NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

JOÃO PEDRO CARVALHO PAULO

Master/BSc in Electrical and Computer Engineering

COLLISION DETECTION FOR UAVS USING EVENT CAMERAS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

March, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COLLISION DETECTION FOR UAVS USING EVENT CAMERAS

JOÃO PEDRO CARVALHO PAULO

Master/BSc in Electrical and Computer Engineering

Advisers: André Mora
Assistant Professor, NOVA University Lisbon
Tomasz Kryjak
AGH University of Science and Technology

Co-adviser: Krzysztof Blachut
AGH University of Science and Technology

Examination Committee

Chair: Luis Brito Palma
Assistant Professor, FCT-NOVA

Rapporteur: José Manuel Ribeiro da Fonseca
Associate Professor with Habilitation, FCT-NOVA

Adviser: André Mora
Assistant Professor, FCT-NOVA

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2023

Collision detection for UAVs using Event Cameras

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.10) [31].

To my family and friends...

Acknowledgements

I am deeply grateful to Tomasz Kryjak and the Department of Automatic Control and Robotics from the AGH University of Science and Technology, for providing me with the opportunity to work with them. Professor Tomasz's support and expertise have been invaluable to my research.

I would also like to extend my heartfelt thanks to Krzysztof Blachut, who played a crucial role in the success of my dissertation. His extensive knowledge, insightful feedback, and unwavering support have been instrumental in shaping my research and academic growth. I am deeply grateful for his mentorship, encouragement, and guidance throughout this journey.

Additionally, I would like to thank Professor André Mora and the NOVA School of Science and Technology for their support and encouragement at the beginning of this process. Their assistance and belief in me have been an essential source of inspiration and motivation.

Finally, I would like to express my appreciation to my family and friends for their enduring support throughout my academic journey. Their love, encouragement, and belief in me have been my greatest source of strength, and I am forever grateful for their presence in my life.

"I believe that each person constitutes a unique pattern of potentialities, and these potentialities can only be developed if this person is educated in the sense of stimulating his or her creativity."
(Unknown)

Abstract

This dissertation explores the use of event cameras for collision detection in unmanned aerial vehicles (UAVs). Traditional cameras have been widely used in UAVs for obstacle avoidance and navigation, but they suffer from high latency and low dynamic range. Event cameras, on the other hand, capture only the changes in the scene and can operate at high speeds with low latency. The goal of this research is to investigate the potential of event cameras in UAVs collision detection, which is crucial for safe operation in complex and dynamic environments.

The dissertation presents a review of the current state of the art in the field and evaluates a developed algorithm for event-based collision detection for UAVs. The performance of the algorithm was tested through practical experiments in which 9 sequences of events were recorded using an event camera, depicting different scenarios with stationary and moving objects as obstacles. Simultaneously, inertial measurement unit (IMU) data was collected to provide additional information about the UAV's movement. The recorded data was then processed using the proposed event-based collision detection algorithm for UAVs, which consists of four components: ego-motion compensation, normalized mean timestamp, morphological operations, and clustering.

Firstly, the ego-motion component compensates for the UAV's motion by estimating its rotational movement using the IMU data. Next, the normalized mean timestamp component calculates the mean timestamp of each event and normalizes it, helping to reduce the noise in the event data and improving the accuracy of collision detection. The morphological operations component applies mathematical operations such as erosion and dilation to the event data to remove small noise and enhance the edges of objects. Finally, the last component uses a clustering method called DBSCAN to group the events, allowing for the detection of objects and estimation of their positions. This step provides the final output of the collision detection algorithm, which can be used for obstacle avoidance and navigation in UAVs. The algorithm was evaluated based on its accuracy, latency, and computational efficiency.

The findings demonstrate that event-based collision detection has the potential to be an effective and efficient method for detecting collisions in UAVs, with high accuracy and

low latency. These results suggest that event cameras could be beneficial for enhancing the safety and dependability of UAVs in challenging situations.

Moreover, the datasets and algorithm developed in this research are made publicly available, facilitating the evaluation and enhancement of the algorithm for specific applications. This approach could encourage collaboration among researchers and enable further comparisons and investigations.

Keywords: Event-cameras, UAVs, Collision detection algorithm, Ego-motion, IMU, Dynamic objects

Resumo

Esta dissertação explora o uso de câmeras de eventos para deteção de colisões em veículos aéreos não tripulados (UAVs). As câmeras tradicionais têm sido amplamente utilizadas em UAVs para evitar obstáculos, mas sofrem de alguns problemas como alta latência ou baixa faixa dinâmica. As câmeras de eventos, por outro lado, capturam apenas as alterações na cena e podem operar em alta velocidade com baixa latência. O objetivo desta pesquisa é investigar o potencial de câmeras de eventos na deteção de colisões em UAVs, o que é crucial para uma operação segura em ambientes complexos e dinâmicos.

A dissertação apresenta uma revisão do estado atual da arte neste tema e avalia um algoritmo desenvolvido para deteção de colisões em UAVs baseado em eventos. O desempenho do algoritmo foi avaliado através de testes práticas em que foram registadas 9 sequências de eventos utilizando uma câmera de eventos, retratando diferentes cenários com objetos estacionários e em movimento. Simultaneamente, foram capturados dados da unidade de medida inercial (IMU) para fornecer informações adicionais sobre o movimento do UAV. Os dados registados foram então processados usando o algoritmo proposto de deteção de colisões, que consiste em quatro etapas: ego-motion compensation, normalized mean timestamp, operações morfológicas e clustering.

Primeiramente, o ego-motion compensation compensa o movimento do UAV estimando o seu movimento rotacional usando os dados do IMU. Em seguida, o componente de normalized mean timestamp cálcula o timestamp médio de cada evento e normaliza-o, ajudando a reduzir o ruído nos dados de eventos e melhorando a precisão da deteção de colisões. A etapa de operações morfológicas aplica operações matemáticas como erosão e dilatação nos dados dos eventos para remover pequenos ruídos. Finalmente, a última etapa utiliza um método de clustering chamado DBSCAN para agrupar os eventos, permitindo a deteção de objetos e a estimativa das suas posições. Esta etapa fornece o output final do algoritmo de deteção de colisões, que pode ser usado para evitar obstáculos em UAVs. O algoritmo foi avaliado com base na sua precisão, latência e eficiência computacional.

Os resultados demonstram que a deteção de colisões baseada em eventos tem o potencial de ser um método eficaz e eficiente para a deteção de colisões em UAVs, com

alta precisão e baixa latência. Estes resultados sugerem que as câmeras de eventos poderiam ser benéficas para melhorar a segurança e a confiabilidade dos UAVs em situações desafiadoras.

Além disso, os conjuntos de dados e o algoritmo desenvolvido nesta pesquisa estão disponíveis online, facilitando a avaliação e o aprimoramento do algoritmo para aplicações específicas. Esta abordagem pode incentivar a colaboração entre os investigadores da área e possibilitar mais comparações e investigações.

Palavras-chave: Câmaras de eventos, UAVs, Algoritmo de deteção de colisões, Egomotion, IMU, Objectos dinâmicos

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence *(p. 5)*

CNN Convolutional Neural Network *(p. 26)*
CPU Central Processing Unit *(p. 69)*

DBSCAN Density-Based Spatial Clustering of Applications with Noise *(p. 54)*
DVS Dynamic Vision Sensor *(p. 31)*

EAS Events Aggregation and Superimposition *(p. 33)*

FOGs Fibre-Optic Gyroscopes *(p. 41)*
FPGA Field-Programmable Gate Array *(p. 68)*
fps frames per second *(pp. 5, 10)*

HDR High Dynamic Range *(p. 8)*
Hz Hertz *(pp. 5, 10)*

IMO Independently Moving Objects *(p. 26)*
IMU Inertial measurement unit *(p. 16)*
INS Inertial Navigation Systems *(p. 42)*
IWE Image of Warped Events *(p. 31)*

MEMS Micro Electro Mechanical System *(p. 41)*
MiB mebibytes *(p. 62)*

OPTICS Ordering Points To Identify the Clustering Structure *(p. 67)*

PD Proportional Derivative Controller *(p. 20)*

RLGs Ring Laser Gyroscopes *(p. 41)*
ROS Robot Operating System *(p. 41)*

UAV Unmanned Aerial Vehicle *(p. 1)*

1

Introduction

Collision detection for UAVs is a critical aspect of ensuring the safety and reliability of these systems. The use of UAVs has rapidly increased in recent years, with applications ranging from military operations to civilian search and rescue, agriculture, and even package delivery. However, as UAVs become more prevalent, the risk of collisions with other aircraft or objects in the environment increases. This is particularly concerning in crowded urban areas, where the potential for collision with other aircraft or obstacles is high.

