Graduation Project

EVALUATION OF NEUROMORPHIC IMAGE SENSORS FOR VISUAL ODOMETRY

Sergio PERTIERRE DO MONTE
SPID – Robotic Engineering

Supervisors:
Ahmet ŞEKERCİOĞLU
Franck DAVOINE
Vincent FREMONT

August 24, 2017

Abstract

Differently from conventional cameras, the neuromorphic image sensors try to imitate the sophisticated human visual system, capturing the movement on the image with a micro-second precision. These relatively new sensors have a great potential to high speed robots and computer vision applications, and they solve the common problems from frame-based cameras: high-latency, redundancy and low temporal resolution. In essence, they asynchronously capture only the pixels which have changed their brightness levels, called "events". Then, because of their different approach of the scene, they require completely new algorithms. Since autonomous car localization problems need quick responses from the external environment, these sensors can play a decisive role in the vehicle's visual odometry. Therefore, the present work develop a feature tracking algorithm using the Dynamic and Active-pixel Vision Sensor (DAVIS) because it combines in the same array of pixels a normal grayscale frame-based stream, where the corners are detected, and an event-based sensor whose events are used to track them. Despite the noise interference, it is shown that the algorithm can find and track the strongest features.

Keywords
Event-based Camera, Feature tracking, Neuromorhic Sensor, Dynamic Vision Sensor, DAVIS, Autonomous Car, ROS, High Speed Robots.

Résumé

Contrairement aux caméras classiques, les capteurs neuromorphiques d'images essayent d'imiter le système complexe de la vision humaine, ils sont capables de capturer le mouvement sur l'image avec une précision de micro-seconde. Ces capteurs relativement nouveaux ont un grand potentiel pour les robots à haute vitesse et les applications de vision par ordinateur, et ils résolvent les problèmes courants des caméras basées sur des images : haute latence, redondance et faible résolution temporelle. En principe, ils ne capturent de manière asynchrone que les pixels qui ont changé leurs niveaux de luminosité, appelés «événements». Ensuite, en raison de leur approche différente de la scène, ils nécessitent des algorithmes complètement nouveaux. Étant donné que les problèmes de localisation de la voiture autonome nécessitent des réponses rapides de l'environnement externe, ces capteurs peuvent jouer un rôle décisif dans l'odométrie visuelle du véhicule. Par conséquent, le présent travail développe un algorithme de feature tracking à l'aide du Dynamic and Active-pixel Vision Sensor (DAVIS) car il combine dans le même tableau de pixels un flux basé sur des images en niveaux de gris, où les coins sont détectés et un capteur d'événements dont les événements sont utilisés pour les suivre. Malgré les interférences de bruit, il est démontré que l'algorithme peut trouver et suivre les features les plus fortes.

Keywords
Caméra Événementielle, Feature Tracking, Capteur Neuromorique, Capteur de Vision Dynamique, DAVIS, Voiture Autonome, ROS, Robots Haute Vitesse.

Acknowledgments

I would like to give my sincere thanks and appreciations to my supervisors Mr. Franck Davoine and Mr. Ahmet ŞEKERCİOĞLU, who have guided me during the whole project, offering their aid whenever I needed and giving important advices. Also, they taught me a lot in our weekly meetings, which have helped me to organize the work and set short-term goals. I am glad to have had them as supervisors.

I would like to thank Mr. Vincent FREMONT, who supervised and gave me valuable hints in my project. Also, special thanks to Mr. Philippe BONNIFAIT, who had helped me to find this internship by forwarding my email to Mr. DAVOINE.

Thanks to the personal of Heudiasyc and UTC for all their administration and technical support during the whole internship.

Moreover, I would like to express my gratitude to my parents and all my friends, who despite the distance, have never ceased to support me along the project.

Contents

List of Figures

Chapter 1

Introduction

Today's methods to deal with vision systems for robots use conventional frame-based cameras with satisfactory results. However, as normal cameras do not provide the information between frames, it is crucial to autonomous cars more sophisticated sensors with high-frequency measurements updates. The bio-inspired neuromorphic image sensors, also called event based cameras, have such characteristics, updating each pixel on the image that changes in a micro-second resolution. Therefore, the goal of the present project is to make use of these properties to make in ROS [1] environment a feature tracking function, which is an essential building block to vision applications such as visual odometry.

Presentation of Heudiasyc The host laboratory was created in 1981 and means Heuristics and Diagnostics of Complex Systems ("*HEUristique et DIAgnostic des SYstèmes Complexes*", in French). It is a mixed research unity between the University of Technology of Compiègne (UTC) and the National Center for Scientific Research (CNRS). The director Ali Charara and the adjoint-director Yves Granvalet ensure the laboratory governance with the aid of team leaders.

Heudiasyc's activity is based on the synergy between upstream research and finalized research, in order to meet the major challenges of the company: safety, mobility and transport, environment, health. They have a close collaboration with business partners, particularly industrial companies. Four teams are responsible for the scientific activities developed on the laboratory:

- ASER: Automation, Embedded Systems, Robotics
- DI: Decision, Image
- ICI: Information, Knowledge, Interaction
- RO: Networks, Optimization

Related work: Nowadays, feature detection and tracking algorithms for conventional cameras present good results. However, neuromorphic image sensors, also called event-based sensors, need new algorithms as they have a unique approach to the scene, they acquire asynchronously the relevant data information transmitting the brightness change of each pixel, what is called "event". As they have a micro-second resolution, they don't present the problem of blind time between frames as frame-based cameras. Therefore, event-based cameras can be used in many applications, such as high-speed control [5]. A simple event-based feature tracking is shown in [9]. In [11] a Iterative Closest Point (ICP) algorithm tracked a black polygonal microgripper on a white background. For the visual odometry in high-speed robots, it is preferable tracking several local features with determined positions.

In the previous work [4], DVS was used to do a low-latency event-based visual odometry, where it was put beside a CMOS camera, and together they could localize the robot. One of the challenges of this approach is to extrinsically calibrate these two sensors. So they devised an unsupervised spatiotemporal calibration technique which creates a virtual sensor. Nonetheless, as these devices do not have the same resolution, focus centers and timestamps, they needed to do several approximations. Then, based in this relatively accurate virtual sensor, they could make an event-based visual odometry method, which is based on Markov localization, where each event defines a probability function that weights the relative motion estimate. Finally, the system is robust to motion blur and can well estimate the rotation. However, the results for translation are quite noisy because the

[1]Robot Operating System

limitations of the DVS. The approach in [14] and its improved version [8] detect the features in the grayscale image and tracks them using the event camera DAVIS [2]. The present work is based on these principles, which will be deepened in chapter 4 and chapter 5.

Motivation and approach: For self driving cars, several sensors such as GPS and scanner lasers are integrated to improve the localization system. Therefore, as the event-based camera has no latency and time discretization problem, it is interesting to have this sensor aboard. Thus, the present work aims the development of a feature tracking algorithm that can communicate by ROS nodes.

As mentioned, the basic idea is to detect the features in a normal image frames, and then track them using only the information of the brightness change in the scene, which indicates the movement. This approach keeps the low latency property of the camera, proper for autonomous high-speed robots.

Outline: The report is organized as follows. In chapter 2 the event-based cameras and their use are detailed deeply. Inside chapter 3, the ROS environment is explained and its application in the project. chapter 4 and chapter 5 describe the proposed approach for feature detection and tracking respectively. chapter 6 discusses the effectuated experiments. Finally, chapter 7 recapitulates the project and gives some hints for future works.

Chapter 2

Event-Based Camera Review

Neuromorphic engineering is a new research area, and their concepts were first established by Carver Mead in 1989 at CalTech, USA. The way how neurons communicate has inspired scientists to design VLSI[1] and programs able to execute complex computations that copy behaviors from the living beings in an autonomous way. An example of neuromorphic device, used in this work, is the silicon retina: a human vision based sensor to capture movements.

Over the last decades, several improvements were made in conventional cameras, which the main idea is to sample an image of the scene at a constant time rate. Although it has presented satisfactory outcomes in the many feature detection and track researches, there are some limitations from this cameras that do not allow them to be used in autonomous cars:

- Redundancy: They capture redundant visual information of the scene.
- Low discretization: They don't provide information between the frames.
- High latency: They need a considerable time to capture and process the last taken image.

Thus, as the agility of a robot relies on the discretization time and latency, these are decisive topics to be considered to work with high speed robots such as autonomous cars.

An event-based sensor, overcomes the problems cited above. Each pixel is independent from each other and asynchronously transmits in real time the brightness changes. This is called an "event", and the time between events have a micro-second resolution. Thus, in an event camera, the latency and discretization problems were solved in a hardware level. Also, there is no redundancy, once that such camera will just transmit the changes in the scene. However, as this new sensor has a different approach of the scene compared to frame-based cameras, conventional computer vision algorithms cannot be easy adapted and then, new algorithms need to be implemented to deal with this device.

2.1 Sensor Models

In nowadays, there are two main neuromorphic image sensors: Dynamic Vision Sensor (DVS) [6] and Asynchronous Time-based Image Sensor (ATIS) [13], which are better described in the following sections. For the present work, it was used a recorded dataset from a Dynamic and Active-pixel Vision Sensor (DAVIS) [2], which is an event-based camera that combines a CMOS camera and the DVS in the same array of pixels. This is useful to make real time feature tracking using both events and gray level image, and also there is no need to calibrate the two outputs, facilitating the process.

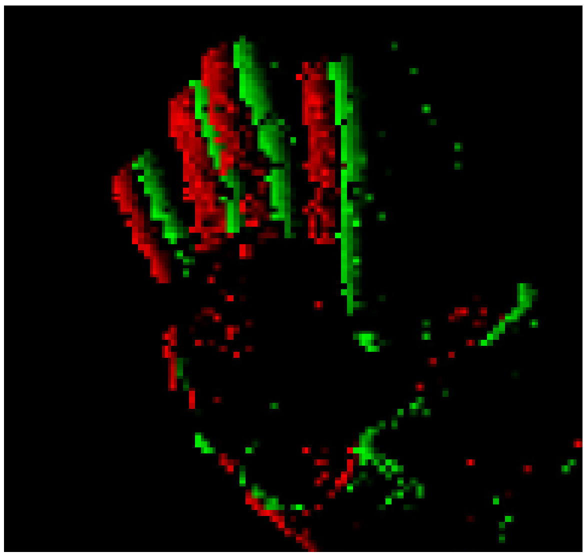
2.1.1 DVS

Biological vision systems are really quick and transmit information in an efficient way due its complexity, making these systems better than electronic vision devices in almost all aspects. Inspired by this model, ETHZ[2] (Swiss Federal Institute of Technology in Zurich) created the Dynamic Vision Sensor (DVS), see figure 2.1a. It is driven by the actions ("events") happening in the scene, differently from the conventional cameras, that have

[1]Very Large Scale Integrated Circuits

[2]Eidgenössische Technische Hochschule Zürich

(a) Dynamic Vision Sensor.



(b) A hand movement captured by the DVS. The green dots represent the positive events, and the red, the negatives.

Figure 2.1 – The DVS (left) and its output (right).

no relation to the incoming visual information and just shot pictures in a fixed sampling time interval. Thus, the neuromorphic image processing is called "event-based", while the conventional image based processing is called "frame-based" because of its constant time rate.

However, the sensor presents a resolution of 128×128 pixels, what is very low. In addition, as it can only capture the movement in the scene, if nothing moves and the camera stays stationary, there will be no output. Therefore, this sensor cannot be the alone inside the robot, but it can refine the data of another sensor, such as a laser rangefinder or a frame-based camera.

The DVS is an USB camera and its parameters can be adjust online. Also, There is no need of any previous knowledge about the electronic structure to be used in robotics applications. Indeed, this is the first prototype developed by ETHZ that can be used, almost every other neuromorphic laboratory invest heavily on proof-of-concept hardware development, which make the sensors hard to be handled by other users.

