

$$l'(t) = \mathbf{F}\mathbf{z}', \quad \mathbf{F} \doteq [\mathbf{c}]_x \mathbf{P}'\mathbf{P}^+ \tag{3.36}$$

Therefore, to compute the parametrization of $l'(t)$, the explicit form of the fundamental matrix $\mathbf{F}$ must be derived. As mentioned in chapter 3.1, the projections $\mathbf{c}$ and $\mathbf{c}'$ are contained in every epipolar plane π for any three-dimensional point $\forall \mathbf{X}$. Consequently, they are contained in every lines l and l'. Thus, from the property derived in section 1.2.5:

$$\mathbf{c}'^T l = 0 \wedge l'^T \mathbf{c} = 0 \quad \forall \mathbf{X} \tag{3.37}$$

Using the definition of fundamental matrix $\mathbf{F}$:

$$\mathbf{c}'^T \mathbf{F}\mathbf{x} = 0, \quad \forall \mathbf{x} \quad \wedge \quad \mathbf{x}'^T \mathbf{F}\mathbf{c} = 0, \quad \forall \mathbf{x}' \tag{3.38}$$

Since the identity is verified for any projections $\forall \mathbf{x}, \mathbf{x}'$, $\mathbf{c}'$ is the left null vector of $\mathbf{F}$ and $\mathbf{c}$ is the right null vector of $\mathbf{F}$.

$$\mathbf{c}'^T \mathbf{F} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \wedge \quad \mathbf{F}\mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.39}$$

In matrix form:

$$\begin{aligned} \begin{pmatrix} 1 & 0 & f' \end{pmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} = 0 \quad &\Longrightarrow \quad \begin{cases} F_{11} + F_{31} f' = 0 \\ F_{12} + F_{32} f' = 0 \\ F_{13} + F_{33} f' = 0 \end{cases} \\ \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ f \end{pmatrix} = 0 \quad &\Longrightarrow \quad \begin{cases} F_{11} + F_{13} f = 0 \\ F_{21} + F_{23} f = 0 \\ F_{31} + F_{33} f = 0 \end{cases} \end{aligned} \tag{3.40}$$

From which $\mathbf{F}$ has the form:

$$\mathbf{F} = \begin{bmatrix} ff'd & -f'c & -f'd \\ -fb & a & b \\ -fd & c & d \end{bmatrix} \tag{3.41}$$

Therefore, the homogeneous representation of l' results:

$$l'(t) = \mathbf{F}\mathbf{z}'(t) = \mathbf{F}\begin{pmatrix}0\\ t\\ 1\end{pmatrix} = \begin{pmatrix}-f'(ct+d)\\ at+b\\ ct+d\end{pmatrix} \tag{3.42}$$

The perpendicular distance from the point $\mathbf{x}'$ to the pencil of lines $l'(t)$ results:

$$d(\mathbf{x}', l'(t))^2 = \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2} \tag{3.43}$$

Therefore, the total sum of the differences results:

$$\begin{aligned}&\min_t C(t) = d(\mathbf{x}, l(t))^2 + d(\mathbf{x}', l(t)')^2\\ &= \min_t \frac{t^2}{1+(tf)^2} + \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2}\end{aligned} \tag{3.44}$$

In conclusion, the optimization problem is reduced to the minimization of a function of a single variable. The minimization problem is solved by computing the derivative:

$$C'(t) = \frac{2t}{(1+f^2t^2)^2} - \frac{2(ad-bc)(at+b)(ct+d)}{((at+b)^2 + f'^2(ct+d)^2)^2} \tag{3.45}$$

The minimum and maximum of $C(t)$ occur for $C'(t) = 0$. The terms of $C'(t)$ are collected over a common denominator and the numerator is equal to 0.

$$\begin{aligned}num : &t((at+b)^2 + f'^2(ct+d)^2)^2\\ &-(ad-bc)(1+f^2t^2)^2(at+b)(ct+d)\\ &= 0\end{aligned} \tag{3.46}$$

The minimum and maximum of $C(t)$ are the roots of a 6^{th} degree polynomial. The solutions are up to 6 real roots, for a total of 3 minimums and 3 maximums. The real minimum $\bar{t}$ is found as the minimum value assumed by $C(t)$ in the roots, in addition to the values of $C(t)$ for $t \to \infty$. The roots are found using numerical analysis, but the theorem itself is not iterative.

The corrected projections $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ are found from $\bar{t}$ as the closest points to $\mathbf{x}$ and $\mathbf{x}'$ contained in the lines $l(\bar{t})$ and $l'(\bar{t})$. Subsequently, the three-dimensional point $\mathbf{X}$ is found by linear triangulation of $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$. In the end, the locations of the three-dimensional points $\mathbf{X}_i$ are known in the global reference system. The points $\mathbf{X}_i$ are used to solve PnP and obtain a representation of the scene.

3.3 Perspective-*n*-Point Problem

After a set of three-dimensional points $\mathbf{X}_i, i \in \{1, ..., n\}$ has been determined, PnP evaluates the pose of the camera in a new frame j. The algorithm compares the three-dimensional points with the corresponding projections $\mathbf{x}_i^j$ and establishes the relative position of the camera in j with respect to $\mathbf{X}_i$.