One promising approach to addressing this problem is the use of event cameras, which have the ability to capture visual information at extremely high frame rates, making them ideal for detecting and tracking fast-moving objects in real-time. In addition to the practical benefits of using event cameras for collision detection in UAVs, there is also a significant potential for innovation and research in this area. The use of event cameras in Unmanned Aerial Vehicle (UAV) collision detection represents a relatively new and emerging field, with many opportunities for innovative ideas and approaches with the potential to make significant contributions to the safety and reliability of these systems. As such, it is an excellent topic for a dissertation, offering the opportunity to make a meaningful impact and contribute to the advancement of this important field.

While there has been some research on using event cameras for collision detection on UAVs, there are still many open questions and opportunities for further investigation. For example, robust algorithms are needed to effectively track obstacles in complex environments, and the performance of these algorithms needs to be evaluated in realistic scenarios. The safety of UAVs is closely tied to public perception and acceptance of this technology. If UAVs are perceived as unsafe or unreliable, it will be difficult to gain widespread adoption and trust in their use. By prioritizing the development of reliable and effective collision detection systems, we can help build confidence in the safety of UAVs and pave the way for their widespread adoption in the future.

1.1 Objective

The primary objective of this work was to develop a collision detection algorithm that is robust and reliable. The algorithm developed is available online to anyone interested in the practical aspect of this topic. Additionally, a dataset with event camera recordings that mimic real-world situations was created to support this effort. This dataset provides a valuable resource for researchers and developers who wish to compare and enhance their own algorithms. The excitement surrounding this project is due to the limited number of online event camera collision detection algorithms available currently. The ultimate goal is to foster new and innovative UAVs applications through the use of the proposed algorithm.

1.2 Supplementary material

The Python implementation of the algorithm can be found here:

https://github.com/jppaulo13/COLLISION-DETECTION-USING-EVENT-CAMERAS.git

The datasets recorded to test and evaluate the algorithm can be found here:

https://drive.google.com/drive/folders/1lhNRML8HVNRZbxZB26XQtKz80BL1BURB?usp=share_link

1.3 Document structure

This work is encompassed by the present introduction as well as five other chapters structured in:

1. Theory – This chapter provides a comprehensive overview of the fundamental concepts and principles related to collision detection for UAVs and event cameras.

2. State of the art – This chapter examines the existing literature on collision detection algorithms, focusing specifically on the use of event cameras and highlighting the strengths and weaknesses of the algorithms.

3. Methodology – This chapter describes the methodology used in this study to develop the proposed event camera collision detection algorithm, as well as the dataset.

4. Experimental results – The chapter presents the results of the experimental tests conducted to evaluate the performance of the proposed event camera collision detection algorithm. It also explains some decisions made during the development of the algorithm.

5. Conclusion and future work – The final chapter summarizes the main findings and conclusions of the study. It also highlights the contributions of the proposed algorithm and suggests potential future research directions to further improve the

algorithm's performance. Finally, it proposes a possible implementation in hardware.

2

Theory

2.1 Cameras

Cameras are technical devices designed to capture and record visual information. They are utilized across a broad range of applications, including but not limited to photography, videography, surveillance, and scientific research. The field of camera technology has undergone a rapid evolution in recent years, leading to the development of an array of different camera types with varying features and capabilities.

On the other hand, computer vision is a subset of artificial intelligence that empowers computer systems to derive significant insights from digital images, videos, and other visual inputs. These insights can then be used to make decisions or provide recommendations based on the processed information. In simpler terms, Artificial Intelligence (AI) enables computers to think, whereas computer vision allows them to see, perceive, and comprehend visual inputs [56]. The past 60 years of research in computer vision have primarily focused on frame-based cameras.

2.1.1 Standard cameras

Frame-based video cameras, also known as standard cameras, utilize lenses that focus incoming light onto a sensor chip. This sensor chip contains an array of light-sensitive pixels. When the shutter, whether it is mechanical or electronic, is open, the pixels collect light for a specific exposure duration, forming an image, as illustrated in figure 2.1. These sensors are designed to capture sequences of images, creating a video by sampling the scene based on an external clock, such as 15 frames per second (fps) or 15 Hertz (Hz) [25].

However, frame-based cameras have significant limitations, such as high latency, motion blur, low dynamic range, and redundant sampling. These limitations can be detrimental for certain applications, such as in the field of robotics and UAVs. In contrast, event cameras, also known as asynchronous or dynamic vision sensors, have revolutionized the field of computer vision by addressing these limitations because unlike frame-based cameras, they don't suffer from the aforementioned problems.

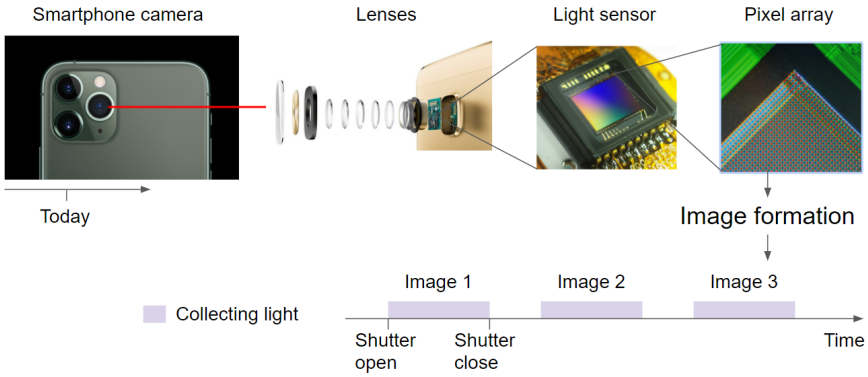

Figure 2.1: Frame-based camera work principle [18].

2.1.2 Event Cameras

An event camera, also known as a dynamic vision sensor, uses smart pixels that operate independently and asynchronously [44]. The pixels only transmit bits of information if they detected change of brightness in the scene, otherwise they stay silent. These bits of information are called events. Since the events are generated asynchronously, the output of the event cameras is a sequence of events instead of full frames. These sensors sample the scene based on how intensity evolves at each pixel, therefore there is no need for an external clock. An event includes the pixel coordinate position, the time, and 1-bit polarity of the intensity change, brightness increase (ON), or brightness decrease (OFF). If the pixel does not detect a sufficient intensity change, the pixel stays silent and does not transmit/trigger any information [23]. In the scene, only the informative pixels (the ones that transmit information) will appear. The image 2.2 shows an example of the output of a standard camera and an event camera.

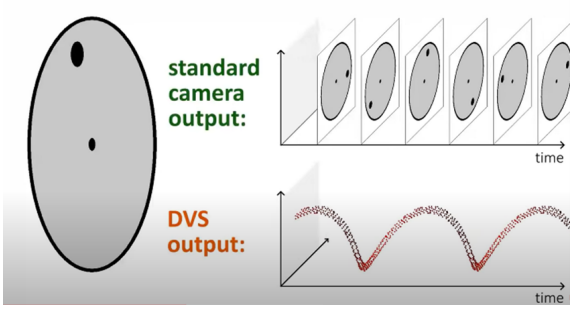

Figure 2.2: Event camera vs standard camera [21].