The output of this sensor is a sequence of asynchronous events. Each pixel creates an event when the signal scene reflectance increases or decreases according to a predetermined threshold at the time it occurs. In other words, when the change in log luminance surpass the predetermined threshold, an ON (positive polarity) or OFF (negative polarity) event is produced by the independent pixel. More precisely, the event output is a tuple $e = (x, y, t, p)$, which combines the spatiotemporal coordinates and the event's polarity $p = \pm 1$. As shown in figure 2.1b, where there is a hand doing a simple translation movement, the DVS captures only the pixels that have change its brightness, being unable to see the rest of the scene.

The effectiveness of the visual data compression rate depends on the dynamic objects of the scene in sight. More specifically, if $I(t)$ is the illumination captured by a pixel in a certain (x, y) position, an event will be triggered whether the logarithmic luminosity change surpasses a certain threshold: $|\Delta \ln I| := |\ln I(t) - \ln I(t - \Delta t)| > C$. Where Δt is the difference between the current and last event triggered at the same pixel.

Figure 2.2 exposes a simplified circuit for a DVS pixel. Each pixel circuit has three main parts:

- The photoreceptor circuit: Responds to logarithmic brightness changes
- The differencing circuit: Removes the DC current and set the initial illumination level after a event is generated. What makes the pixel sensitive just to temporal contrasts.
- The comparators: Using a predetermined voltage threshold, they generate "ON" or "OFF" events.

Therefore, with this design, only light changes in the scene can generate data. In addition for the polarity

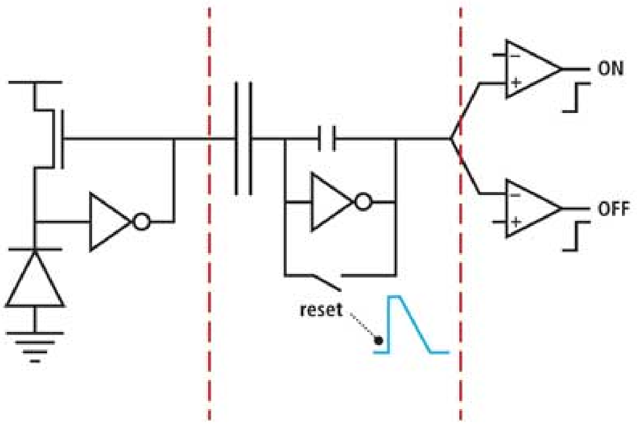


Figure 2.2 – DVS pixel's internal circuit

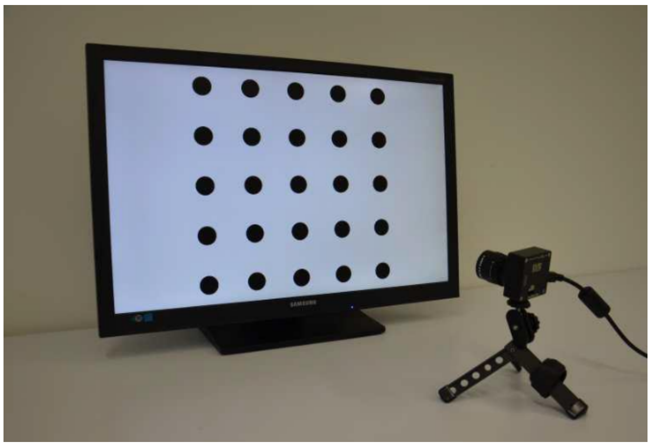


Figure 2.3 – Calibration of a DVS using blinking patterns in a computer screen.

information, each pixel will also send its location and the time when it occurred to a PGA[3] package.

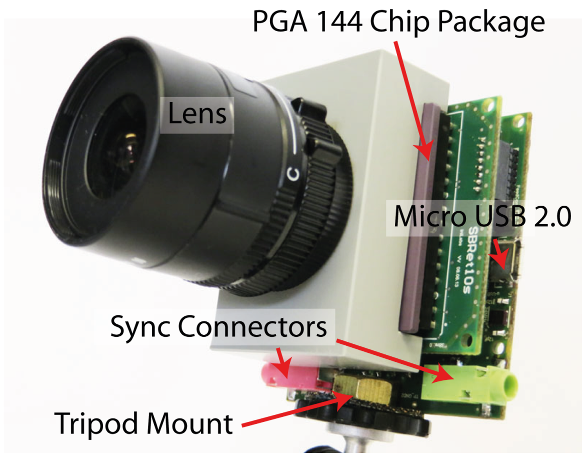
Calibration: To calibrate this kind of sensor, the standard pinhole camera model is still valid, but the standard passive calibration methods cannot be used once it would be need to move the camera. Normally, it is used blinking patterns with computer screens or LEDs, represented in figure 2.3. Also, there is an open source ROS DVS driver to calibrate the camera parameters[4].

2.1.2 DAVIS

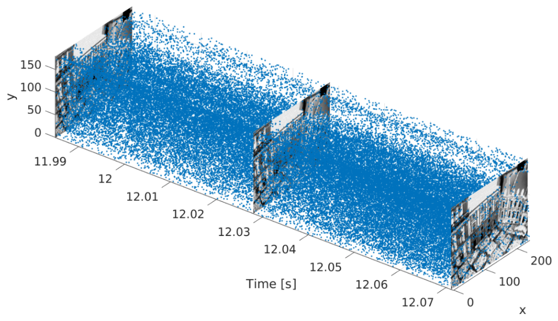
The Dynamic and Active-pixel Vision Sensor (DAVIS), see figure 2.4a, is an event-based sensor also developed by ETHZ. With several upgrades compared with its predecessor, this vision sensor is a powerful device that combines the DVS and a CMOS camera in the same array of pixels and have an on board IMU[5]. It means that with only a DAVIS, it is possible to do visual tasks such as object recognition and tracking to high-speed robotics. Also, the spatial resolution is 240×180 pixels, which is still low when compared with conventional cameras, but is better than the original DVS detailed in section 2.1.1.

As shown in figure 2.4b, the output of the DAVIS is the grayscale frames from the CMOS camera and the asynchronous events. It is interesting to see on this image the difference between the two sensors. The conventional camera takes several pictures over the time in a fixed sampling ratio, while DVS captures the pixels that change its luminosity in a ratio much higher, what clearly covers the blind time between two consecutive frames.

[3]Pin Grid Array
[4]`https://github.com/uzh-rpg/rpg_dvs_ros`
[5]Inerial Measurement Unit



(a) Dynamic and Active-pixel Vision Sensor



(b) The normal DAVIS output: it captures the grayscale frames as a conventional camera and the events (in blue), which are faster and asynchronous. Several events can be captured in the middle time between frames.

Figure 2.4 – The DAVIS (left) and its output over the time(right).

The fact that this newly device perfectly align the DVS and the grayscale pixels facilitates the camera calibration process, an essential stage for the visual applications, as shown in [4]. The CMOS camera provides the absolute illumination on demand and it is up to 24 Hz. In addition, the IMU sensor is placed 3 mm from the optical center of the camera and is aligned to the camera axis, which is interesting to applications such as object speed estimation.

UZH has developed a ROS driver capable to calibrate online the intrinsic and extrinsic camera parameters while plugged to the computer. Its installation has no problems in ROS kinect distribution (Ubuntu 16.04), however for indigo distribution (Ubuntu 14.04) it presented some issues that could be solved after a discussion in the GitHub page[6]. The figure 2.5 refers to this driver.

2.1.3 ATIS

The ATIS sensor is the most recent prototype of silicon retinas, it has an array of independent pixels from each other and they transmit the movement of the scene, as the eye vision system. The electronic circuit captures these changes at the time they occurred and also the light intensity information through a conditional exposure measurement device. Therefore, its output is a stream of spikes encoding both visual informations in the same way that the brain can interpret.

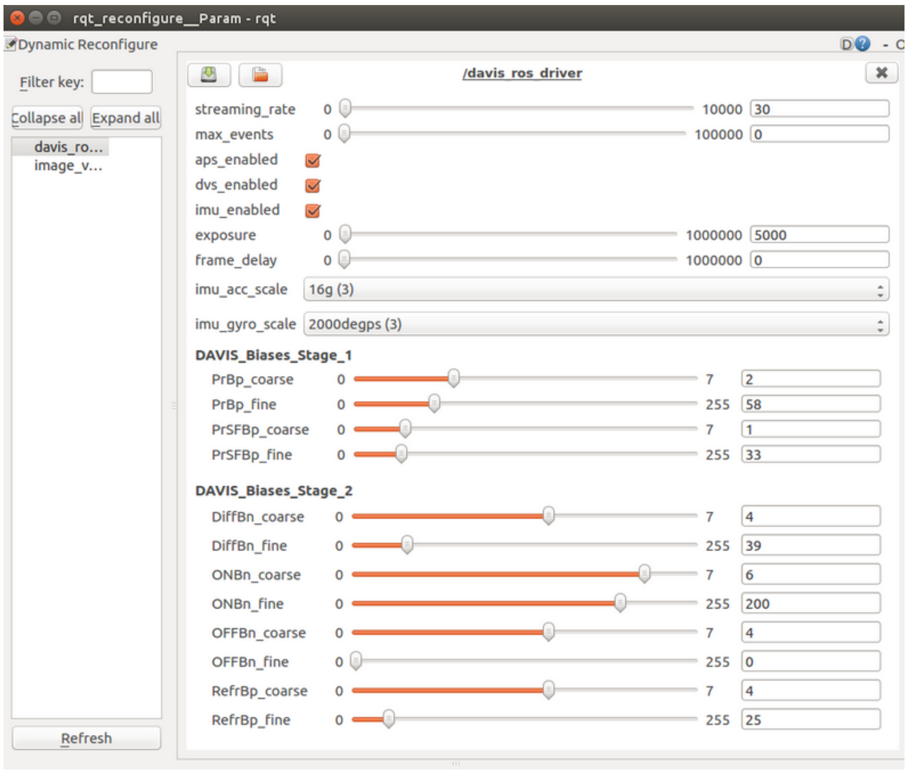
Despite there are some published studies using this sensor, such as [13] and [12]. ATIS does not have yet much scientific support compared to its concurrents DVS and DAVIS, what is one of its disadvantages.

However, differently from the DVS and DAVIS, this neuromorphic sensor can obtain the grayscale luminance level of each pixel that has generated an event. Therefore a frame-based camera is not needed for this sensor and it is able to reconstruct the grayscale image over time.

The original idea of the internship wa to work with a camera from the company Chronocam[7], which has an ATIS integrated. Nonetheless, as they could not send this sensor in time, it was necessary to use a dataset recorded from a DAVIS. This company has also developed a software to calibrate the camera parameters.

Chronocam recorded one hour on the road using their camera, what is interesting to see since there are not too many outdoor examples available using these sensors. This can be seen in figure 2.6, where the above image shows the output events and the other, the reconstructed grayscale image.

[6]`https://github.com/uzh-rpg/rpg_dvs_ros/issues/36`
[7]French company that develops a event-based camera in Paris.



Parameter	Min	Max	Value
streaming_rate	0	10000	30
max_events	0	100000	0
aps_enabled			✓
dvs_enabled			✓
imu_enabled			✓
exposure	0	1000000	5000
frame_delay	0	1000000	0
imu_acc_scale			16g (3)
imu_gyro_scale			2000degps (3)
DAVIS_Biases_Stage_1			
PrBp_coarse	0	7	2
PrBp_fine	0	255	58
PrSFBp_coarse	0	7	1
PrSFBp_fine	0	255	33
DAVIS_Biases_Stage_2			
DiffBn_coarse	0	7	4
DiffBn_fine	0	255	39
ONBn_coarse	0	7	6
ONBn_fine	0	255	200
OFFBn_coarse	0	7	4
OFFBn_fine	0	255	0
RefrBp_coarse	0	7	4
RefrBp_fine	0	255	25

Figure 2.5 – The DAVIS driver developed by UZH, where is possible to see all the camera parameters that the user is able to adjust.