In particular, PnP is solved using $n = 3$ points [11] [19], which is the smallest subset of PnP that provides a finite number of solutions. When more than three correspondences are known $n > 3$, the algorithm randomly divides the points into sets of 3 elements and solves the P*3*P with each set, to determine a stable solution and identify noisy outliers.

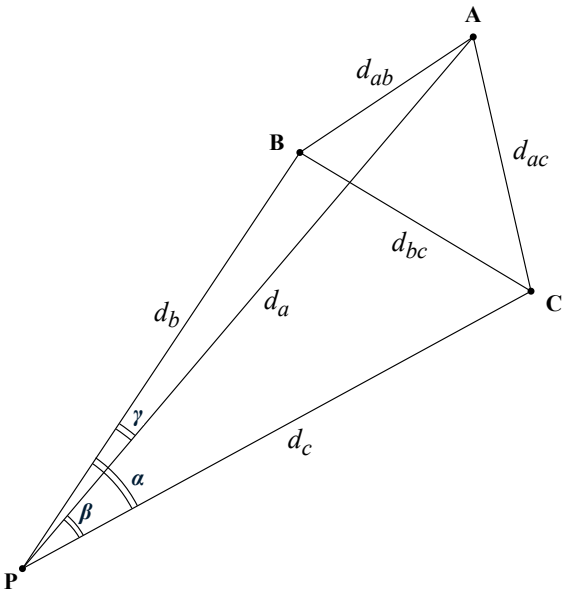


Figure 3.8: P*3*P Problem

Given a calibrated camera with a calibration matrix K, the position of its centre $\mathbf{P}$ is unknown in the global reference system. To determine $\mathbf{P}$, consider 3 known three-dimensional points $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, expressed in homogeneous coordinates. The distance between $\mathbf{P}$ and the 3 points is unknown and is defined as:

$$d(\mathbf{P}, \mathbf{A}) = d_a, \quad d(\mathbf{P}, \mathbf{B}) = d_b, \quad d(\mathbf{P}, \mathbf{C}) = d_c \tag{3.47}$$

Meanwhile, the relative distances between the three points are known, since $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are known:

$$d(\mathbf{A}, \mathbf{B}) = d_{ab}, \quad d(\mathbf{B}, \mathbf{C}) = d_{bc}, \quad d(\mathbf{AC}) = d_{ac} \tag{3.48}$$

Let **a**, **b**, and **c** be the image projections of the points **A**, **B**, and **C** in the current view. **a**, **b**, and **c** are used to determine the angles $\alpha = \angle\mathbf{BPC}$, $\beta = \angle\mathbf{APC}$ and $\gamma = \angle\mathbf{APB}$. In order to compute the angles, the unit vectors $\hat{\mathbf{u}}_a$, $\hat{\mathbf{u}}_b$ and $\hat{\mathbf{u}}_c$ are defined. A unit vector $\hat{\mathbf{u}}_x$ originates in the camera centre **P** and points towards the image projection **x**, $\forall\mathbf{x}$. In homogeneous coordinates:

$$\begin{aligned}\hat{\mathbf{u}}_a &= \frac{(K[\mathbb{1}_3|0])^+\mathbf{a}}{\|(K[\mathbb{1}_3|0])^+\mathbf{a}\|_2} \\ \hat{\mathbf{u}}_b &= \frac{(K[\mathbb{1}_3|0])^+\mathbf{b}}{\|(K[\mathbb{1}_3|0])^+\mathbf{b}\|_2} \\ \hat{\mathbf{u}}_c &= \frac{(K[\mathbb{1}_3|0])^+\mathbf{c}}{\|(K[\mathbb{1}_3|0])^+\mathbf{c}\|_2}\end{aligned} \tag{3.49}$$

Where:

1. $K[\mathbb{1}_3|0]$ is the projection matrix. To define $\hat{\mathbf{u}}_x$ in the camera centre **P**, $K[\mathbb{1}_3|0]$ considers a null translation and rotation.
2. $(K[\mathbb{1}_3|0])^+\mathbf{x}$ is the back-projection of **x**, evaluated using the pseudo-inverse matrix $(\cdot)^+$. As mentioned, depth is lost in the projection, but a unit vector needs only the direction.
3. $\|\cdot\|_2$ is the Euclidean norm and normalizes the unit vector.
4. $\hat{\mathbf{u}}_x \in \mathbb{R}^4$ is a homogeneous vector expressed in 4 coordinates.