2.1.3 Event cameras advantages

In order to navigate safely and avoid obstacles or other objects, it is crucial for a robot to quickly detect their presence and take appropriate actions. This is especially important when the robot and the object are moving quickly relative to each other, as any delay in perception could have serious consequences. Perception latency, which refers to the time

it takes for the robot to perceive its environment and analyze the data in order to generate control commands, plays a vital role in this process [19].

The paperwork [19] explores the trade-off between the maximum speed a flying robot can achieve while securing safe navigation to the robot´s hardware and the surrounding environment. After analysing the impact of the latency on the maximum speed for a monocular frame-based camera, a stereo frame-based camera, and a monocular event camera, the results allowed to conclude some ideas:

1. When the sensing range (the distance where every event in it can be detected by the sensor) of the camera and the robot's agility is small, the difference between the 3 types of cameras is not noteworthy.

2. The stereo frame-based camera and the event camera improve their performance and, as a result, their speed by increasing their agility and sensing range.

3. The difference in maximum speed between stereo frame-based cameras and event cameras becomes noticeable after a spike in the agility of the robots.

Event cameras can help UAVs fly faster by 7-12% compared to using a traditional frame-based camera. The advantage of using an event camera is even greater for robots that are highly agile. This is according to research on the subject. Although in some situations, the stereo camera and the event camera had a similar performance, event cameras have a leverage in other aspects:

1. Event cameras do not need to deal with the impracticality of having two cameras.

2. For small drones/quadrotors not having two cameras makes the platform lighter.

3. Having a high dynamic range makes the event cameras more suitable for navigation in adverse lighting situations.

4. Event cameras' latency is independent from the exposure time, which can significantly decrease their latency.

5. Higher temporal resolution reduces the motion blur and makes obstacle detection easier at high speed.

6. Low power consumption.

This paper proved that the latency of event cameras between the time a visual signal is triggered and the time it is processed to output control commands is significantly shorter than that of standard cameras. Despite the good results of event cameras, the robotic industry must adapt its algorithms to this new way of capturing visual information. Standard vision algorithms cannot be applied, since the output of an event camera is a stream of asynchronous events rather than images, therefore new algorithms need to be developed to fully exploit the potential of event cameras.

In order to fully understand the advantages that event cameras offer in comparison to traditional frame-based cameras, it is necessary to have a deeper understanding of certain concepts.

2.1.3.1 Latency

Latency refers to the delay or delay time between the occurrence of an event and the detection or recording of that event by a sensor or system. In the context of cameras, latency refers to the delay between when an event occurs in the scene and when the camera captures that event [59]. High latency in a camera system can negatively impact the performance of that system in certain applications, such as robotics, UAVs, and high-speed motion tracking.

Frame-based cameras, which capture images of a scene at a fixed rate, typically have high latency due to the delay between the time an event occurs and the time the camera's sensor captures that event. This delay is known as the shutter lag, which includes the time it takes for the sensor to read out the previous frame and for the mechanical shutter to open and close. The high latency of frame-based cameras can be problematic in high-speed motion tracking applications, as the delay can result in a significant discrepancy between the state of the scene at the time of capture and the state of the scene at the time the event occurred.

Event cameras, on the other hand, are designed to detect and output pixel-level changes in the scene. As it was mentioned, event cameras work by outputting a stream of events that correspond to changes in the light intensity at each pixel. These events are generated as soon as a change in intensity is detected, reducing the sensor latency. This approach allows event cameras to achieve low latency, even in the range of microseconds, which can be crucial in high-speed motion tracking applications.

2.1.3.2 High dynamic

Dynamic range refers to the ability of a camera or imaging system to capture and represent a wide spectrum of brightness levels in a scene, from the darkest shadows to the brightest highlights [58]. High Dynamic Range (HDR) cameras are able to capture and represent a broader scope of brightness levels than traditional cameras, which can result in more detailed and accurate images and videos.

Frame-based cameras, also known as traditional cameras, typically have a low dynamic range. This is because the sensor in the camera can only capture a limited range of brightness levels. When the scene has a high contrast, with both bright and dark areas, the camera will struggle to capture the details in the bright and dark areas simultaneously. This can result in washed out highlights 2.3 or blocked-up shadows in the final image.

Event cameras have a high dynamic range due to their ability to detect and output pixel-level changes in the scene and operating in a logarithmic representation which

results in a higher dynamic range, enabling them to represent a much wider spectrum of brightness levels.

Figure 2.3: Dynamic range example [4].

2.1.3.3 Motion Blur

Motion blur is an artifact that can occur in photographs and videos when the motion of an object or the camera itself causes the image to appear blurry. Motion blur can be caused by a slow shutter speed, a moving camera, or a moving subject [26].

In traditional frame-based cameras, motion blur can occur when objects in the scene are moving quickly while the camera is capturing an image 2.4. This is because the shutter is open for a fixed duration of time, and during that time, the objects in the scene may have moved. The result is a blur in the image, which can make it difficult to identify or track moving objects. This can be especially problematic in high-speed motion tracking applications, where the motion blur can significantly reduce the effectiveness of the tracking algorithm.

Event cameras, however, don't suffer from the problem of motion blur. When an object is moving, the event camera will detect the movement and output events that correspond to it, rather than capturing a blurry image. This allows event cameras to provide a clear representation of the moving objects, which can be especially useful in high-speed motion tracking applications.

Moreover, some event cameras also have high frame rate capability, meaning they can output events at high frequencies up to tens of kilohertz. This can provide an additional advantage for high-speed motion tracking, as the higher frame rate allows the camera to capture more information about the scene and helps to reduce the motion blur.

2.1.3.4 Redundant sampling

Redundant sampling refers to the acquisition of data that does not contain any new or useful information [57]. In the context of cameras, redundant sampling occurs when a traditional frame-based camera captures images of a scene at a fixed rate, regardless of

Figure 2.4: Motion Blur example [26].

whether there is any change in it. This means that many images may be captured with the same scene, resulting in redundant data 2.5.

Aforesaid, frame-based cameras typically sample the scene based on an external clock, such as 15 frames per second (15 fps, 15 Hz). This fixed rate of acquisition can result in the capture of many images with the same scene, even when there are no changes, leading to a high degree of redundancy in the data. This can be a problem in applications where storage and computational resources are limited, as the redundant data can consume large amounts of storage space and require additional processing time to be analyzed.

On the contrary, the events from event cameras are generated as soon as a change in intensity is detected, which allows the camera to capture more information about the scene and represent it more accurately. Additionally, event cameras also work in an asynchronous mode, meaning they only output events when there is a change in the scene and they don't have to operate at a fixed frame rate.

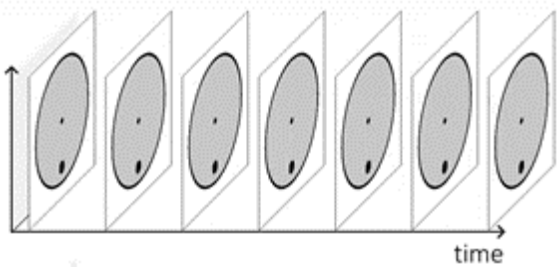

Figure 2.5: Redundant sampling demonstration [44].

2.2 Collision avoidance systems

The use of unmanned aerial vehicles has been marred by frequent collisions with birds, other objects, or intentional sabotage by malicious individuals, resulting in damage to equipment and the environment. In order to address these concerns and ensure safe operations, it is crucial for UAVs to have the fastest possible response time. UAVs have gained widespread attention in various fields such as military, commercial, search and rescue, traffic monitoring, border security, and atmospheric research due to their ability to access hazardous locations without endangering human lives. Therefore, it is essential

to develop collision avoidance algorithms to enable UAVs to operate autonomously and avoid obstacles or other objects during flight [61].

2.2.1 Perception

The first step in any collision avoidance algorithm is a perception, which involves acquiring data around the UAVs' surroundings. This step primarily utilizes sensors, which can be divided into two main categories: active and passive ones.