2.2 Advantages of Event-based Vision

In this chapter it was possible to see several advantages brought by this new approach from the scene, what has motivated different studies in robotics and computer vision applications. Thus, this section will summarize these benefits.

2.2.1 Frame and Event Acquisition

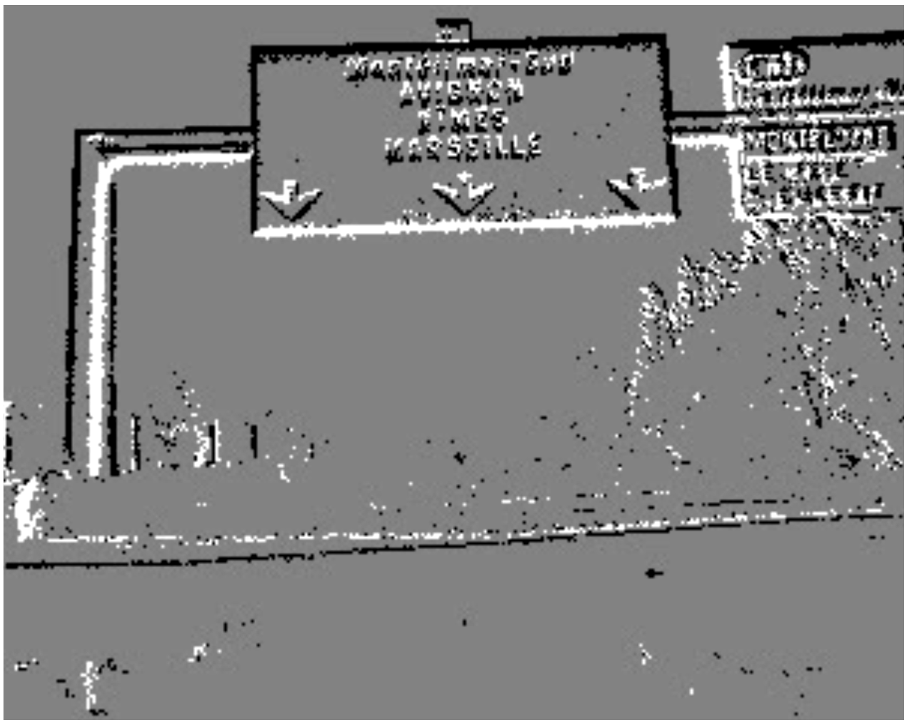
Let $I(t)$ be a frame from a conventional camera in the instant t, then, the image sequence will be $I = \{I(t_0), I(t_0 + \Delta t), \ldots, I(t_0 + k\Delta t)\}$. As reported, there is a blind time, Δt, corresponding to the time between frame acquisition that event-based cameras can see due to its high temporal resolution of 1 μs. Also, if there is no change happening on the recorded video, the frames $I(t_n)$ and $I(t_n + \Delta t)$ will be equal, what is an unnecessary computational cost. The events overcome this redundancy problem as they are only recorded when there are changes. In addition, sensors like DVS have a large dynamic range (130dB) when compared with frame-based cameras (around 70dB). Dynamic range is the ability of capturing light intensity without saturation.

Other advantages that neuromorphic sensors can bring are its low power consumption of only 23 mW and the low latency[8] of 15 μs provided by the DVS, for example.

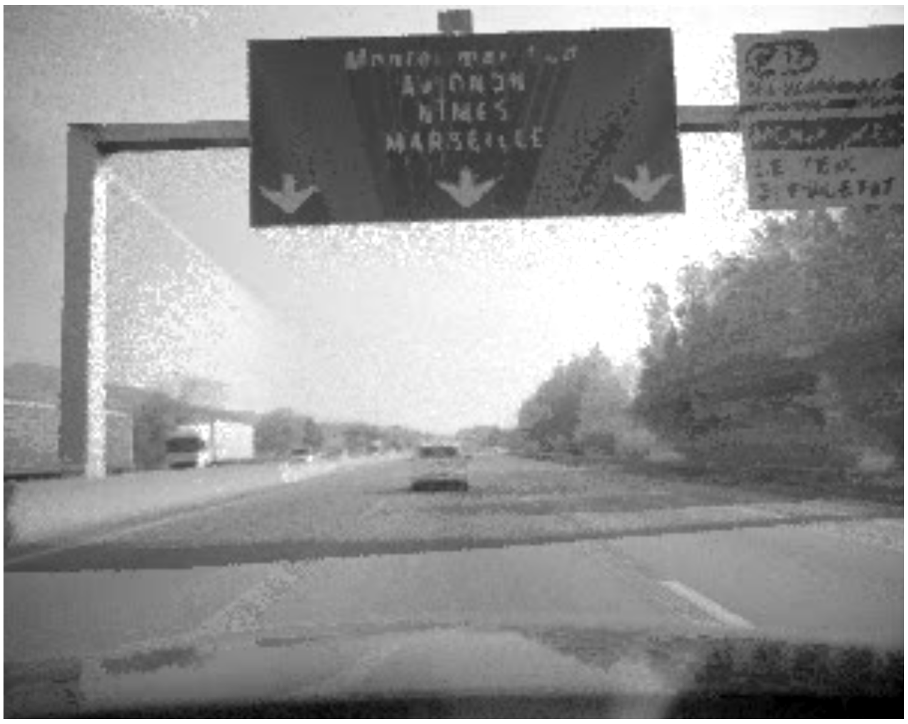
2.2.2 Event Processing

Event based algorithms are more robust in detection. The DVS only captures the dynamic object, eliminating the rest, therefore it reduces the necessity of image segmentation, what takes to much time and it is subject to errors in conventional frame-based methods. These algorithms don't consume energy while there is no changes in scene, a very relevant characteristic in robotics. Finally, they also allow time-based programming, which is ignored by frame-based algorithms, being only able to do spatial correlation between pixels.

[8]Time required to obtain and process the last frame



(a) The events coming from the ATIS sensor, where white pixels represent the positive brightness change, and the black represent the negative.



(b) The rebuilt scene using the data from the grayscale values took by ATIS.

Figure 2.6 – Outdoors recorded data by Chronocam.

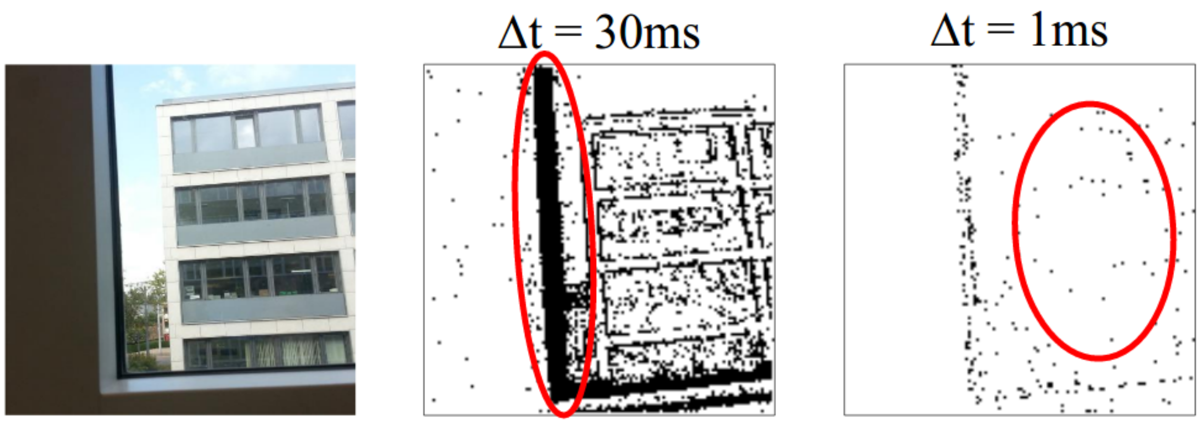


Figure 2.7 – Different event lifetimes.

2.3 Event-Based Computation

The event-based notions already presented in this chapter are important for the understanding of the present work. Nonetheless, it is also necessary to show the formal mathematical formulation[9] of the event-based programming, presented below.

Let a grayscale level image be $I(t)$. It will be composed by a $M \times N$ matrix that measures the light intensity captured by the sensor. Thus, if all the pixels belong to the function $f(x, y, t)$, where (x, y) are their corresponding positions, then $I(t)$ can be written as follow:

$$I(t) = \{f(0, 0, t), f(1, 0, t), \ldots, f(M-1, N-1, t)\}$$

Where $t \in [t_0, t_0 + k\Delta t]$. This classical approach to the scene has redundant information when it comes to static objects. The event-based solution for this problem removes the constant $f(x, y, t)$ terms, it means, when $\dot{f}(x, y, t) \approx 0$. This corresponds to irrelevant luminosity changes. Therefore, the mathematical definition for an event is:

$$e(x, y, t) = \begin{cases} sign(\dot{f}(x, y, t)), & \text{if } \dot{f}(x, y, t) > |\Delta T| \\ \emptyset, & \text{if } \dot{f}(x, y, t) < |\Delta T| \end{cases}$$

ΔT being a predefined luminosity change threshold. The two possible values that an event can assume (± 1) represent the contrast changes. Inspired in the neurons model from the body, each event has a certain influence which decays over time. Thus, let model a set of events $G(t)$ with an exponential decay function, demonstrating the event weight lose:

$$G(t) = \left\{ e(x, y, t_i) | e^{\frac{t - t_i}{\tau}} > \delta \right\}$$

Where δ is the threshold, t_i the instant where the event was generated and τ a constant for the decay function.

However, for most applications, the weight of the decay function assumes binary values, creating the concept of *event lifetime*, which is the event active time in the system. Visually, it can be understood in figure 2.7[10]. The events are stored in a buffer in a FIFO system. When an event stays longer than a predetermined threshold time $\Delta t = \tau \ln(\delta)$, it is removed from the buffer. The figure also shows the issues when choosing different time thresholds: the large integration causes motion blur, while the small causes sparsity.

2.4 UZH Dataset

These event-based cameras, as it was demonstrated, have a different output than normal cameras. Therefore, a paradigm shift is needed to deal with it. However, this kind of sensor is expensive, so it is interesting to have some datasets to deal with it before buying one.

[9]Based in the mathematical approach given in `https://tel.archives-ouvertes.fr/tel-00916995/document`
[10]Figure extracted from `http://www.rit.edu/kgcoe/iros15workshop/papers/IROS2015-WASRoP-Invited-04-slides.pdf`

File	**Description**	**Line Content**
events.txt	One event per line	*timestamp x y polarity*
images.txt	One image reference per line	*timestamp filename*
images/00000000.png	Images referenced from images.txt	
imu.txt	Frames per each timestamp	*timestamp ax ay az gx gy gz*
groundtruth.txt	A ground truth measurement per line	*timestamp px py pz qx qy qz qw*
calib.txt	Camera parameters	*fx fy cx cy k1 k2 p1 p2 k3*

Table 2.1 – A text format dataset .

Based on this issue, UZH[11] (University of Zurich) provides a DAVIS dataset in its site[12] to facilitate the comparison between event and frame based image methods to visual localization. The content of this dataset, presented in [10], has several files where is possible to treat the translation, rotation and external environments. However, one difficulty is that the ground truth to the outdoors recorded files used in this work were not provide, and due to vibrations, the IMU was very noisy, then it was not recorded.

The datasets are provided in text format and, for convenience to those using ROS, there is also a rosbag version. For this work, it was used the following rosbag files: *shapes_ rotation.bag*, *shapes_ translation.bag*, *urban.bag*, *slider_ depth.bag* and *slider_far* (rosbag files are better explained in the next chapter). A normal text output is detailed in table 2.1.

[11]Universität Zürich
[12]`http://rpg.ifi.uzh.ch/davis_data.html`

Chapter 3

ROS Environment

ROS is an open source middleware[1] to make robots execute some tasks. It was designed to be a common software platform for people working with robots, independently of their level of knowledge[2]. The community shares ideas and code, what facilitates the programming, once there is no need to develop and maintain all the code by oneself, saving time and letting focus in parts of the system that one wants to work on. Thus, this chapter introduces the basic concepts of ROS and its use through the present project[3]. Because of its advantages, this frameware is popular in academic research, being mentioned in several papers.