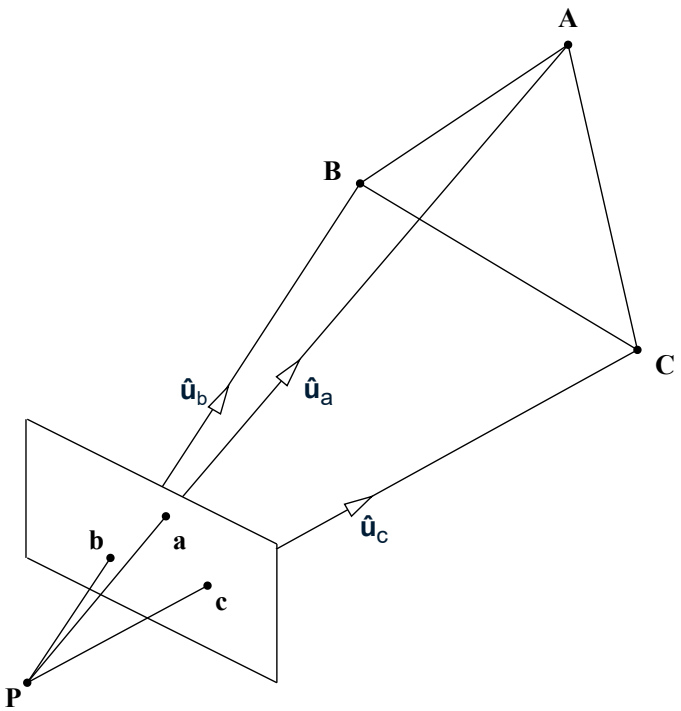


Figure 3.9: Image Plane

From the homogeneous unit vectors $\hat{\mathbf{u}}_a$, $\hat{\mathbf{u}}_b$ and $\hat{\mathbf{u}}_c$, the algorithm computes the inhomogeneous unit vectors $\hat{u}_a$, $\hat{u}_b$ and $\hat{u}_c$. Then, the angles α, β and γ are determined.

$$\begin{aligned} \cos\alpha &= \hat{u}_b \cdot \hat{u}_c \\ \cos\beta &= \hat{u}_a \cdot \hat{u}_c \\ \cos\gamma &= \hat{u}_a \cdot \hat{u}_b \end{aligned} \tag{3.50}$$

Subsequently, the unknown distances d_a, d_b and d_c may be found from the law of cosines applied to the triangles **PAB**, **PAC** and **PBC**.

$$\begin{cases} d_a^2 + d_b^2 - 2d_a d_b \cos\gamma = d_{ab}^2 \\ d_a^2 + d_c^2 - 2d_a d_c \cos\beta = d_{ac}^2 \\ d_b^2 + d_c^2 - 2d_b d_c \cos\alpha = d_{bc}^2 \end{cases} \tag{3.51}$$

The equations have an infinite number of solutions. Therefore, since the correct solution must have a physical meaning, the following "reality conditions" are applied:

$$\begin{cases} d_a > 0,\ d_b > 0,\ d_c > 0,\ d_{ab} > 0,\ d_{ac} > 0,\ d_{bc} > 0 \\ d_{ab} + d_{ac} > d_{bc},\ d_{ab} + d_{bc} > d_{ac},\ d_{bc} + d_{ac} > d_{ab} \\ 0 < \alpha, \beta, \gamma < \pi,\ 0 < \alpha + \beta + \gamma < 2\pi \\ \alpha + \beta > \gamma,\ \beta + \gamma > \alpha,\ \gamma + \alpha > \beta \\ I_o = 4\cos\alpha^2 + 4\cos\beta^2 + 4\cos\gamma^2 - 8\cos\alpha\cos\beta\cos\gamma - 1 \neq 0 \end{cases} \tag{3.52}$$

In particular, the last condition $I_o \neq 0$ states that the points **P**,**A**,**B** and **C** are not coplanar.

$d_c \neq 0$, therefore the three equations may be rewritten as:

$$\begin{cases} \dfrac{d_a^2}{d_c^2} + \dfrac{d_b^2}{d_c^2} - \dfrac{d_a}{d_c}\dfrac{d_b}{d_c} 2\cos\gamma = \dfrac{d_{ab}^2}{d_c^2} \\ \dfrac{d_a^2}{d_c^2} + 1 - \dfrac{d_a}{d_c} 2\cos\beta = \dfrac{d_{ac}^2}{d_c^2} = \dfrac{d_{ab}^2}{d_c^2}\dfrac{d_{ac}^2}{d_{ab}^2} \\ \dfrac{d_b^2}{d_c^2} + 1 - \dfrac{d_b}{d_c} 2\cos\alpha = \dfrac{d_{bc}^2}{d_c^2} = \dfrac{d_{ab}^2}{d_c^2}\dfrac{d_{bc}^2}{d_{ab}^2} \end{cases} \tag{3.53}$$

To simplify the equations, let:

$$\begin{cases} x = \dfrac{d_a}{d_c}, \qquad y = \dfrac{d_b}{d_c} \\ \\ v = \dfrac{d_{ab}^2}{d_c^2}, \qquad s = \dfrac{d_{bc}^2}{d_{ab}^2}, \qquad t = \dfrac{d_{ac}^2}{d_{ab}^2} \\ \\ p = 2\cos\alpha, \quad q = 2\cos\beta, \quad r = 2\cos\gamma \end{cases} \tag{3.54}$$

After substitution, only x, y and v are unknowns, and the equations reduce to:

$$\begin{cases} x^2 + y^2 - xyr - v = 0 \\ x^2 + 1 - xq - tv = 0 \\ y^2 + 1 - yp - sv = 0 \end{cases} \tag{3.55}$$

In the equation, $|r| = |2\cos\gamma| < 2$ since $0 < \gamma < \pi$. Therefore, $v = x^2 + y^2 - xyr$ is contained between $x^2 + y^2 - 2xy < v < x^2 + y^2 + 2xy \implies 0 \leqslant (x-y)^2 < v < (x+y)^2 \implies v > 0$.