Active sensors have their own power source and do not rely on external factors such as sunlight, allowing them to operate effectively both during the day and at night. Additionally, active sensors are able to operate in adverse weather conditions and are not susceptible to interference. They emit a pulse of electromagnetic radiation towards the target to be measured and detect the radiation that is reflected back. These sensors can be further divided into several subcategories such as radar, LiDAR, and ultrasonic sensors.

Radar sensors use radio waves to detect objects. They can work in all-weather conditions, including fog, rain, and dust. LiDAR sensors use laser light to measure distance and detect objects. They can operate in all-weather conditions, including fog, rain, and dust. Ultrasonic sensors use sound waves to detect objects. They have a shorter range compared to radar and LiDAR sensors. Passive sensors, on the other hand, use the range of the optical spectrum and can only operate during daylight conditions. These sensors do not emit any energy, instead, they read the energy discharged by the object to be measured or captured. An example of a passive sensor is a camera. Cameras can be used to detect and track objects in the UAVs' environment. They are sensitive to light and can capture images and videos in high resolution.

In addition to cameras, other passive sensors include infrared sensors, which can detect heat signatures, and ultrasonic sensors, which can detect sound waves. These sensors can be used in combination with active sensors to provide a more comprehensive understanding of the UAVs' environment and improve the accuracy of collision avoidance algorithms.

In conclusion, the perception step in a collision avoidance algorithm for UAVs is a critical component that enables the UAVs to understand its environment. Active and passive sensors are the two main types of sensors used in this step, and each has its own unique capabilities and limitations. By using a combination of these sensors, UAVs can improve their ability to detect and avoid potential collisions, ensuring the safe operation of the system.

2.2.2 Action

The second and final step in any collision avoidance system for UAVs is the action step, which involves maneuvering the UAVs to avoid a collision. This process can be divided into four different types of actions: geometric, force-field, optimized, and sense and avoid.

The geometric approach uses the location and velocity information of the UAVs and the obstacles to ensure that a minimum distance between the UAVs and the obstacles is maintained. This is typically achieved by simulating potential trajectories for the UAVs and selecting the one that results in the greatest separation from the obstacles.

The force-field approach uses attractive and repulsive forces to repulse the UAVs from obstacles and attract the UAVs to its goal. This method relies on knowledge of the position and size of the obstacles, as well as the motion and geometry of theUAVs.

The optimized approach uses probabilistic search algorithms to calculate the best collision avoidance trajectory based on geographical information. This method balances computational complexity with response time, making decisions quickly while minimizing the computational power required.

The sense and avoid approach focuses on simplifying the collision avoidance process by breaking it down into individual detection and avoidance of obstacles. This approach allows each UAVs in a group to manage its own course without needing to know the plans of the other UAVs.

Overall, the action step in a collision avoidance system for UAVs is critical for ensuring the safe operation of the system by maneuvering the UAVs to avoid collisions. Each of the four types of actions has its own strengths and weaknesses and the choice of action depends on the specific requirements of the UAVs system and the operating environment.

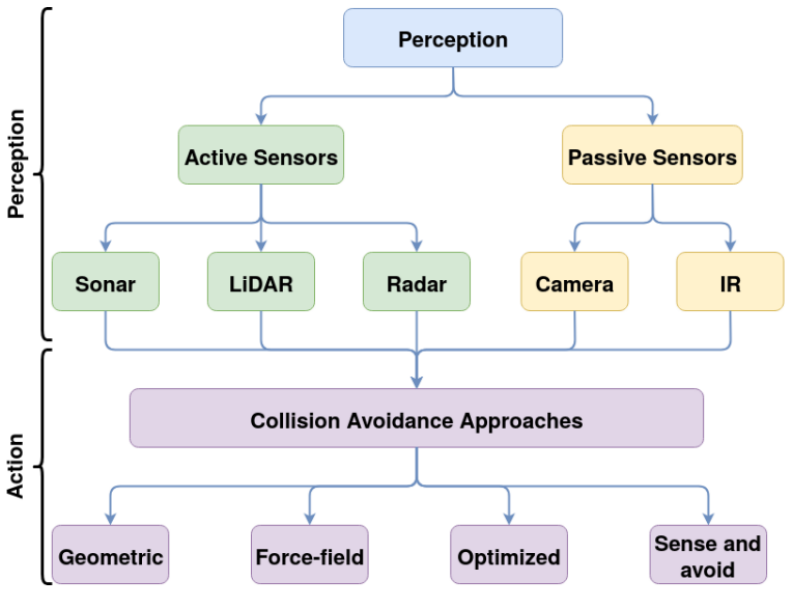

Figure 2.6: Collision avoidance system generalised modules [61].

A collision avoidance system must be capable of:

1. Detecting an obstacle and its attributes (e.g. position).
2. Conclude if the obstacle is approaching.
3. If there is a risk of collision.

4. Make calculations and perform a safety collision avoidance maneuver.

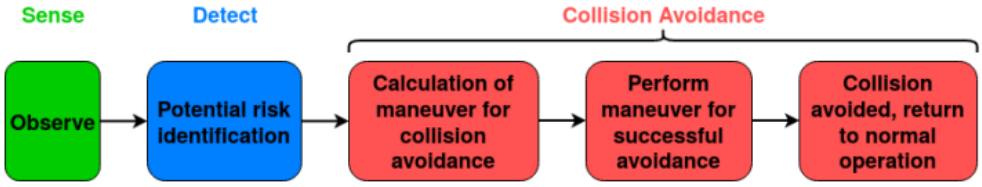

Figure 2.7: General process for collision avoidance [61].

2.3 Collision detection

In the context of collision avoidance systems for UAVs, the perception step is where the collision detection algorithms are implemented. These algorithms are essential for ensuring the safe operation of UAVs by utilizing a combination of sensors and computational methods to detect and avoid potential collisions with other objects in the UAVs' operational environment. The goal of collision detection is to identify and locate objects that may pose a threat to the UAVs and to provide the necessary information for the UAVs' control system to take appropriate action to avoid a collision.

One common approach for collision detection is to use stereo cameras to capture images of the UAVs' environment in 3D. These images are then processed using computer vision algorithms to detect and track objects in the scene. Computer vision algorithms such as stereo matching, optical flow, and feature-based methods are used to extract the 3D information of the scene. This approach can be used to detect both static and dynamic objects, such as buildings, trees, and other UAVs.

Radar and LiDAR sensors are also commonly used for collision detection in UAVs. These sensors use radio waves or laser light to measure the distance to nearby objects. They are particularly useful for detecting objects in adverse weather conditions or at night when cameras may not be effective. LiDAR sensors can provide high-resolution 3D point clouds of the environment, while radar sensors can detect objects at longer ranges and can provide information about the object's velocity.

Another type of sensor that can be used for collision detection is ultrasonic sensor, which emits sound waves and measures the time it takes for the sound waves to return. This method is mostly used for short-range collision detection and is useful for detecting obstacles that are close to the UAVs, such as walls or ceilings.

In addition to detecting objects, collision detection systems in UAVs also need to determine the UAVs' own motion and trajectory. Ego-motion compensation is an important aspect of collision detection systems. It refers to the correction of the motion of the sensor or camera mounted on the UAVs to account for the motion of the UAVs itself. This concept is crucial in a wide range of applications, including self-driving vehicles, industrial robot arms, and other autonomous navigation systems. By accurately estimating

the motion of the object itself, it is possible to gain a more precise understanding of the surrounding environment, which can help prevent algorithmic and mathematical errors and ultimately reduce the likelihood of accidents.

One common method for estimating ego-motion in standard cameras is through visual odometry techniques, which analyze a sequence of images captured by the moving camera. Visual odometry is a technique used in robotics and computer vision to determine the location and orientation of a robot by evaluating the related camera data [40].

However, the estimation of ego-motion in event cameras, which are specialized types of cameras that can detect changes in the scene at high temporal resolution, requires different methods or variations of the same techniques used for standard cameras. These specialized cameras are different from standard cameras and have different characteristics which have to be taken into account when estimating ego-motion.