Fundamentally, a ROS environment has some codes, called "nodes", that communicate between them by publishing and subscribing to a certain type of information called "message". For example, in this work it was needed to create a new kind of message for "events" published by the DVS. Then, the main algorithm subscribes to this incoming information in order to be able to use it. After all the treatment, it publishes a final image, which the node "*image_ view*" subscribes to visualize it. The ROS hierarchy is illustrated by figure 3.1.

Catkin workspace: Catkin contains a set of macros and custom python scripts to provide extra functionality on top of the normal CMake flow. Every Catkin will have a "*src*" folder where the package will be created. Doing "*catkin_ make*" will generate a "*build*" folder containing results of its work as libraries (more relevant when working with C++ algorithms), and a "*devel*" folder, which contains the setup for the whole workspace and codes within.

ROS packages: A package contains some data, documentation and code, and it could depend on other packages as well. The command "*catkin_ create_ package*" makes a directory with a CMakeLists.txt, a package XML and a "*src*" directory. The XML contains a bunch of metadata and has the most part of the package documentation. The most important ROS packages used in this project were: *catkin_ simple, image_ view, rosbag, roscpp, cv_ bridge* and *pcl.*

Rosrun: With the command "*rosrun*" it is possible to play some nodes (code) from some package. Then we can visualize the graph with Qt-based graph visualizer for ROS: *rqt_ graph.*

Names, namespaces and remapping: Names are a fundamental concept in ROS. Nodes, message streams (often called "topics") and parameters must all have unique names. For example, if "camera" is the name for a camera which sends "image" as message topic and read the parameter "frame_rate". To do not write the same code to another camera that could be used, there is the concept of namespace and remapping.

Namespace allows to make different files with the same name (in different folders). For example: .../image_right/image. And remapping allows to send these data stream to another program without needing to modify the source code of the program.

Roslaunch: It is a XML command-line tool designed to automate the launching of collections of ROS nodes.

Rosbag: Rosbag is a binary file that allows the user to play recorded data and work with it. This is useful when it there is not possible to use the robot or the sensor.

[1]It is a software that serves as a bridge to connect database and programs on a network
[2]There are several tutorials for those who are just starting in robotics
[3]It was used the *indigo* ROS distribution (Ubuntu 14.04).

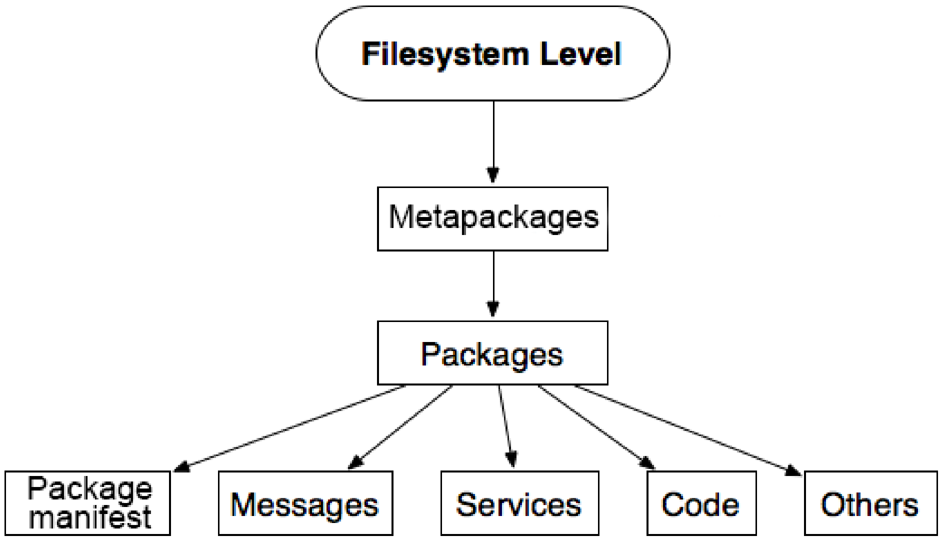


Figure 3.1 – ROS hierarchy.

3.1 Roscpp

Roscpp is the implementation of C++ in ROS, allowing the programmer to code nodes in this language. In addition, it was designed to be the high-performance library for ROS. However, despite the known fact that using C++ instead of Python might result in some performance improvements, one of the main motivations to use roscpp instead of rospy was the library PCL[4] (better explained in chapter 5), which is only available in C++.

In the Github page of UZH[5], it was found a roscpp package which was used to calibrate the DVS and DAVIS camera while it is connected. Also, there was a code used to play the rosbag files. Thus, this package was used as base reference to the ROS structure program of the present work.

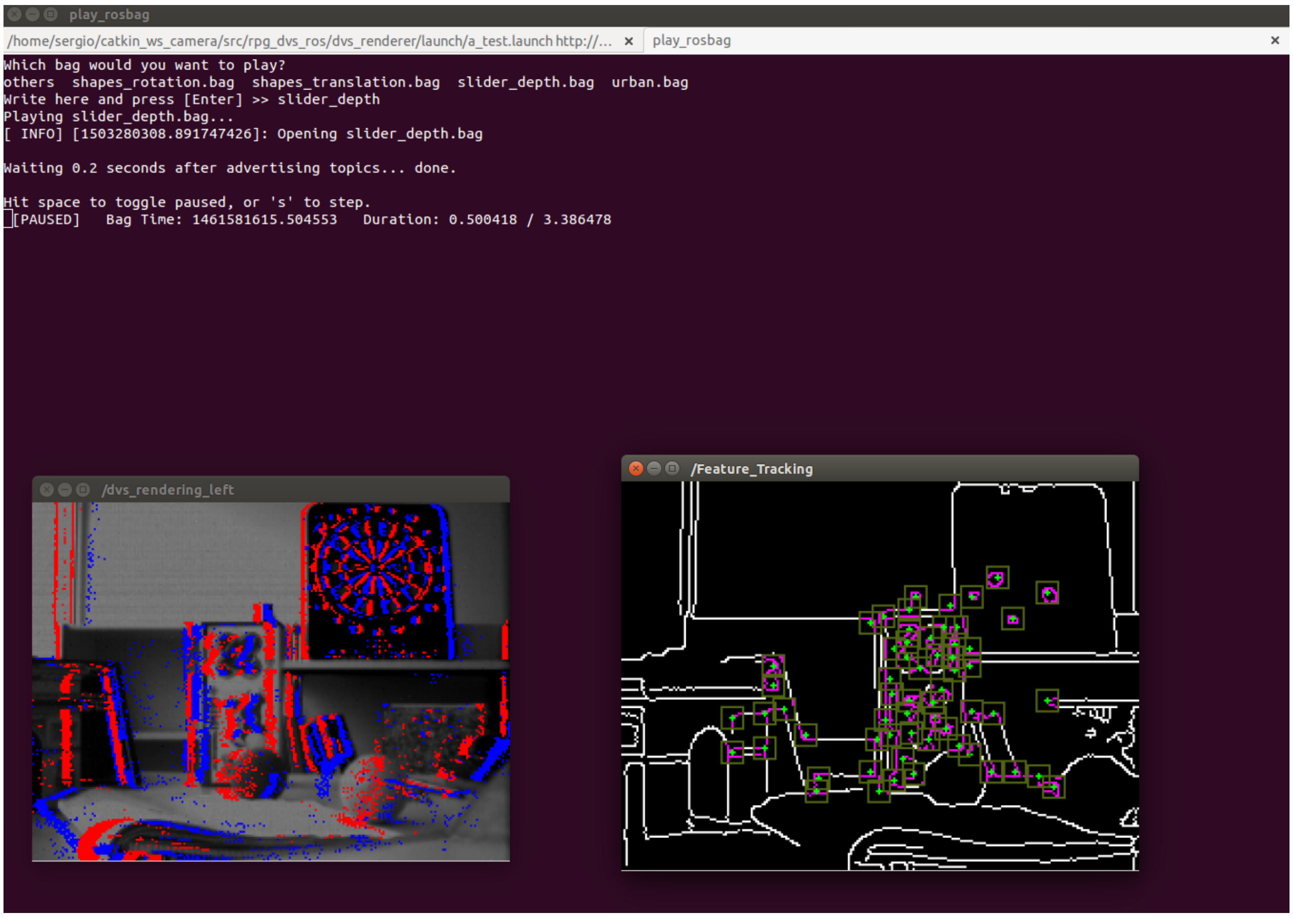
3.2 Launch File

A roslaunch serves to automate the nodes launching, as it was defined. This project has several nodes and package dependencies, thus it was needed to create a launch file to run the main node and all its dependent programs. It it possible to see the XML code and the ROS Qt-based graph visualizer of this file in section *Annexes*.

3.3 Bash File

Even if the roslaunch can make the process of launching nodes more quick, it is still needed to make several commands in many terminals to launch the program, what can really slow the work progress over time, principally while debugging. Thinking on this "limitation", a bash file was created with all the needed commands in order to run the entire program with only one executable file. The aftermath of this bash file is demonstrated in figure 3.2.

[4]Point Cloud Library.
[5]`https://github.com/uzh-rpg/rpg_dvs_ros`


play_rosbag
/home/sergio/catkin_ws_camera/src/rpg_dvs_ros/dvs_renderer/launch/a_test.launch http://...
play_rosbag
/dvs_rendering_left
/Feature_Tracking

```
Which bag would you want to play?
others  shapes_rotation.bag  shapes_translation.bag  slider_depth.bag  urban.bag
Write here and press [Enter] >> slider_depth
Playing slider_depth.bag...
[ INFO] [1503280308.891747426]: Opening slider_depth.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [PAUSED]   Bag Time: 1461581615.504553   Duration: 0.500418 / 3.386478
```

Figure 3.2 – The output of the bash file. It shows the available rosbags in the folder and asks the user which one should be played.

Chapter 4

Feature Detection

Due to the nature of how events are generated by the brightness changes, only taking a certain number of events over the time, making edges, are informative. Then, the intersection of these edges creates corners, which are points of interest, or "features". This traditional methodology has been proven optimally trackable in frame-based applications. As it is possible to obtain corners from event-based cameras, the main idea is to use the grayscale image to detect the features and then use events to track them (as shown in figure 4.1), keeping the low-latency property of the sensor.

Therefore, as the feature detection only uses the frame-based output, conventional techniques were used and will be described in this chapter. Also, as the interest of the grayscale image, in this context, is just used to find strong corners, frames are not required to be send in a constant rate.

4.1 Motivation

Feature detection and tracking with usual cameras have been hardly exploited in robotics and computer vision to locate and follow objects and estimate their speed, as well as the camera's position. Therefore several methods were developed to meet the different specifications and uses.

After the detection of strong corners in the scene, a patch around them is extracted. This is called feature descriptor, and in the classical approach they are followed using the incoming images, by detecting them in these new frames and calculating their new position. This method uses considerable amounts of image processing and it is one of the reasons of why this method does not fit in high-speed robotics.

Therefore, new methods and sensors, such as DVS, need to be developed in order to overcome this problem. The next section will introduce the work done, explaining the first steps of the feature detection process and showing some results using the provided DAVIS dataset by UZH.

4.2 Approach and Methodology

The global idea for feature detection and tracking realized in this work follows the schema in figure 4.2. The detailed process to detect features is illustrated in algorithm 1.