$v > 0$ implies that the variable d_c may be uniquely determined in the future as $d_c = d_{ab}/\sqrt{v}$. Therefore, the first equation is used to express v as a function of the other two unknowns $v = v(x, y)$ and is removed from the system.

$$\begin{cases} (II - tI) & : & (1-t)x^2 - ty^2 - qx + trxy + 1 = 0 \\ (III - sI) & : & (1-s)y^2 - sx^2 - py + srxy + 1 = 0 \end{cases} \tag{3.56}$$

As a consequence, P*3*P is reduced to the solution of two quadratic equations. The quadratic equations possess at most four physical solutions, or infinite solutions without reality conditions. To solve the ambiguity, the code evaluates the depth of all solutions and keeps only positive depths, as the observed points must be in front of the camera.

The three distances d_a, d_b and d_c are used to retrieve the pose of the camera with respect to the global system. Given the unit vectors $\hat{u}_a$, $\hat{u}_b$ and $\hat{u}_c$, the relative position of $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$ in the camera reference system is calculated in inhomogeneous coordinates.

$$\begin{cases} \tilde{A}_P = d_a \hat{u}_a \\ \tilde{B}_P = d_b \hat{u}_b \\ \tilde{C}_P = d_c \hat{u}_c \end{cases} \tag{3.57}$$

Given the points $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$ expressed in inhomogeneous coordinates in the global reference system, the absolute orientation R and translation t are found as the solutions of the equations:

$$\begin{cases} \tilde{A} = R\tilde{A}_P + t \\ \tilde{B} = R\tilde{B}_P + t \\ \tilde{C} = R\tilde{C}_P + t \end{cases} \tag{3.58}$$

The system of equations contains $3 \times 3 = 9$ equations and $3 + 3 = 6$ unknowns. 3 unknowns are related to camera orientation and 3 to camera location. Therefore, the solution is found with redundancy against noise. In addition, the code checks the reprojection error produced by the known three-dimensional points to identify the outliers.

P*n*P is combined with triangulation to obtain the motion of the camera and a three-dimensional representation of the scene. The process is self-sustaining but, in case no matches are found between projections and three-dimensional points, it could resort again to the estimation of the essential matrix.

3.4 Bundle Adjustment

Following the PnP+triangulation sequence, a set of three-dimensional points $\mathbf{X}_i, i \in \{1, ..., n\}$ and a set of projection matrices $\mathbf{P}^j, j \in \{1, ..., m\}$ are obtained. The sets describe, respectively, the scene and the pose of the camera. However, both representations contain errors, since the measurements are inherently affected by noise. Therefore, given a set of projections $\mathbf{x}_i^j$ of $\mathbf{X}_i$ in the scenes j, the projective map is not satisfied:

$$\mathbf{x}_i^j \neq \mathbf{P}^j \mathbf{X}_i \; in \; General \tag{3.59}$$

To increase accuracy, the sets $\mathbf{X}_i$, $\mathbf{P}^j$, and $\mathbf{x}_i^j$ are introduced in the cost function of an optimization problem [15]. Assuming the error to be mostly Gaussian, the approach is called maximum likelihood and works the same way as the two-view optimal solution 3.2. The approach computes the camera matrices $\bar{\mathbf{P}}^j$ and the three-dimensional points $\bar{\mathbf{X}}_i$ that minimize the re-projection error, which is the difference between the measured projections $\mathbf{x}_i^j$ and the projections of the estimated three-dimensional points $\mathbf{P}^j \mathbf{X}_i$.

$$\min_{\mathbf{P}^j, \mathbf{X}_i} \sum_{i=1}^{n} \sum_{j=1}^{m} d(\mathbf{P}^j \mathbf{X}_i, \mathbf{x}_i^j)^2 \tag{3.60}$$

Where $d(\cdot, \cdot)$ is the Euclidean distance between homogeneous points.

Optimization involves a large number of parameters. In fact, each camera matrix $\mathbf{P}^j$ has at least 11 degrees of freedom, and every point $\mathbf{X}_i$ has 3 degrees of freedom, for a total of $3n + 11m$ parameters as a lower limit. To reduce computational cost, it is common to divide the complete video in multiple steps and then merge the results. The solution is computed using the Levenberg-Marquardt optimizer and requires good initialization.

3.5 Camera Calibration

Despite being discussed at the end, calibration is performed at the beginning of the 3^{rd} step of pose estimation. The process refers to the estimation of the intrinsic matrix K of the camera, which is evaluated using a known set of three-dimensional points $\mathbf{X}_i, i \in \{1, ..., n\}$ and their corresponding projections $\mathbf{x}_i$ [6].