Once an object is detected and its motion is estimated, the UAVs' control system can take appropriate action to avoid a collision. This may involve changing the UAVs' speed, direction, or altitude, or taking evasive maneuvers. The control system uses the information provided by the sensors and motion estimation algorithms to plan a safe trajectory for the UAVs to follow.

In conclusion, collision detection systems in UAVs are a critical component of ensuring the safe operation of these systems. They use a combination of sensors, such as cameras, radar, and LiDAR, to detect and track objects in the UAVs' environment, and use visual odometry to determine the UAVs' own motion and trajectory. Once an object is detected, the UAVs' control system can take appropriate action to avoid a collision, ensuring the safe operation of the system. The method used to detect and avoid collisions may vary depending on the specific requirements of the UAVs system and the operating environment.

3

State of the art

3.1 Dynamic obstacle avoidance for quadrotors with event cameras

One application for event cameras in flying robots is obstacle detection. Creating efficient algorithms for detecting objects and avoiding collisions is, among others, a high-potential field for UAVs and event cameras. A researcher from the University of Zurich proved the effectiveness of a moving-obstacle detection algorithm in an article [20]. As the article said, "Our moving-obstacle detection algorithm works by collecting events during a short-time sliding window (figure 3.1) and compensating for the motion of the robot within such a time window."

3.1.1 How does the algorithm work?

Event cameras generate events either from moving objects in the camera´s view or from ego-motion. Ego-motion is the estimated movement of the camera (mounted on the robot/UAVs) within the environment. To generate events from only moving objects, the first step for developing the algorithm was to remove all data generated by the UAVs' ego-motion. This method is called ego-motion compensation.

3.1.2 Ego-motion compensation by the article

Ego-motion compensation has been addressed in various papers and with various approaches. The paper under discussion used a simpler and a more computationally efficient ego-motion compensation method aiming to reduce to the fullest the latency. To do so, they estimated the ego rotation using an average of the IMU's angular velocity. This Inertial measurement unit (IMU) data was collected during a small specific time span using an IMU sensor.

IMU is a device, which can be integrated into the event camera, that can measure and report specific gravity and angular rate of the UAVs [2]. Then they compute the normalized mean time stamp for each pixel (figure 3.2).

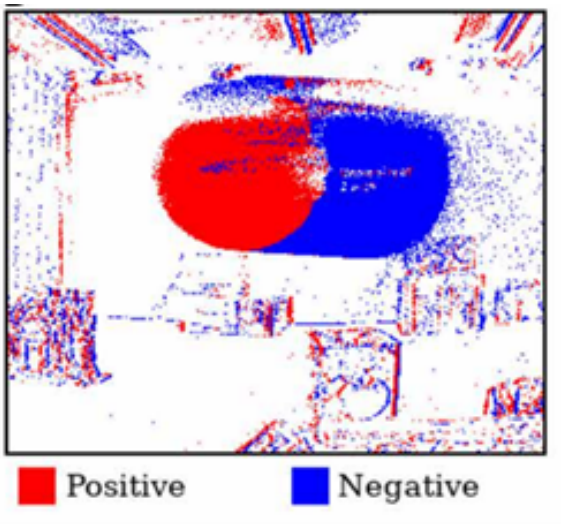

Figure 3.1: Events accumulated in defined time span [20].

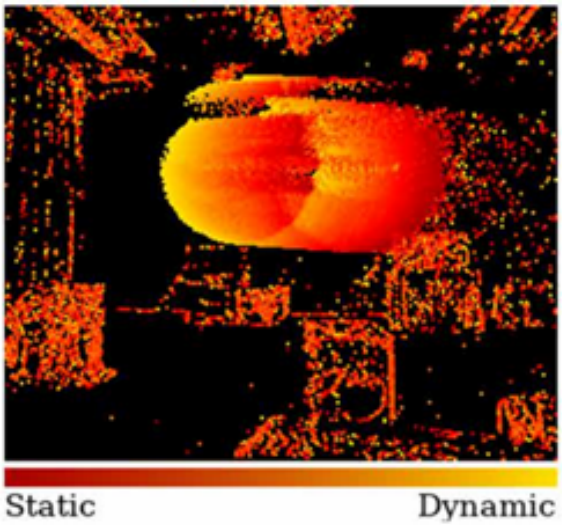

Figure 3.2: Events with colour code representing the normalized mean time stamp [20].

The normalized mean time stamp is the process to determine which pixels belong to a moving object and which to the background. After normalizing the data, they can then categorize it. This mathematical technique allows to assign scores to the events and then through a thresholding procedure, to set apart the events from moving objects and the rest (figure 3.3).

After the ego-motion compensation, the algorithm can now only deal with pixels that have at least one remaining event.

The possibility of the presence of multiple moving obstacles and noise in the scene requires the next step to separate the individual objects and the noise from each other. This process is called clustering (figure 3.4).

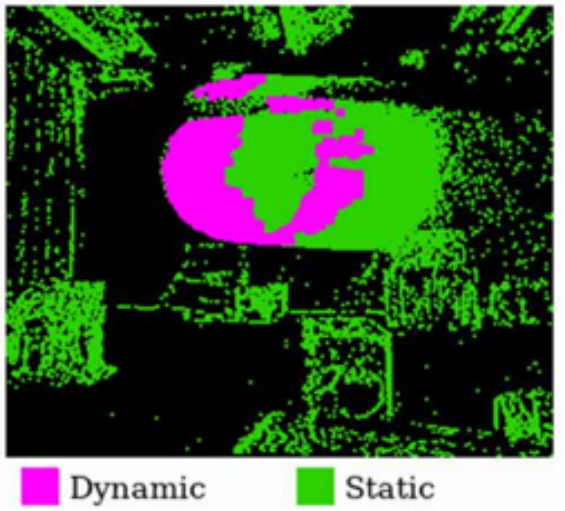

Figure 3.3: After thresholding [20].

Figure 3.4: Obstacle segmentation [20].

3.1.3 Optical flow

Optical flow is about understanding how things are moving in a scene and how motion is happening. The optical flow calculation allows estimating the image plane velocity from the image points. It is difficult to group objects with varying velocities and distances from the UAV using solely the information obtained from ego-motion compensation. Therefore, it is necessary to incorporate an optical flow estimation to the algorithm. Following an investigation, the researchers determined that the Lucas-Kanade algorithm would be suitable for processing the unthresholded normalized mean time stamp image obtained from ego-motion compensation. This optical flow algorithm is advantageous in that it is less susceptible to noise. The Lucas-Kanade algorithm operates under the assumption that the optical flow in a small neighborhood of the scene is constant across all points within that neighborhood. Hence, the optical flow equation, according to the Lucas-Kanade method, can be expressed as shown in figure 3.5, where the q_i are the pixels

inside the neighbourhood, the I_x, I_y, I_t are the partial derivatives of the image I, in this case, the unthresholded normalized mean time stamp image produced by the ego-motion compensation, with x and y being the pixel´s position and t the time. The w_i values (weights) are important because it is how the algorithm can be adjusted to have a better performance. In practice, it is better to give more weight to the pixels that are closer to the central one. The V_x and V_y gives us the image flow vector from the neighbourhood. This image flow vector (velocity) will then be useful to maximize the accuracy of the clustering. More information about this topic can be found in [3].

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i w_i I_x(q_i)^2 & \sum_i w_i I_x(q_i) I_y(q_i) \\ \sum_i w_i I_x(q_i) I_y(q_i) & \sum_i w_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i I_x(q_i) I_t(q_i) \\ -\sum_i w_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

Figure 3.5: Optical Flow equation – Lucas-Kanade method [3].