Algorithm 1 Grayscale frame-based corner detection

```
Feature Detection:
- Run Canny edge detector on the grayscale image (returns a binary mask: TRUE if there is an edge, FALSE otherwise).
- Save the edge mask.
- Apply Harris corner detector on the edge frame to find features.
- Obtain the square edge-map patches in the edge frame using the features position.
- Convert them to model point sets.
- Count how many $N$ edge pixels are inside in each one of these regions.
```

As demonstrated in the algorithm, first the received frame passes through a Canny's edge detection [3] and then the features are detected by a Harris corner detection [7] function. Different from [2, 8], where the corner

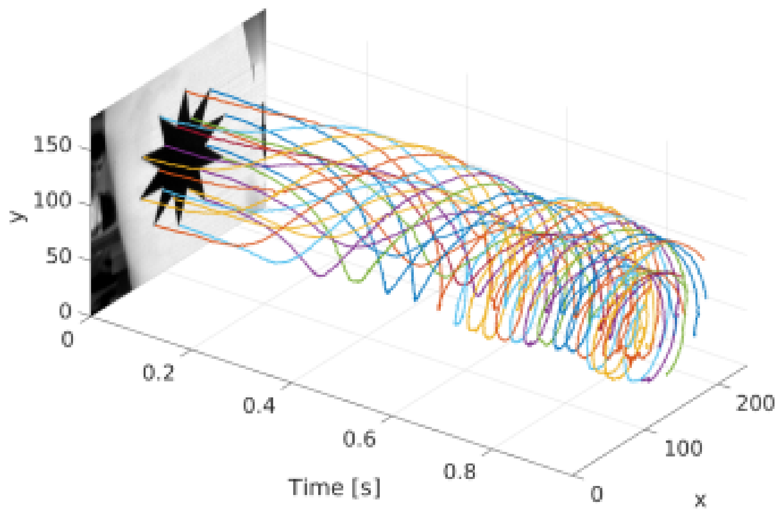


Figure 4.1 – An example of a feature tracking, where the features are detected first in the grayscale image. Figure extracted from [14]

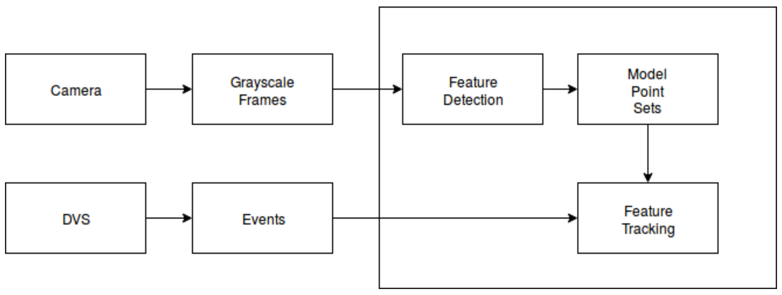


Figure 4.2 – Diagram model of the feature tracking using a DAVIS.

detection comes first. This change is due to the fact that the Harris method was detecting noisy points as features in some datasets.

Then, regions of interest are created around the features, which are called Model Point Sets. Each region has a determined number of white pixels corresponding to the edge. The idea is to count them and find the same N value of events to the feature tracking section.

The incoming grayscale image is saved in the algorithm, then the OpenCV[1] function *blur* smooths the image and the *Canny* applies the edge detector.

4.2.1 Canny Edge Detector

It is an operator developed by John F. Canny [3] to estimate a wide range of edges in the image. This method is composed by five steps detailed below:

Gaussian filter: It is needed to filter the image in order to remove the remaining noise that can affect the edge detection. The equation to a Gaussian filter of size $(2k + 1) \times (2k + 1)$ is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp\left(\frac{(i - (k+1)^2 + (j - (k+1))^2}{2\sigma^2}\right)$$

Where i and j are the image lines and columns respectively.

Differentiation: Four filters are used to detect horizontal, vertical and diagonal edges in the blurred image. The horizontal x image array are convolved with the first derivative of the dimensional Gaussian (of same σ

[1] *C++* library largely used in this work

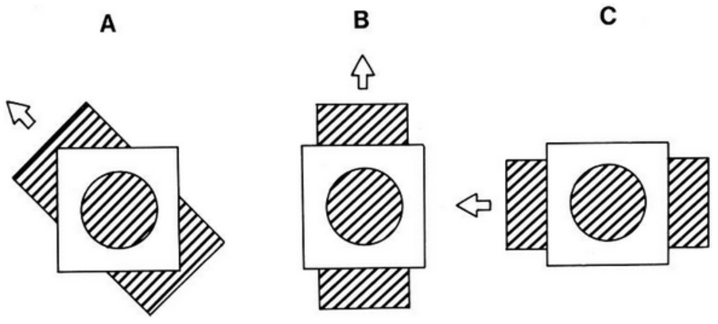


Figure 4.3 – The aperture problem, where is not possible to determine the motion direction.

than the previous stage) aligned with y. The same occurs to the vertical y array. Therefore, it is possible to compute the magnitude and angle of the slope using the hypotenuse and the arctangent[2].

Non-maximum suppression: Each pixel is among a 8-pixels neighborhood. By interpolating the surrounding discrete grid values, there is possible at the neighborhood boarder to compute the gradient magnitudes in both horizontal and vertical directions. Then, if a pixel is not greater than these two values, it is erased.

Edge thresholds: Even if the current edges are more accurate, there is still noisy edge pixels that need to be filtered to preserve the pixels with a high gradient value. So a high and a low thresholds are placed. If a pixel's gradient is bigger than the high threshold, it is marked as strong edge pixel, while those who are smaller the low threshold are eliminated. Finally, if a pixel is between these thresholds, it is marked as a weak edge's pixel.

Edge tracking by hysteresis: Usually, a weak edge's pixel belonging to a true edge is connected to a strong pixel, and those coming from noisy are unconnected. Thus, to find this edge connection a "blob analysis" is applied to the neighborhood of the weak edge's pixels in order to find a strong one. If there is none, the analyzed weak pixel is removed.

The obtained edges from the image source are saved in an image I. The next step is to find the features applying the Harris corner detector method in this image.

4.2.2 Harris Corner Detector

Features are not simply corners obtained from the intersection of edges, actually, to find a good interesting points, it need to do not suffer from the aperture problem, which is shown in figure 4.3. If the object is moving in a diagonal, moving up or left, the resultant movement seen throw the aperture is the same.

The method proposed by C. Harris and M. Stephens [7] overcomes the mentioned aperture problem, finding strong features to track. This is an improvement of the Moravec's corner detector.

First, an image patch over the area (u, v) of the image I is moved by an offset (x, y). The sum of squared differences between these regions (and also weighed by $w(u, v)$, that is the type of window slided over the image) is given by S:

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I(u+x,v+y) - I(u,v))^2$$

Using Taylor expansion:

$$S(x,y) = \sum_u \sum_v w(u,v)(I_x(u,v)I_y(u,v)y)^2$$

Where I_x and I_y are the partial derivatives of I. Then it can be written in the matrix form:

$$S(x,y) = \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$$

[2]It is similar to Sobel operator.

A is the tensor:

$$A = \sum_u \sum_v w(u,v) \begin{pmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{pmatrix}$$

This is the Harris matrix. A feature is determined by a large variation of S in all direction of the vector $\begin{pmatrix} x & y \end{pmatrix}$. To determine this characterized the eigenvalues of A are analyzed. Normally a good corner has two large eigenvalues, then the follow information can be inferred: If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then this pixel has no points of interest. If only one of the eigenvalues is positive and large, then it means that it is an edge. Finally, if both eigenvalues have large positive values, then it is a corner.

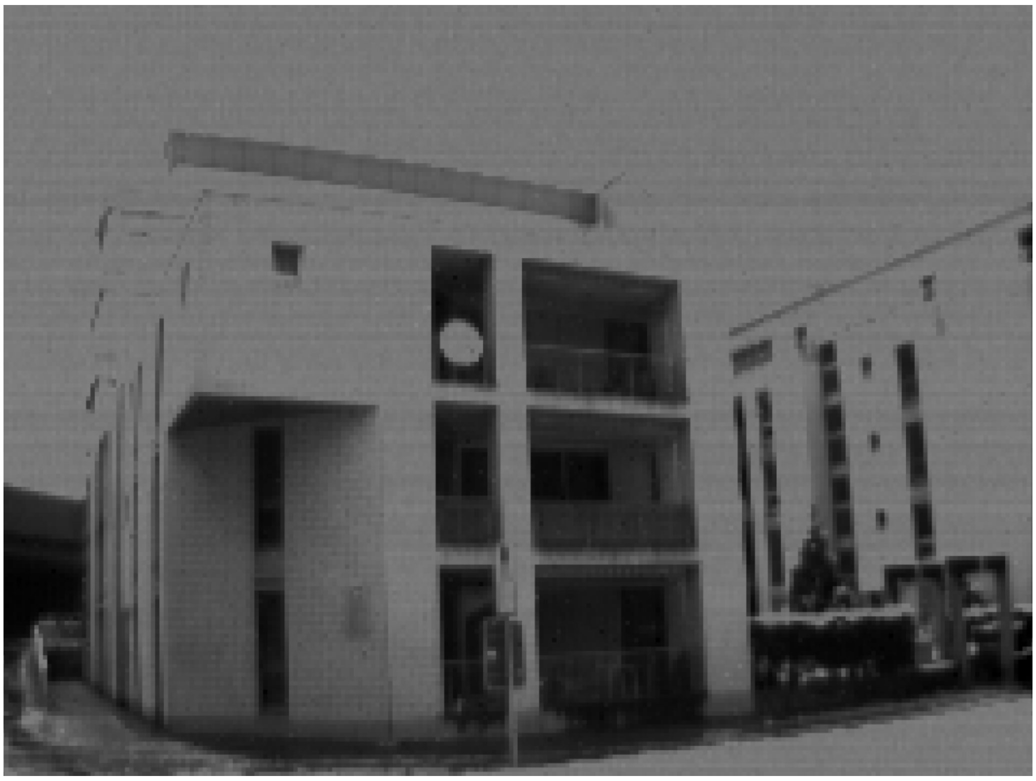
This detector is applied in the image I using the OpenCV function *goodFeaturesToTrack*, which one of its parameters is a boolean to determine the use or not of Harris detector method. The result of these two process is shown in figure 4.4b, where the green crosses are the features and the white edges were detected by the Canny's method.

4.2.3 Model Point Sets

The pixels of the found edge-map constitute of binary masks that show the presence or absence of an edge. These masks, when next from a feature, determine interest shapes to be tracked. Therefore, patches in this image are defined indicating the likely dominant source of events and they are called "model point sets".

In this work, square patches of the same 11×11 pixel size were used to determine the model point sets, but other sizes and shapes can be used.

In figure 4.5, the purple pixels define the edges next to a feature, it means, inside a model point set (green square). These pixels defines the shapes that are interesting to track in the scene. Therefore, the next step is the feature tracking itself.


(a) Grayscale image (obtained by *urban.bag*).