Consider a single homogeneous point $\mathbf{X} = (X, Y, Z, 1)^T$ in the global reference system and its projection $\mathbf{x} = (x, y, 1)^T$. $\mathbf{X}$ is projected to $\mathbf{x}$ through the 3×4 projective map $\mathbf{P}$.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.61}$$

Where P_{ij} are unknowns $\forall i, j$. The map corresponds to a linear system of equations:

$$\begin{cases} x = XP_{11} + YP_{12} + ZP_{13} + P_{14} \\ y = XP_{21} + YP_{22} + ZP_{23} + P_{24} \\ 1 = XP_{31} + YP_{32} + ZP_{33} + P_{34} \end{cases} \tag{3.62}$$

Which may be reduced to two homogeneous equations:

$$\begin{cases} I - xIII & : & XP_{11} + YP_{12} + ZP_{13} + P_{14} - xXP_{31} - xYP_{32} - xZP_{33} - xP_{34} & = 0 \\ II - yIII & : & XP_{21} + YP_{22} + ZP_{23} + P_{24} - yXP_{31} - yYP_{32} - yZP_{33} - yP_{34} & = 0 \end{cases} \tag{3.63}$$

The system contains 2 equations in 12 apparent unknowns. However, since the map $\mathbf{P}$ connects two homogeneous representations, it is defined up to a scale. For this reason, P_{34} is set to $P_{34} = 1$, reducing the number of unknowns to 11. As a consequence, the solution requires at least $n \geq 6$ three-dimensional points $\mathbf{X}_i$ and projections $\mathbf{x}_i$ to estimate $\mathbf{P}$. In case there are more than 6 points $n > 6$, the linear system becomes overdetermined and is solved using least squares.

The map $\mathbf{P}$ consists of two matrices: the intrinsic matrix K and the camera pose $[R \mid t]$. The camera pose is strictly related to the current view of the three-dimensional points $\mathbf{X}_i$, while the intrinsic matrix K is valid for any scene and includes the effects of lens distortion. Therefore, considering a single camera to implement pose estimation, K is constant for every view $K = K'$. However, the formal distinction is maintained in the previous chapters for the general case of pose estimation using different cameras.

Calibration is usually performed using Zhang's calibration method [37], which is applied to multiple views of a checkerboard. The code evaluates how the projections $\mathbf{x}_i$ are distributed in the image plane. In this case, the projections $\mathbf{x}_i$ and $\mathbf{X}_i$ are related by a homography, since $\mathbf{X}_i$ lie on the same plane $\forall i$. Therefore, the homography between the projections $\mathbf{x}_i$ and $\mathbf{X}_i$ may be estimated using multiple projections $\mathbf{x}_i^j$ and knowing the planar distance of the features $\mathbf{X}_i$ on the board.

Subsequently, once the homography has been retrieved, it is used to calculate the projection matrix $\mathbf{P}$.

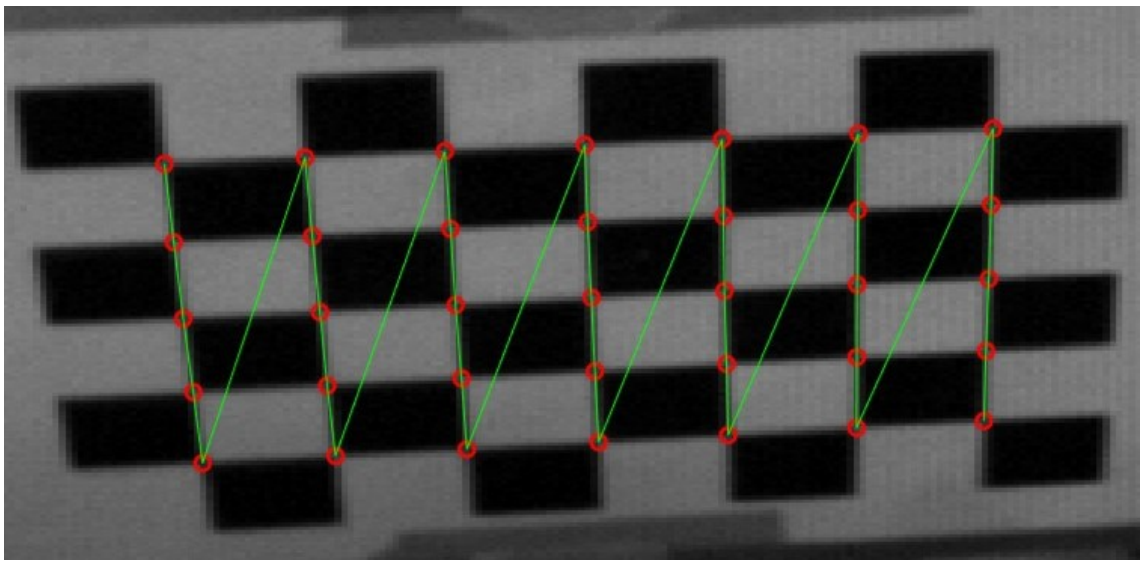


Figure 3.10: Detected Points on a Checkerboard

Calibration algorithms require correspondences between features and projections $\mathbf{X}_i \longleftrightarrow \mathbf{x}_i^j$. However, once feature detection and tracking have been performed, the projections $\mathbf{x}_i^j$ are represented in the same way on both event and traditional devices. Therefore, the codes do not distinguish features acquired with an event camera from features acquired with a traditional one, as any other algorithm used in the third step of pose estimation.