3.1.4 Image to world projection

Once the velocity of the objects has been calculated, the algorithm must then determine their position relative to the UAVs. Subsequently, this information is fed into a fast avoidance algorithm, which is designed to exploit the low sensing latency. To get the position of the objects, it is required to estimate the depth and size of the objects to the image plane. If the UAVs is monocular, meaning it only has one camera, the depth calculation requires knowing the size of the objects before the estimation. Otherwise, it is only necessary to fit a rectangle around the cluster points and get the four corner points and the centre position in the image plane. And with this data and the method cluster´s disparity, explained in detail in the article [52], the algorithm estimates the depth. The size is then calculated using the equation in Figure 3.6, where f is the focal length, ${}_c\hat{Z}$ the depth and w the measure side length of the fitted rectangle.

$$\omega_{\text{est}} = \frac{{}_C\hat{Z}\hat{\omega}}{f}$$

Figure 3.6: Size Equation [3].

After finding the obstacle´s size and depth, the algorithm transforms the plane image in a 3D space. And by leveraging the intrinsic camera matrix, along with the scale factor and the Cartesian coordinates of each point in the cluster, the algorithm is capable of identifying the specific points that correspond to the obstacle's location. A total of six points. Now that we have the position of the objects it is possible to estimate their velocity. It was estimated using a Kalman filter [6], with the object's position as the input.

3.1.5 Obstacle Avoidance

In order to create a fast avoidance algorithm, the researchers utilized a reactive avoidance scheme that relied on artificial potential fields.

3.1.5.1 What is the artificial potential field method?

The method involved constructing an artificial potential field that simulated the use of magnetic force to draw the robot towards its goal and repel it from obstacles in the environment. The force applied to the robot was the sum of the attractive potential field and the repulsive potential field, and was incorporated into the linear speed equation on the kinematic robot. More information on the equations and calculations used can be found in the source article [32]. The block diagram control 3.7 for the artificial potential field includes potential fields for the x and y axes, each of which requires input data on the robot's location, the obstacles' positions on the x and y axes, and the desired endpoint. Once the force is calculated, the algorithm produces a velocity command that is input into the quadrotor's Proportional Derivative Controller (PD) controller, which helps to stabilize the system by predicting future errors in the response.

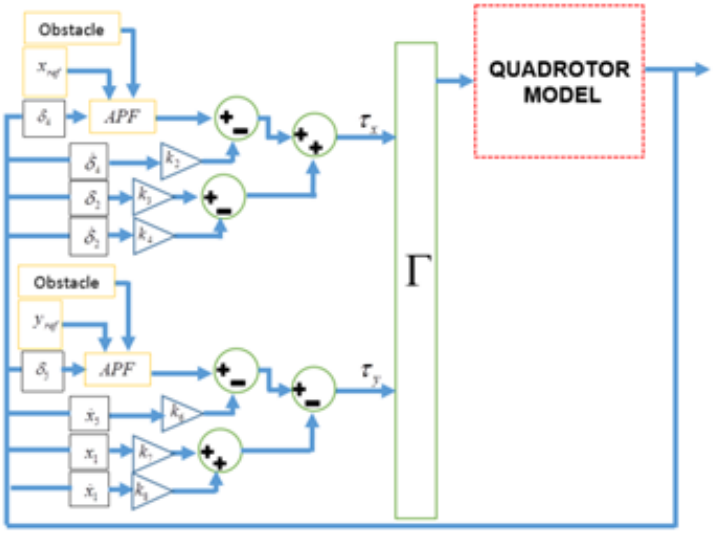

Figure 3.7: Potential field algorithm for quadrotor [32].

Constructing a discretized map is a typical approach for artificial potential fields, however, this mapping approach is applicable in 2D space (x and y axes), but it requires more computational resources to extend it to 3D space. To overcome this limitation, the researchers chose to model the obstacles as ellipsoids and estimated their position and velocity to compute their repulsive forces at each time step. Additionally, they calculated the attractive force towards a given target position. The resulting combined force from these calculations was used to generate a velocity command for the controller to execute.

3.1.5.2 Ellipsoids

An ellipsoid is a three-dimensional closed surface that possesses a unique and interesting property: all plane cross sections made across it are either circles or ellipses. Moreover, the shape of the ellipsoid exhibits a remarkable symmetry that is characterized by having three mutually perpendicular axes that intersect at the center of the ellipsoid. This means that the ellipsoid looks identical when viewed from any of these axes, and is known as a triaxial ellipsoid. Due to its elegant geometrical properties and symmetry, the ellipsoid has a wide range of applications in various fields of science, engineering, and mathematics, including astronomy, geodesy, physics, and statistics.

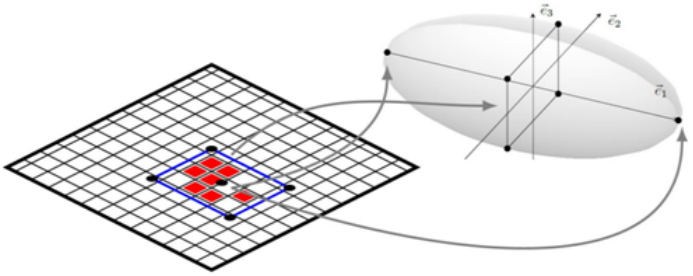

Figure 3.8: Obstacle ellipsoid [20].

The figure 3.8 shows the process of constructing an ellipsoid obstacle in the world's frame of reference, based on the clustered data in the image plane. Specifically, an iterative approach is used to fit a minimal volume ellipsoid around six projected points, which are calculated through the image-to-world projection.

The resulting algorithm has an impressively low overall latency of just 3.5 milliseconds. This latency is sufficient to detect and avoid potential collisions with objects of varying shapes and sizes, even when the relative speeds of these objects are as high as 20 meters per second.

3.2 Night vision obstacle detection and avoidance based on bio-inspired vision sensors

Researchers from University of Turku created an algorithm [61] to help UAVs to avoid collisions in low lighting conditions, such as night-time scenes. Hence, they opted for event cameras due to their ability to gather data under varying lighting conditions. Event cameras do not use a shutter, enabling each pixel to operate autonomously, and the photoreceptors of the pixels function using logarithmic scaling.

3.2.1 How does the algorithm work?

This algorithm consists of four main units: noise cancellation, object detection, depth estimation, and asynchronous adaptive collision avoidance.

3.2.2 Noise cancellation

The introduction states that event cameras are designed to only detect changes in brightness, but hardware limitations such as current leakage and thermal noise can cause events to trigger in the background. To address this issue, the researchers developed a filtering algorithm to remove background noise. The algorithm examines a selected event and a 9×9 pixel area surrounding it, with the selected event at the center. Using a k-nearest neighbor algorithm, the filter assesses the correlation between the selected event and its neighbors. If the event does not have enough correlated neighbors, it is treated as noise and discarded.

3.2.2.1 K-nearest neighbour algorithm

This algorithm belongs to the supervised learning category in machine learning, it is used mostly for classification. The k-nearest neighbour considers the k nearest points to the selected point and predicts the class or its value according to these neighbours. The algorithm chooses the nearest points based on their distance to the selected point. The most popular distance metric used is the Euclidean distance. The algorithm will then classify the point based on the majority of classes in the neighbours, or in the case of this filter the number of neighbours [27].

3.2.3 Object detection

The researchers chose not to adopt the approach described in the algorithm [20]. Instead, they developed a new method that involves accumulating "N" events based on the velocity of the objects, which produces a more precise event frame. Object detection is performed by fitting a local plane using a randomized Hough transform on the accumulated events. The researchers decided to use this approach because the previous method of accumulating events during a fixed time interval led to either noisy or blurred event frames.

3.2.3.1 3D Hough transform

The Hough transform is a technique used to detect objects, and in this context, it utilizes three random events as input and produces a 3D space or parameterized planes as output. The Hough space is defined by three parameters: φ, θ and ρ. Here, θ represents the angle of the normal vector on the xy-plane, φ denotes the angle between the xy-plane and the normal vector in z, as shown in Figure 3.9, and ρ indicates the distance from the origin of the coordinate system.