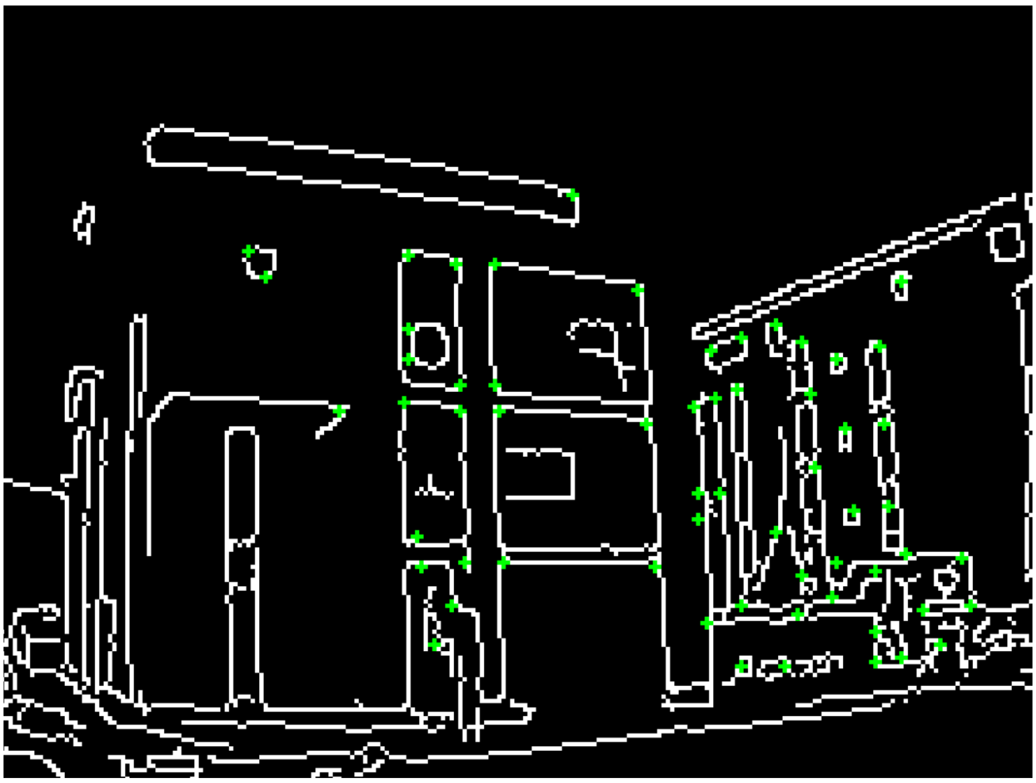
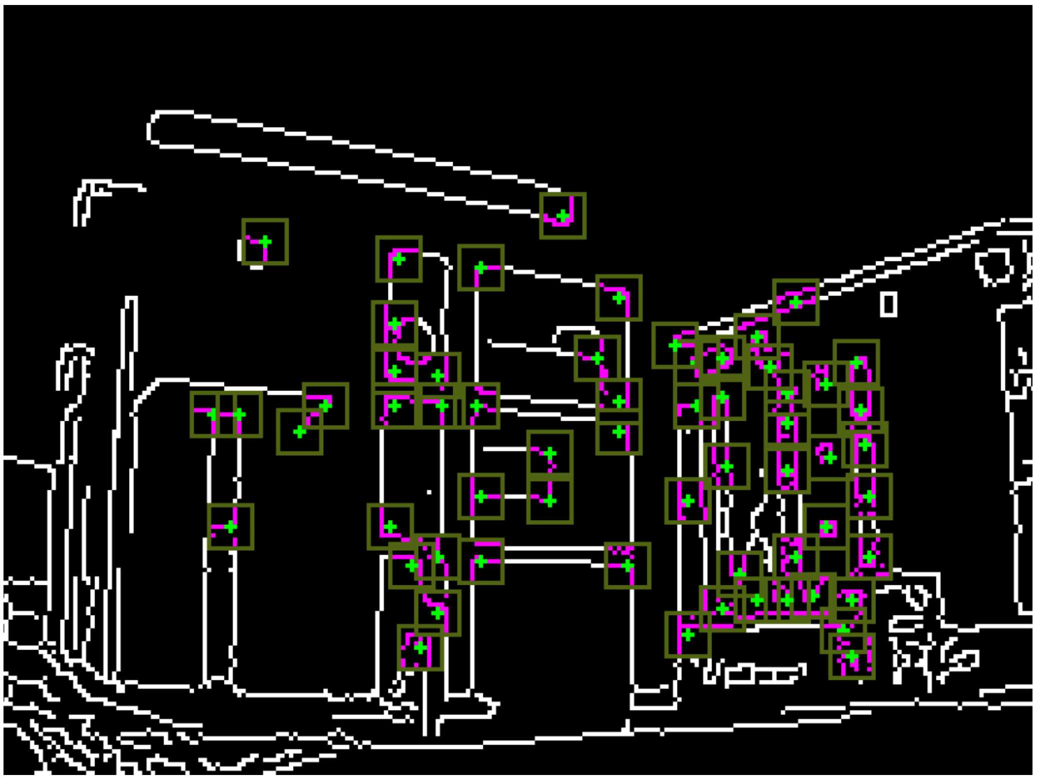
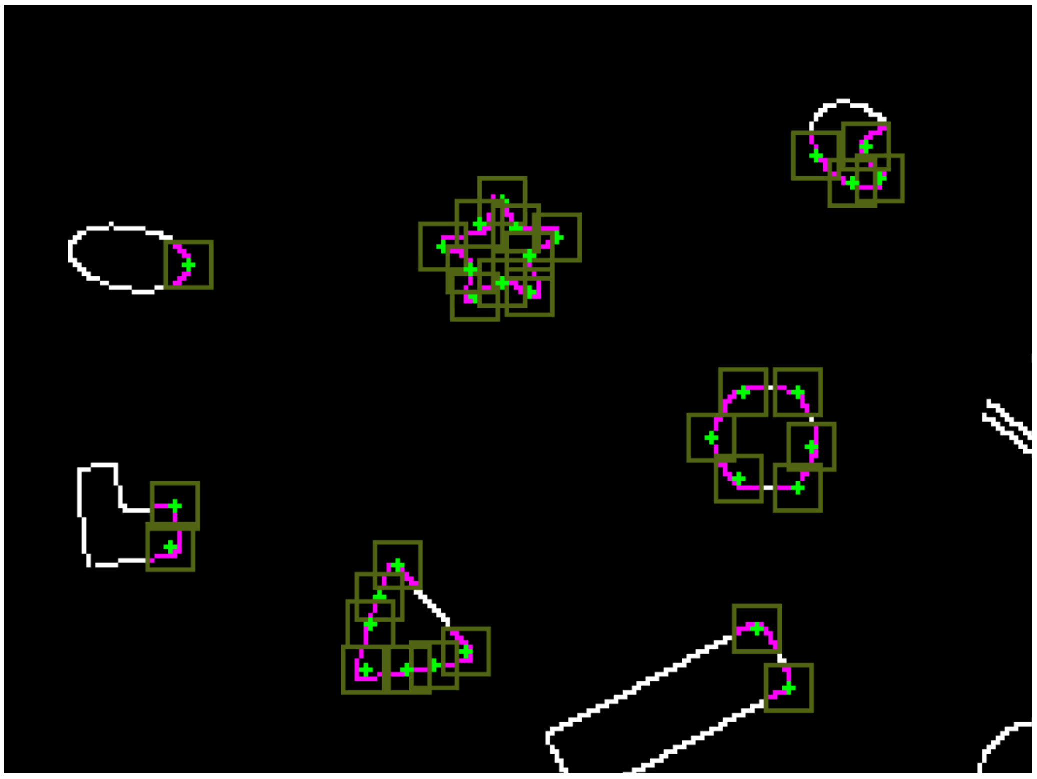

(b) The resultant image after edge (white borders) and corner (green crosses) detection.

Figure 4.4 – The process of feature detection from a given grayscale frame.



(a)



(b)

Figure 4.5 – The resultant model point sets detected in (a) *urban.bag* and (b) *shapes_rotation.bag*. The purple pixels means that this is an edge near to a feature.

Chapter 5

Feature Tracking

Conventional feature tracking methods are limited by the frame rate and shutter speed of the frame-based camera. It means that if a detected feature moves around the scene more quickly than the frequency of coming frames, it would not be tracked and very probably will be detected as a "new" feature. As the interest of the present work is to do a low-latency feature tracking, once the features are obtained, the interest is to track them using the events, which in DAVIS they have a precisely 1 μs rate. This process is illustrated in figure 5.1 and the method used in this chapter follows the algorithm 2.

Algorithm 2 High temporal resolution feature tracking

```
Feature Tracking:
- Define a data point set for each model point set
- Store, for each one of them, the latest $N$ events.
for each incoming event do
    if This event belongs to a data point set then
        - Update the data point set.
        for the updated data point set do
            - Calculate the registration parameters between the data and the model point sets.
            - Update the feature position and its corresponding model point set.
```

5.1 Data Point Sets

As input, the program will get the local and multiple model point sets defined in the previous chapter by the features positions. Then, the idea is not to track only the features, but these entire regions using the events. It is possible by accumulating a certain number of events that happens inside each model point sets, what will be called data point set. This is due to the fact than over time, if there is movement in the scene the accumulated events will "draw" a contour of similar shape to edges.

This data point sets have the same number of events than pixels inside the model point sets, therefore, this is why it was needed to count them in the feature detection section. Each incoming event will be inserted in a certain patch only if its coordinates match to its inside space. Then, for example, if a certain data point set admits until N events, the $(N+1)$st arriving event will replace the oldest event on the set[1]. This process allows to have and use always the newest events stored into these subsets.

Therefore, the data point sets determine the relevant events to be used in the registration of the feature. This procedure is realized by a minimization between the data and the model point sets, but it is better explained in the next section. However, before proceeding, both point sets need to be transformed in the PCL's *PointCloud<pcl::PointXYZRGB>* format in order to use the ICP[2] registration method from this library. A point cloud is basically a set of data points, usually in the 3D coordinate system.

Once both sets were converted in the above described point cloud sets, the outcome are two matrices where the lines represent the feature's number and the columns, all the positions for its pixels (for the model point cloud) or its events (for the data point cloud) in a 3D point format. But, if the registration method is applied

[1]FIFO system.

[2]Iterative Closest Point

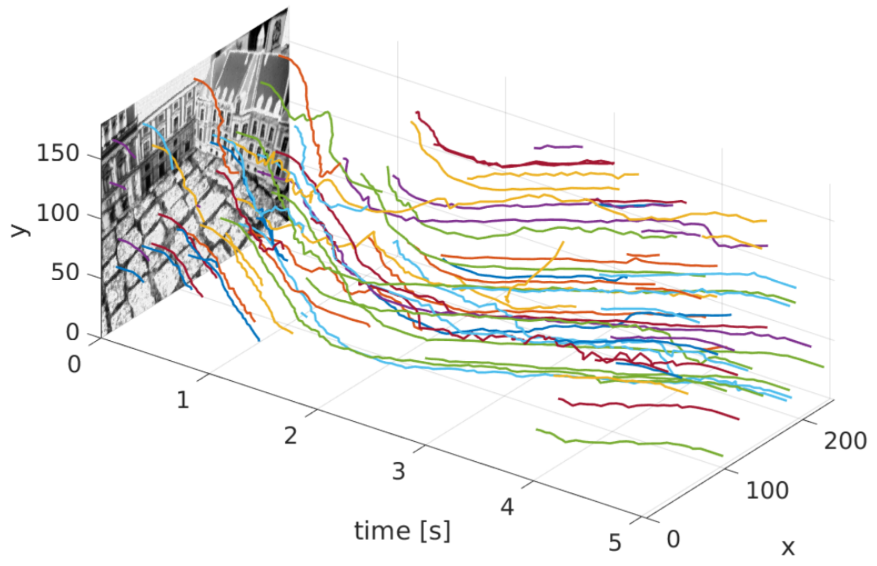


Figure 5.1 – Features tracking using only the events. This example was extracted from [8].

directly in these matrices it means that the scene is considered as only one whole object, what is not desired. To keep each set of points independently from each other, two final point cloud arrays are created. They are auxiliary arrays that, within each iteration will receive the ith line of each point cloud, what corresponds to the ith feature being analyzed. A pass through filter is used on them to remove NaN data. Then, the registration algorithm is applied and the ith feature has its position value updated.

The output of this section is shown by figure 5.2. For each patch there are the same amount of purple edge pixels and yellow pixels, which represent the last arrived events. Figure 5.2a processes a sequence of images recorded in an outdoor scenario by a person who was walking, therefore there is much more noisy when compared to figure 5.2b, which was also recorded by hands, but the video presents a simple translate movements over well defined shapes.

5.2 Registration

The problem of align 3D point cloud data in a model its called registration. Usually, the objective is to determine the relative positions and orientations of the same object (viewed in different angles) in a global coordinate system framework to perfectly match it. In the case of the present feature tracking problem, the interest is to match the edge's shape within a model point set to the near events happening, which should have a similar form in theory.

There are several registration methods such as feature based registration, which uses the SIFT Keypoints and FPFH descriptors. Following the idea of [8], the registration method chosen was a weighed ICP, explained below, which is based in correspondences estimation.