For this reason, Zhang's algorithm could be applied to both representations in theory. However, event cameras require the checkerboard to move, and only perpendicular lines are detected during motion 3.5. Consequently, the E-Calib [32] algorithm implements an equivalent procedure on a planar circle grid, and E2Calib [28] uses blinking LEDs. Both strategies simplify the detection of features compared to straight lines and, particularly, LEDs may be detected on static scenes.

Last but not least, in the case that the event camera is a dual acquisition system events&frames, the camera lens system is shared by both devices. Therefore, K is usually retrieved with traditional frames.

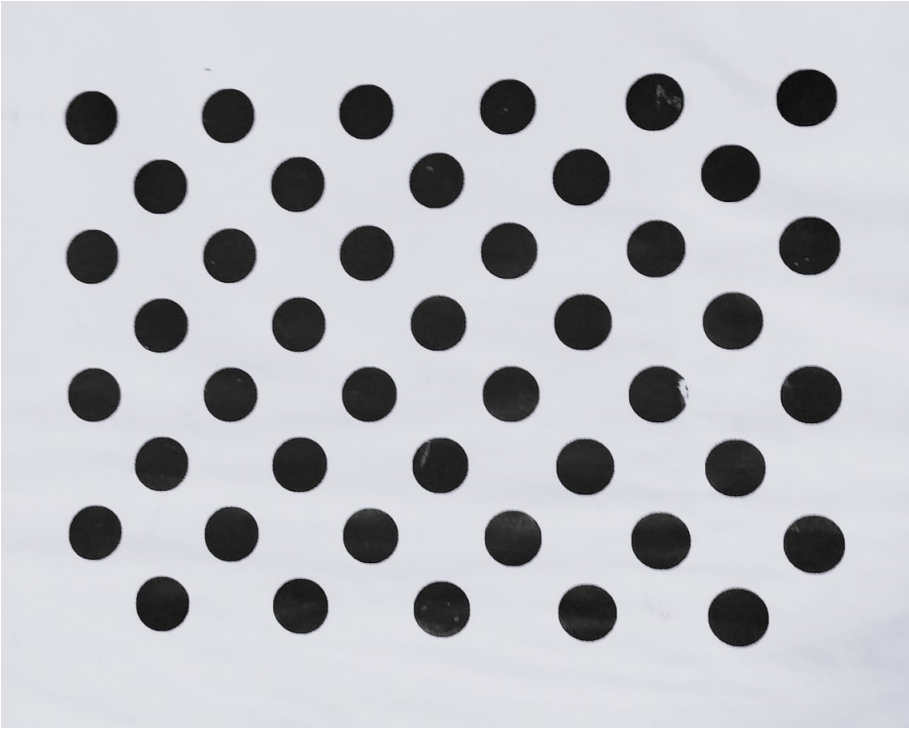
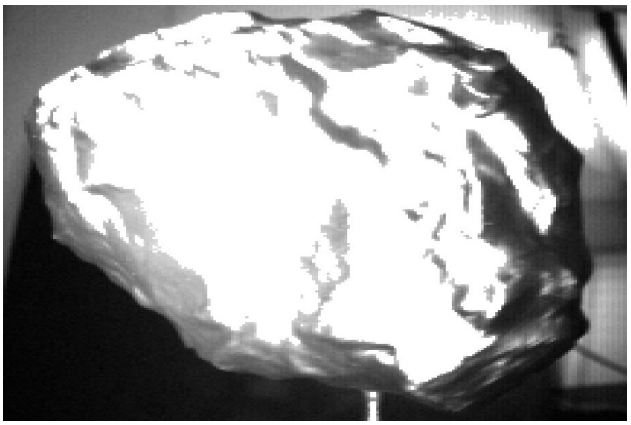


Figure 3.11: Planar Circular Grid

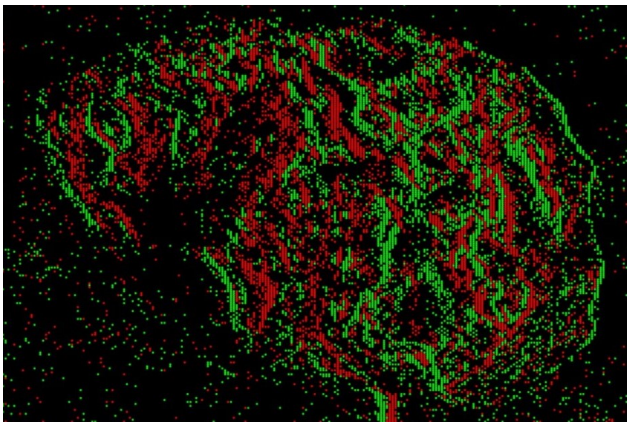
Chapter 4

Conclusions

This work presents a complete procedure for estimating the trajectory of a spacecraft and the shape of a celestial body from the acquisitions of an event camera. Celestial bodies explorations are one of the most challenging missions that are performed in space, and navigation is crucial for their success. For this reason, event cameras may be employed: not only do they reduce the required power, but also they increase the ability of the satellite to study the surface.