To find the plane which corresponds to the object, it is necessary to solve the Hough Transform for each of the 3 random events. Given a point (event) P in cartesian coordinates and solving the equation (3.1), it creates a 3D sinusoid curve for each point, and the intersection of the curves corresponds to the plane 3.10.

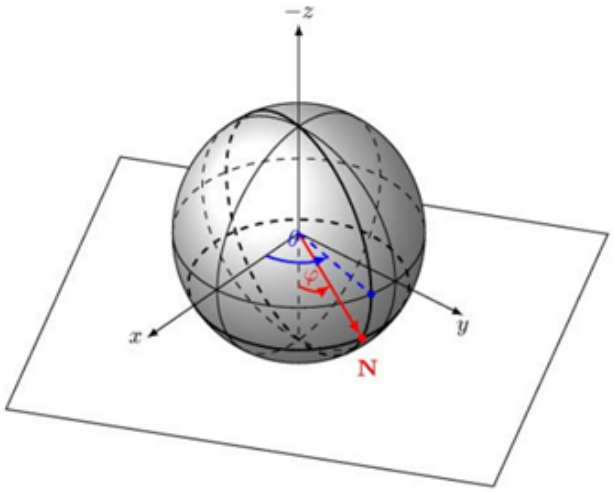

Figure 3.9: Normal vector described by polar coordinates [7].

$$P_x \times cos\theta \times sin\varphi + P_y \times sin\theta \times sin\varphi + P_z \times cos\varphi = \rho \tag{3.1}$$

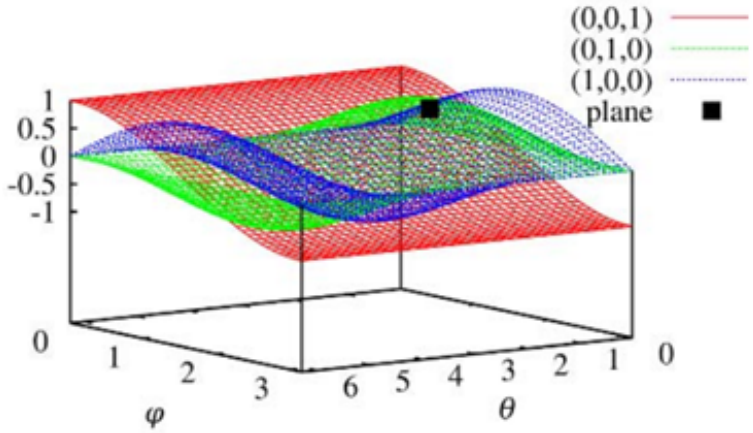

Figure 3.10: Transformation of three points into Hough space [7].

The intersection of the curves in the Hough space is represented by a black point or pixel. In this approach, the selection of the three points used to construct the plane is done randomly. This process is called randomized Hough transform. After randomly selecting three points from a point cloud, a corresponding cell in the Hough space, denoted as A(θ, φ, ρ), is incremented. If the point cloud consists of a plane, the corresponding cell in the Hough space will have a high score. On the other hand, if the three points are widely spread apart, they are unlikely to belong to a plane, and the corresponding cell in the Hough space will have a low score. For more information on the 3D Hough transform, please refer to [7].

3.2.4 Depth estimation

After detecting the object, it is now crucial for the algorithm to estimate the depth of the objects. To do this, the researchers decided to use a more friendly derivation of the eHarris, which they called LC-Harris. They created a new version of the eHarris algorithm because this one is computationally too heavy and complex.

3.2.4.1 eHarris

The described approach involves detecting corners in an image by analyzing the average change in image intensity. This is accomplished by calculating the spatial gradient of the intensity. If the matrix containing the first derivatives of intensity has two large eigenvalues, it is indicative that the pixel is a corner. The matrix is calculated using the equation (3.3), where (3.2) is the gradient of an obtained binary surface and e is the current event.

$$\nabla \sum_{b} = \left[\frac{d \sum_{b}}{dx}, \frac{d \sum_{b}}{dy} \right] \tag{3.2}$$

$$M(\mathbf{e_i}) = \sum_{\mathbf{e} \in W} g(\mathbf{e}) \begin{bmatrix} \frac{d\Sigma_b(\mathbf{e})}{dx}^2 & \frac{d\Sigma_b(\mathbf{e})}{dx} \frac{d\Sigma_b(\mathbf{e})}{dy} \\ \frac{d\Sigma_b(\mathbf{e})}{dy} \frac{d\Sigma_b(\mathbf{e})}{dx} & \frac{d\Sigma_b(\mathbf{e})}{dy}^2 \end{bmatrix} \tag{3.3}$$

To conclude if an event is a corner event, the method associates the eigenvalues with a score, R (3.4). Where λ_1 and λ_2 are the eigenvalues of M.

$$R(\mathbf{e_i}) = \det(M) - k(trace(M))^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{3.4}$$

If the calculated R value for a given event e is greater than a pre-set threshold S, the event is classified as a corner event, according to the method described in [53]. Based on this approach, the researchers developed a new method called LC-Harris, which involves extracting a binary local patch of size 9×9 around each new event occurrence. The most recent neighbors are labeled as 1 in the local patch, and the horizontal and vertical gradients are computed using the binary local patch. The identification of corner events is based on a score calculated from the gradients. Finally, the location, orientation, and extracted corners from the LC-Harris algorithm are used to estimate depth through triangulation.

3.2.5 Asynchronous Adaptive Collision Avoidance

After detecting the obstacles, it is now necessary to estimate their relative velocities. The researchers estimated the velocities with an approach [38] that takes into account the lifetime of each event independently and displays the event for that period. Estimating the lifetime of an event gives the velocity of that event, because the lifetime of an event is

the time interval that an event is considered active. And an event is considered active if the brightness gradient causing the event is visible by the pixel. The velocity of the event is estimated using event-based visual flow [5]. Visual flow is the way of projecting a 3D perspective and motion over time of a 2D static capture, or in this case 2D frame from a time span of events. With the velocity of the UAVs (v) and a timespan ($t_i - t_{i-1}$), the distance travelled by the UAVs can be calculated using the equation (3.5).

$$d_v = v * (t_i - t_{i-1}) \tag{3.5}$$

Then the distance travelled by the obstacles (d_{obji}) can be calculated using the equation (3.6).

$$d_{obji} = \rho_{i-1} - d_v - \rho_i \tag{3.6}$$

where ρ_{i-1} and ρ_i are the distances between the UAVs and the obstacle at t_{i-1} and t_i respectively. Based on the point of collision, the algorithm assigns the highest priorities to the obstacles with the closest points of collision. These priorities are constantly monitored and updated, in case a new obstacle appears in the scene. Then the point of collision with the highest priority is chosen for path planning, which triggers an avoidance maneuver.

3.3 EVDodgeNet: deep dynamic obstacle dodging with event cameras

3.3.1 Introduction

The paper referenced [43] describes a framework that utilizes deep learning to enable UAVs equipped with event cameras to avoid unknown dynamic obstacles. The UAVs in question are equipped with a single front-facing event camera, a lower-resolution down-facing event camera, a sonar for measuring altitude, and an IMU. The proposed AI framework allows for dodging, evading, and avoiding dynamic obstacles using solely on-board sensing and computation, with no prior information. The framework consists of both perception and control modules. The perception module is divided into three segments, as depicted in the figure 3.12.

3.3.2 EVDeBlurNet

The researchers developed a neural network called EVDeBlurNet for improving the quality of event frames. The network takes an event frame as input, which consists of events triggered in a spatio-temporal window. The frame may be blurred due to camera movement, causing misaligned events. To address this, EVDeBlurNet identifies point trajectories along the spatio-temporal point cloud and optimizes a function that balances high

contrast and similarity to the input image, resulting in a higher-quality frame. The input frame has three channels representing the per-pixel average count of positive and negative events and the average time between events per pixel.