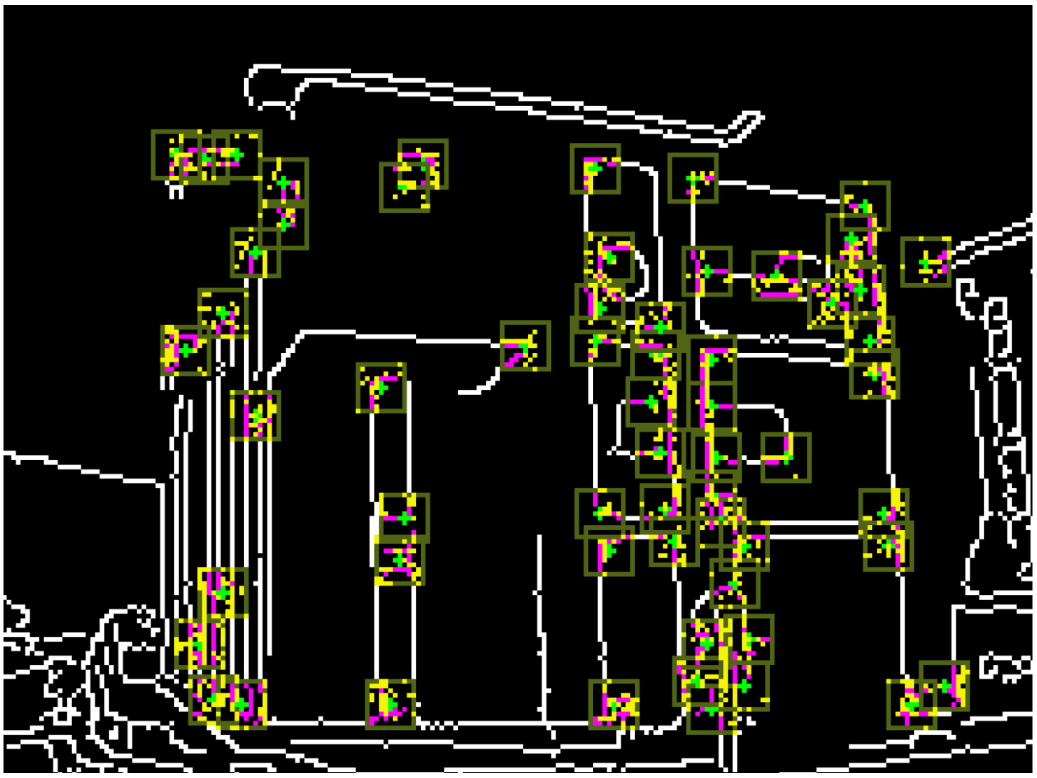
5.2.1 Iterative Closest Point

The data point set e_i is registered to its correspondent model point set p_i by minimizing the weighted distance between them:

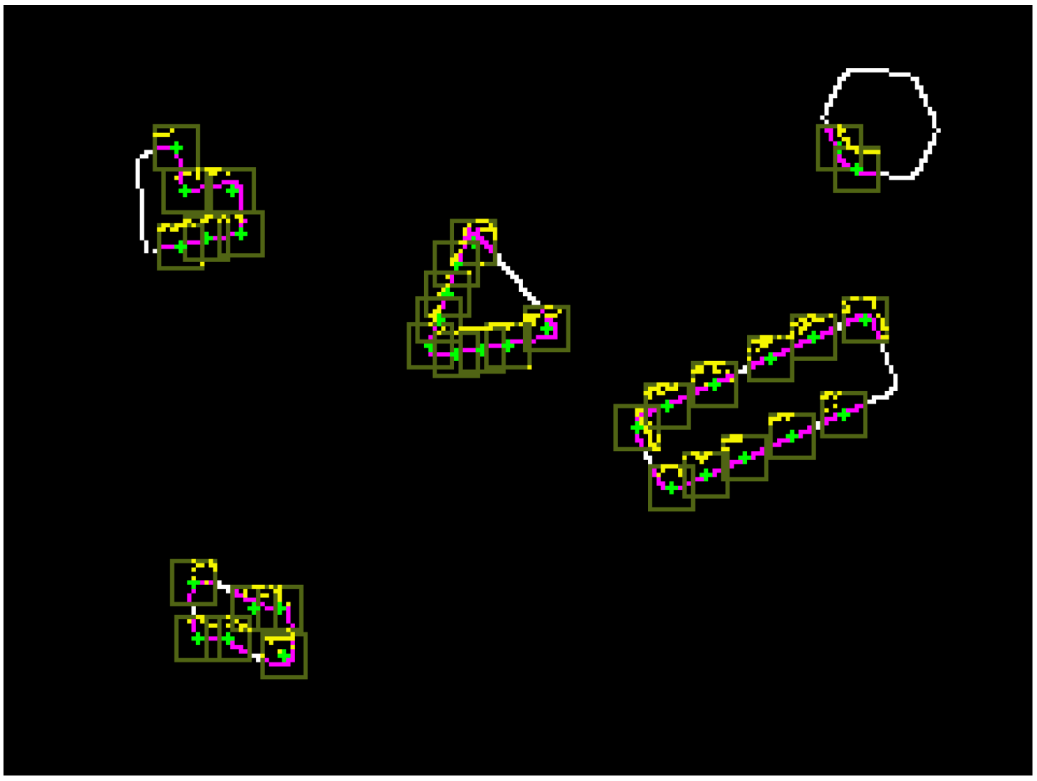
$$\arg\min_{R,t} = \sum_{(e_i,p_i)\in matches} b_i \|R(e_i) + t - p_i\|^2$$

Where an Euclidean transformation (R, t) is assumed. The weights b_i consider the outlier rejection and they are proportional to the quantity of events that are in the 3×3 neighborhood of the analyzed event. The chosen Iterative Closest Point algorithm [1] minimizes the given distance.

The iterative algorithm is composed by three main stages:



(a)



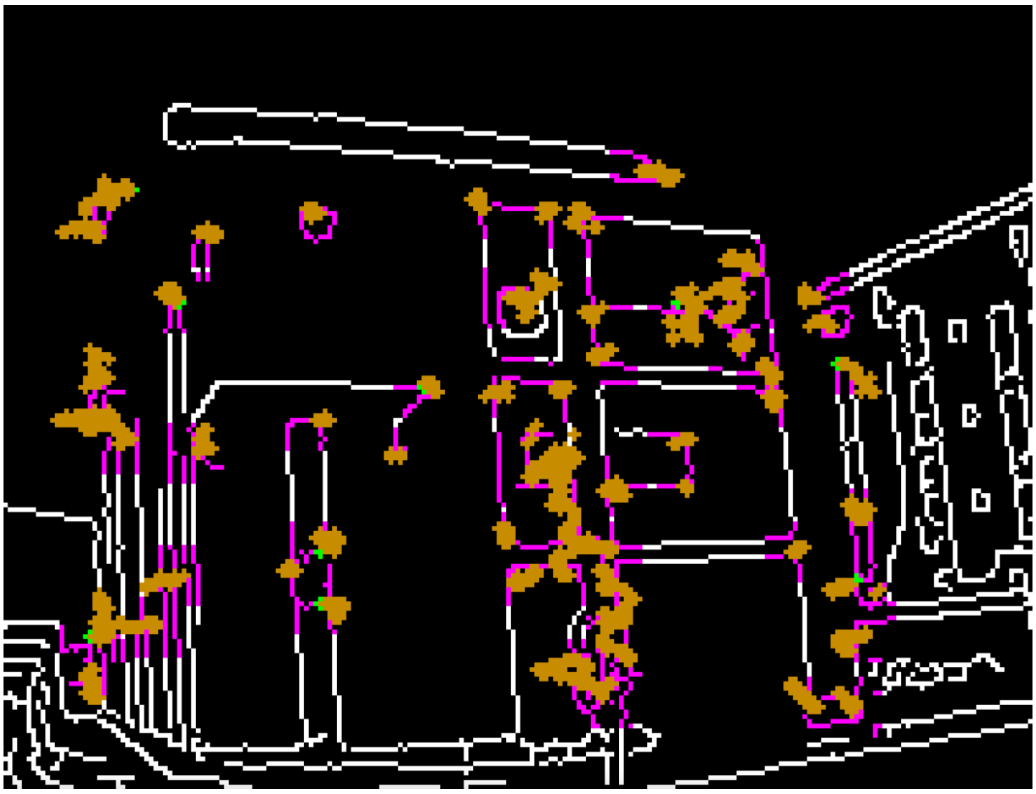
(b)

Figure 5.2 – The green squares are the data point sets, containing the same number of events, in yellow, and pixels, in purple.

- The two candidate (data and model point sets) are established.
- The transformation matrix is computed.
- The found transformation is applied in the feature position, updating its whole model point set.

As the DAVIS provides a very high temporal resolution, the computation after each consecutive event in a certain feature becomes unnecessary, giving a transformation matrix approximatively equal to identity. Therefore, in this work the ICP algorithm was computed after each 25 new incoming events per feature.

In figure 5.3 it is possible to see the final outcome. The set of orange crosses represent the updating position of each feature over the time. Figure 5.3a shows the result of the algorithm in an outside scene, where some features do not estimate well their positions because of the noise produced by the events. In figure 5.3b, the same problem happens, however as it was recorded by a motorized linear slider, the errors originated from hand shaking do not exist, thus the feature tracking is more precise.


(a)

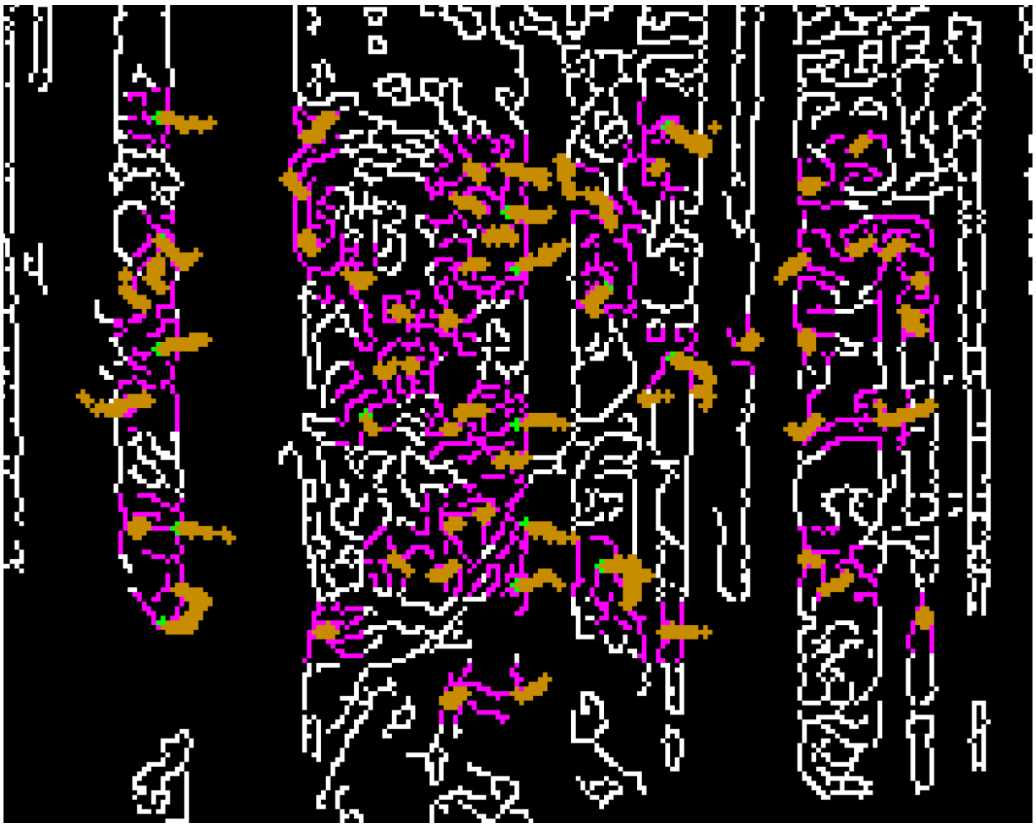

(b)

Figure 5.3 – The orange crosses represent the updated features position of the old features in green.