(a) Image Frame



(b) Event Frame

Figure 4.1: Comparison of Dynamic Range and Motion Blur

The new device requires new algorithms to complete its operations, and this thesis studied the best practical implementations of feature detection and tracking. The algorithms were tested on their accuracy, computational cost, real-time performance, as well as how well they could maintain a significant track of the feature evolution. The precision was specifically required for the context of the comet rotation, which is one of the most difficult scenarios that may be studied, since every point changes due to the change in perspective.

Numerous state-of-the-art strategies have been investigated, and the best have been implemented. Several approaches were based on the surfaces of active events SAE to maintain detections as asynchronous and localized as possible. Among these implementations, geometric and analytical strategies were performed. Additionally, a completely different approach was studied to achieve the best of both worlds: stable and reactive feature evolution by combining frames and events. Among the tests, several strategies obtained near real-time evaluation, without particular optimization protocols.

In addition to event feature detection and tracking, the complete process for simultaneously locating and mapping a three-dimensional scene was investigated and revised for event cameras. The process begins with the acquisition of an event stream and it concludes with the estimation of the trajectory of the moving spacecraft, as displayed in the image.

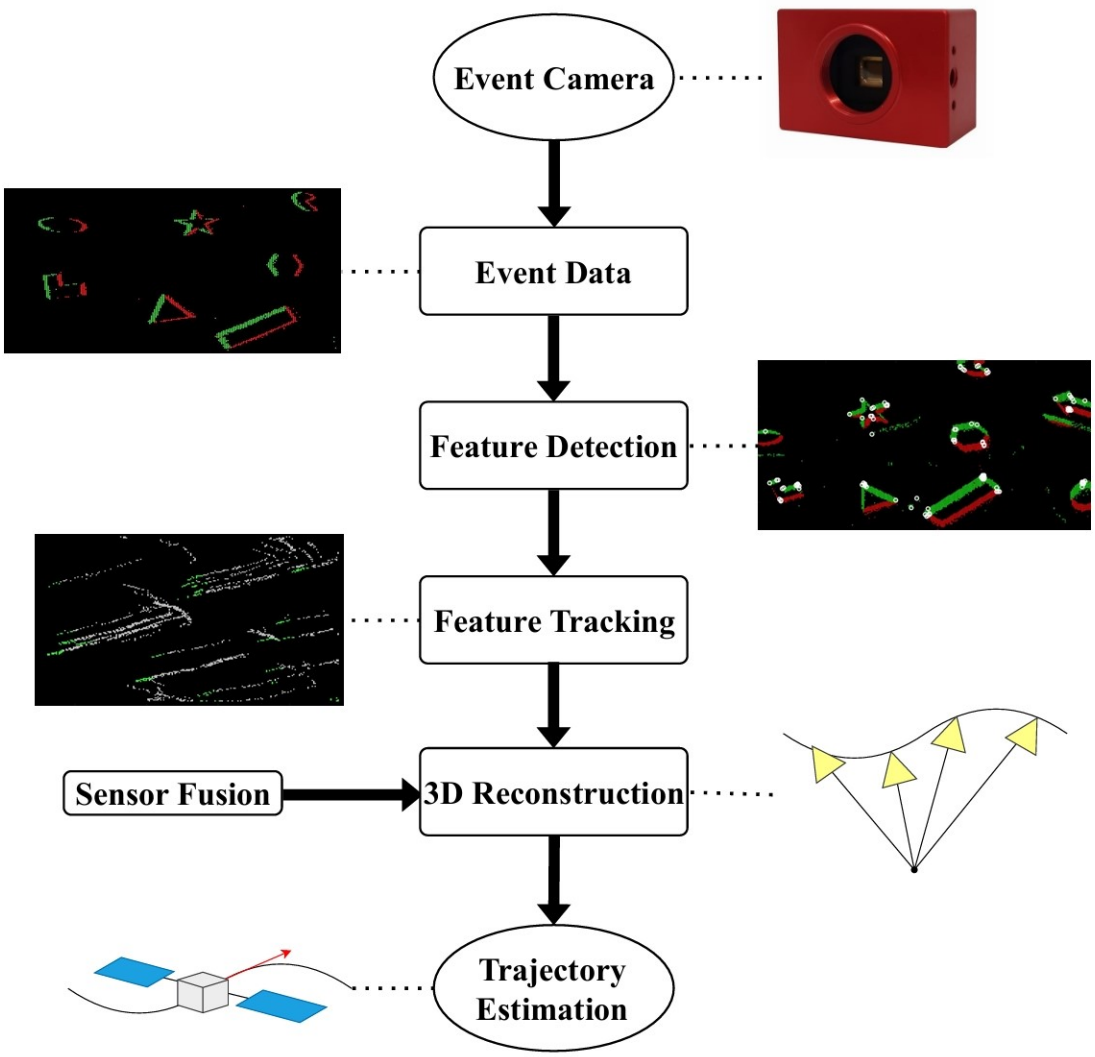


Figure 4.2: Main Steps in Pose Estimation and 3D Reconstruction

Bibliography

[1] Ignacio Alzugaray and Margarita Chli. Asynchronous corner detection and tracking for event cameras in real time. *IEEE Robotics and Automation Letters*, 3(4):3177–3184, 2018.

[2] Ignacio Alzugaray and Margarita Chli. Asynchronous multi-hypothesis tracking of features with event cameras. In *2019 International Conference on 3D Vision (3DV)*, pages 269–278. IEEE, 2019.

[3] Ignacio Alzugaray and Margarita Chli. Haste: multi-hypothesis asynchronous speeded-up tracking of events. In *31st British Machine Vision Virtual Conference (BMVC 2020)*, page 744. ETH Zurich, Institute of Robotics and Intelligent Systems, 2020.

[4] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[5] Sebastiano Chiodini, Robert G Reid, B Hockman, Issa AD Nesnas, Stefano Debei, and Marco Pavone. Robust visual localization for hopping rovers on small bodies. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 897–903. IEEE, 2018.

[6] Peter I Corke, Witold Jachimczyk, and Remo Pillat. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

[7] Jingyun Duo and Long Zhao. An asynchronous real-time corner extraction and tracking algorithm for event camera. *Sensors*, 21(4):1475, 2021.

[8] Ethan Elms, Yasir Latif, Tae Ha Park, and Tat-Jun Chin. Event-based structure-from-orbit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19541–19550, 2024.

[9] Yang Feng, Hengyi Lv, Hailong Liu, Yisa Zhang, Yuyao Xiao, and Chengshan Han. Event density based denoising method for dynamic vision sensor. *Applied Sciences*, 10(6):2024, 2020.

[10] Guillermo Gallego, Christian Forster, Elias Mueggler, and Davide Scaramuzza. Event-based camera pose tracking using a generative event model. *arXiv preprint arXiv:1510.01972*, 2015.

[11] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943, 2003.

[12] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. Eklt: Asynchronous photometric feature tracking using events and frames. *International Journal of Computer Vision*, 128(3):601–618, 2020.

[13] Arren Glover, Aiko Dinale, Leandro De Souza Rosa, Simeon Bamford, and Chiara Bartolozzi. luvharris: A practical corner detector for event-cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10087–10098, 2021.

[14] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988.

[15] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[16] iniVation AG. DAVIS346. https://inivation.com/wpcontent/uploads/2019/08/DAVIS346.pdf, August 2019. Accessed: October 8,2025.

[17] iniVation AG. DV software documentation. https://docs.inivation.com/software/dv/index.html, August 2025. Last updated: 2025-08-05; Accessed: October 8,2025.

[18] Haiyan Jiang, Xiaoshuang Wang, Wei Tang, Qinghui Song, Qingjun Song, and Wenchao Hao. Event stream denoising method based on spatio-temporal density and time sequence analysis. *Sensors*, 24(20):6527, 2024.

[19] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR 2011*, pages 2969–2976. IEEE, 2011.

[20] Hongdong Li and Richard Hartley. Five-point motion estimation made easy. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 1, pages 630–633. IEEE, 2006.

[21] Ruoxiang Li, Dianxi Shi, Yongjun Zhang, Kaiyue Li, and Ruihao Li. Fa-harris: A fast and asynchronous corner detector for event cameras. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6223–6229. IEEE, 2019.

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

[23] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.

[24] Jacques Manderscheid, Amos Sironi, Nicolas Bourdis, Davide Migliore, and Vincent Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10245–10254, 2019.

[25] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. Fast event-based corner detection. 2017.

[26] Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. Continuous-time trajectory estimation for event-based vision sensors. 2015.

[27] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International journal of robotics research*, 36(2):142–149, 2017.

[28] Manasi Muglikar, Mathias Gehrig, Daniel Gehrig, and Davide Scaramuzza. How to calibrate your event camera. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1403–1409, 2021.

[29] Issa AD Nesnas, Benjamin J Hockman, Saptarshi Bandopadhyay, Benjamin J Morrell, Daniel P Lubey, Jacopo Villa, David S Bayard, Alan Osmundson, Benjamin Jarvis, Michele Bersani, et al. Autonomous exploration of small bodies toward greater autonomy for deep space missions. *Frontiers in Robotics and AI*, 8:650885, 2021.

[30] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[31] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[32] Mohammed Salah, Abdulla Ayyad, Muhammad Humais, Daniel Gehrig, Abdelqader Abusafieh, Lakmal Seneviratne, Davide Scaramuzza, and Yahya Zweiri. E-calib: A fast, robust, and accurate calibration toolbox for event cameras. *IEEE Transactions on Image Processing*, 33:3977–3990, 2024.

[33] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[34] Olaf Sikorski, Dario Izzo, and Gabriele Meoni. Event-based spacecraft landing using time-to-contact. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1941–1950, 2021.

[35] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156, 2000.

[36] Valentina Vasco, Arren Glover, and Chiara 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)*, pages 4144–4149. IEEE, 2016.

[37] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2002.

[38] Jinxiu Zhao, Li Su, Xiangyu Wang, Jinjian Li, Fan Yang, Na Jiang, and Quan Hu. Dtfs-eharris: a high accuracy asynchronous corner detector for event cameras in complex scenes. *Applied Sciences*, 13(9):5761, 2023.