Chapter 6

Experiments

In this chapter, it will be presented the idea of the tests to validate the method and to observe its application in different ambiances, for example near and far objects. Knowing that some used datasets have dominant translations over the recorded data and others, rotations, it is possible to make use of these characteristics to study the rotational and translational estimation of the program.

As mentioned in chapter 2, the dataset provided by UZH has no IMU values because they are too noisy. However, some of the recorded files have a ground truth that gives the positions over time and the orientation in quaternions unit $q = (q_x, q_y, q_z, q_w)^\top$, where $q_x, q_y, q_z, q_w \in \mathbb{R}$. These four elements represent the rotation such that $q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1$.

For simplicity and as there is no need to present the quaternion algebra here, the fundamental information to know in order to use this data is that the rotation matrix R associated to a rotation quaternion is defined by:

$$R = \begin{pmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_x) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_z) \\ 2(q_x q_z - q_w q_z) & 2(q_y q_z + q_w q_y) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix}$$

Therefore, the main idea is to detect a feature and track it only using events along the scene and compute its positions against the ground truth data to find the tracking error. And then, obtain this error in pixels per time as shown in figure 6.1, where the black line is the mean feature tracking error and the blue space is the confidence interval.

Even if natural scenes do not have a provided ground truth, it is possible to see in the last chapter that the algorithm still tracks the strongest features (figure 5.3a). As Canny's method detects hight changes in the image contrast, in outdoors scenes it is more difficult to obtain precise edges once there is several different grayscale levels in magnitude. This is also shown in figure 5.3b, where the best features tracked come from the boarders, reflecting in a well defined contour.

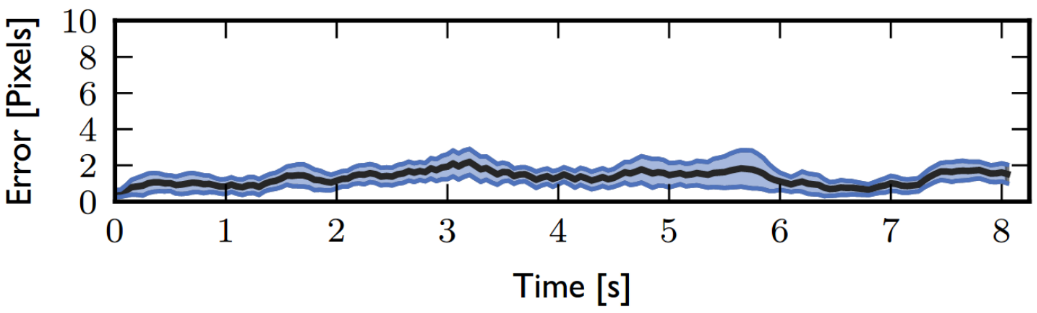


Figure 6.1 – The black line represents the mean feature tracking error. The blue space surrounding the mean indicates the confidence interval. This approach example was extracted from [8].

Chapter 7

Conclusion

The new characteristics brought by the neuromorphic image sensor DAVIS, which combines a frame-based grayscale images with events in the same array of pixel, have proven be useful in autonomous cars applications for feature tracking, an important step for visual odometry. The method consists in using a single image to detect corners as conventional frame-based techniques do and then follow them using only the brightness changes in the scene. The low-latency property of the system meets the requirements of high speed robots, allowing to track objects even in the blind time between the frames of classical cameras with a micro-second precision. Moreover, the redundancy data coming from static scenes are completely eliminated, what saves the computational cost.

Although there are issues in tracking some features due to event noise, the algorithm still matches and follows the strongest features in the scene. Well defined borders as the end of a wall are less susceptible to this noise, what is expected because the high intensity change in the luminosity. This reflects one of the limitations of event-based cameras: their relatively low resolution, which affects both feature detection and tracking.

Despite some improvements that need to be done in order to attain a more accurate and robust tracking, the work done serves as base to a future visual odometry study using event-based cameras in a ROS environment. Where a data fusion can be done with other sensors for autonomous vehicle localization. Some ideas are suggested below.

Hints for Future Work: A current problem for the feature tracking algorithm is the high noise produced in the DVS, giving several "fake" movements that can be computed in the ICP during registration. In order to filter these pixels, a similar approach to the last step of Canny's method presented in chapter 4 can be applied: edge tracking by hysteresis. As the idea of the methodology is to make the edges follow the events, a filter can be implemented in order to check if within each event neighborhood there are other events, and in positive case, if they are "strong" events, filtering then the "weak" events.

Another improvement idea is given in [8], it consists in create two histograms per feature of the same area than the data point sets, but storing more events over time. One retains the first N events and the other, the last N events. Then, use them to filter noisy events, achieving a higher robustness to the system.

Bibliography

[1] P. J. Besl and N. D. McKay, *A method for registration of 3-d shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (1992), pp. 239–256.

[2] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, *A 240x180 130db 3us latency global shutter spatiotemporal vision sensor*, IEEE Journal of Solid-State Circuits, 49 (2014), pp. 2333–2341.

[3] J. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8 (1986), pp. 679–698.

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

[5] T. Delbrück and P. Lichtsteiner, *Fast sensory motor control based on event-based hybrid neuromorphic-procedural system*, in ISCAS, IEEE, 2007, pp. 845–848.

[6] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch, *Activity-driven, event-based vision sensors*, IEEE, May 2010, pp. 2426–2429.

[7] C. Harris and M. J. Stephens, *A combined corner and edge detector*, in Alvey Conference, 1988, pp. 147–152.

[8] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, *Low-latency visual odometry using event-based feature tracks*, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2016, pp. 16–23.

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

[10] E. Mueggler, H. Rebecq, G. Gallego, T. Delbrück, and D. Scaramuzza, *The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM*, CoRR, abs/1610.08336 (2016).

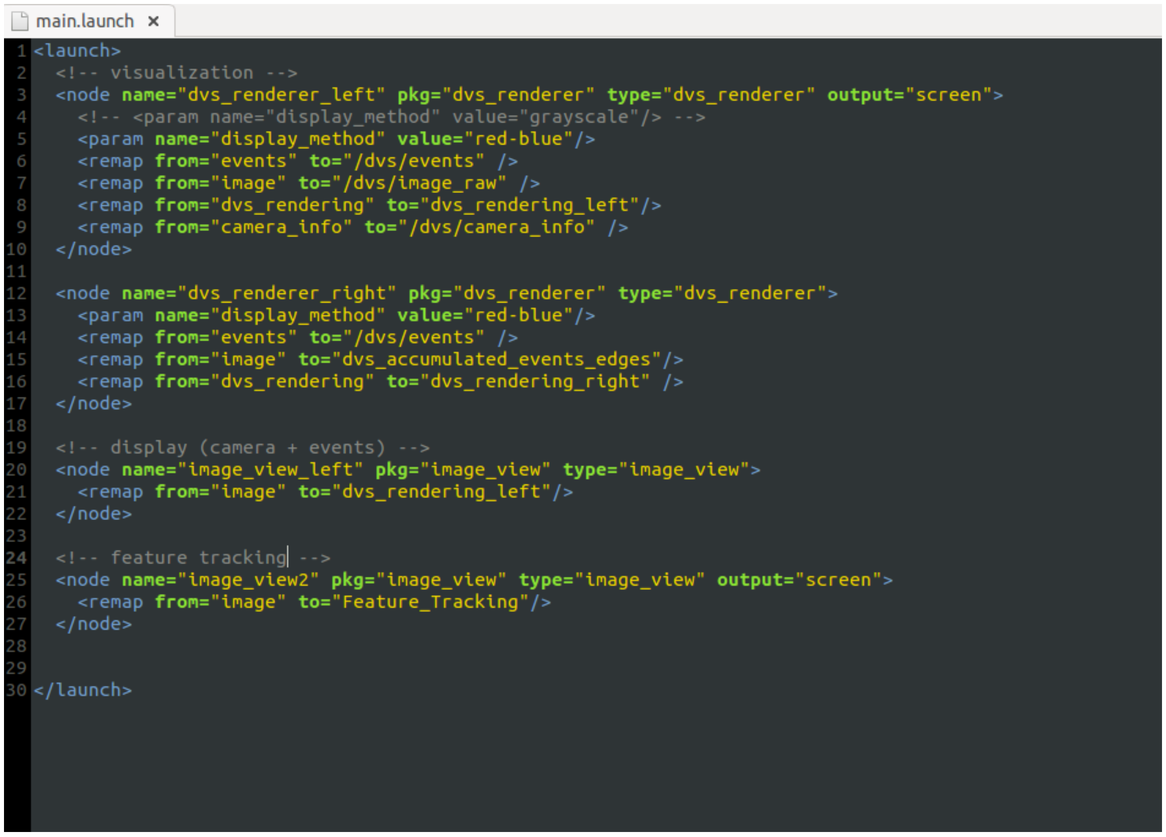
[11] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Régnier, *Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics*, IEEE Trans. Robotics, 28 (2012), pp. 1081–1089.

[12] C. Posch, D. Matolin, and R. Wohlgenannt, *A qvga 143db dynamic range asynchronous address-event pwm dynamic image sensor with lossless pixel-level video compression*, in 2010 IEEE International Solid-State Circuits Conference - (ISSCC), Feb 2010, pp. 400–401.

[13] C. Posch, D. Matolin, and R. Wohlgenannt, *An asynchronous time-based image sensor*, in ISCAS, IEEE, May 2008, pp. 2130–2133.

[14] D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza, *Feature detection and tracking with the dynamic and active-pixel vision sensor (davis)*, in 2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP), June 2016, pp. 1–7.

ANNEXES



```
main.launch ×
<launch>
  <!-- visualization -->
  <node name="dvs_renderer_left" pkg="dvs_renderer" type="dvs_renderer" output="screen">
    <!-- <param name="display_method" value="grayscale"/> -->
    <param name="display_method" value="red-blue"/>
    <remap from="events" to="/dvs/events" />
    <remap from="image" to="/dvs/image_raw" />
    <remap from="dvs_rendering" to="dvs_rendering_left"/>
    <remap from="camera_info" to="/dvs/camera_info" />
  </node>

  <node name="dvs_renderer_right" pkg="dvs_renderer" type="dvs_renderer">
    <param name="display_method" value="red-blue"/>
    <remap from="events" to="/dvs/events" />
    <remap from="image" to="dvs_accumulated_events_edges"/>
    <remap from="dvs_rendering" to="dvs_rendering_right" />
  </node>

  <!-- display (camera + events) -->
  <node name="image_view_left" pkg="image_view" type="image_view">
    <remap from="image" to="dvs_rendering_left"/>
  </node>

  <!-- feature tracking -->
  <node name="image_view2" pkg="image_view" type="image_view" output="screen">
    <remap from="image" to="Feature_Tracking"/>
  </node>


</launch>
```

Figure 7.1 – The XML main.launch file to automate the ROS nodes launching.

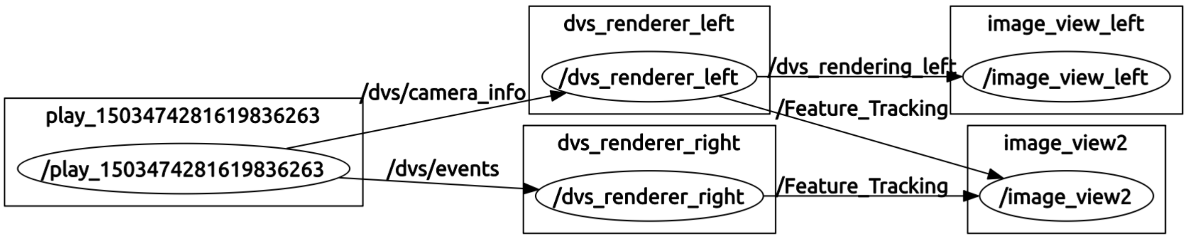


Figure 7.2 – ROS Qt-based graph to visualize the state of the system.