3.3.3 EVHomographyNet

After deblurring the event frame, the algorithm must estimate the ego-motion/odometry. The method used the downfacing camera of the UAVs, the IMU and the distance sensor to gather data for the ego-motion estimation. The researchers adapted the works from two articles [39] and [14], both the unsupervised and supervised learning algorithm trains a deep convolutional neural network to estimate planar homographies 3.11.

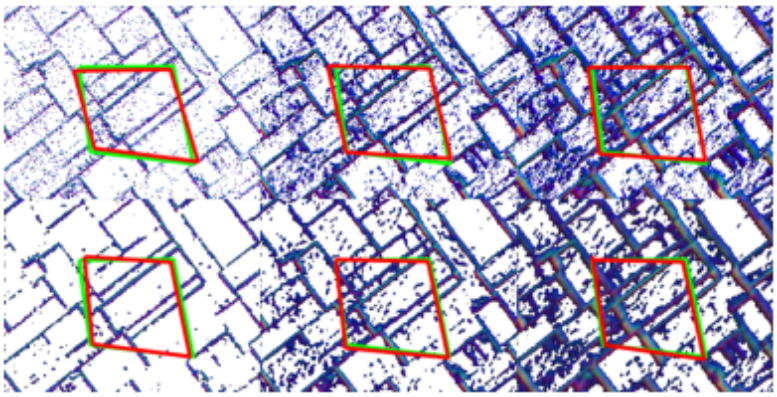

Figure 3.11: Output of EVHomographyNet for raw and deblurred event frames at different integration times. Green colour denotes the ground truth, and red colour denotes the predicted ego-motion [43].

3.3.4 EVSegFlowNet

EVSegFlowNet is a method that combines segmentation and optical flow learning in a semi-supervised manner. One approach to segment moving objects is to use simulated data with known segmentation for each frame and train a Convolutional Neural Network (CNN) to predict the Independently Moving Objects (IMO) and background segmentation using cross-entropy loss. However, estimating 3D IMO motion using a monocular camera without prior knowledge is impossible, so the method predicts a safe trajectory based on the velocity direction of the IMOs on the image plane. This velocity direction can be obtained by tracking the segmentation mask of the IMO or computing the mean optical flow direction of the region of interest, which is computed using a CNN trained by the researchers.

3.3.5 Datasets

In order to train their networks, the researchers generated synthetic scenes using randomized wall textures, objects, and object/camera trajectories. This approach allowed for an

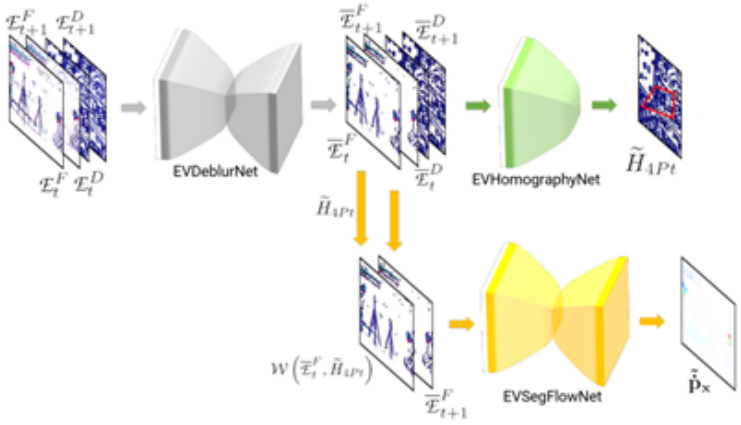

Figure 3.12: Overview of the proposed neural network-based navigation stack [43].

unlimited amount of training data to be generated with one or more moving objects in the scene. The researchers followed a similar approach as described in a previous work [42] to generate the scenes. They created seven unique configurations, each consisting of a room with three objects in motion, as shown in figure 3.13. By training on simulated data, the networks were able to be directly transferred to the real world without any re-training or fine-tuning.

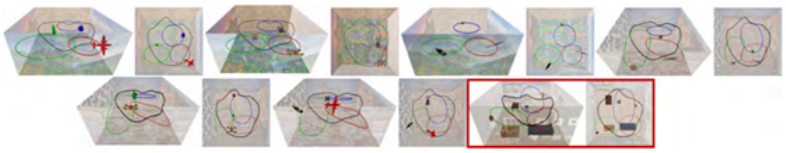

Figure 3.13: Various scene setups used for generating data [43].

3.4 Event-based moving object detection and tracking

The system proposed in this work [35] considers the 3D motion estimation and segmentation with no prior knowledge about its motion or the scene. The ego-motion estimation is obtained directly from the event stream, and the dynamic objects are detected and tracked based on the inconsistencies in the motion field. Event stream is a cloud of events within a small time interval. The implemented algorithm can be divided in two segments: motion compensation, and object detection and tracking.

3.4.1 Motion compensation

The first section of segment uses a 4-parameter model (3.7) to compensate for the background motion and detect the objects. The 4-parameter model consists of: the shift parallel to the image plane (h_x,h_y), a motion towards the image plane (h_z), and the rotation around the z axis (θ).

$$M^G = \{h_x, h_y, h_z, \theta\} \tag{3.7}$$

The algorithm uses the model on the global warp field (3D representation) to describe the distortion caused by the camera motion on the event cloud. The goal is to transform the original event coordinates $\{x, y, t\}$ into new coordinates $\{x_0, y_0, t\}$ that are aligned with a reference frame. The transformation is represented mathematically by (3.8), where the new coordinates $\{x_0, y_0\}$ are a function of the original coordinates $\{x, y\}$ and the parameters of the model. For the sake of simplicity, the timestamp is excluded and remains unaltered during the transformation. The authors assume linear event trajectories within the time slice, which means that the motion of each event can be described by a straight line.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - t * \left[\begin{bmatrix} h_x \\ h_y \end{bmatrix} + (h_z + 1) * \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right] \tag{3.8}$$

3.4.2 Object detection and tracking

In order to identify objects that are in motion, this section assigns a score to each pixel, based on the variations in τ. τ is a discretized plane that displays the average timestamp of events projected onto each pixel by the warp field. The score indicates the deviation between independently moving objects and the background. By using this technique, the pixel can be labeled either as a component of the background or as part of a moving object.

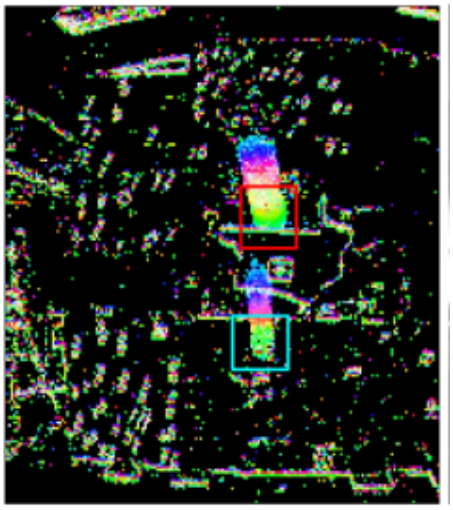

Figure 3.14: A frame captured from the "Two Objects" dataset, displaying the misalignment between two objects after global motion compensation has been applied. The green hue represents the most recent events [35].

After detecting the moving objects 3.14, the algorithm implements a simple Kalman filter to prevent missing or wrong detections.

3.4.3 Datasets

The DAVIS240B bio-inspired sensor was used to collect the data. They gathered over 30 recordings in total, but the sequence with a periodically flashing bright light in a dark room and another UAVs moving is the dataset's centrepiece. The bright light in a dark room is an excellent indicator for the algorithm because it causes a lot of noise.

3.5 A unifying contrast maximization framework for event cameras

This article [22] presents a framework for handling numerous computer vision challenges using event cameras: estimation of motion, depth, and optical flow.

3.5.1 General description of the framework

The framework produces two products: estimated point trajectories that implicitly establish correspondences between events and the trajectories which can be used to correct the edge motion.

The framework finds the point trajectories on the image plane that best fits the event data. The image plane 3.15 is a set of events acquired during a time window (typically with milliseconds interval).

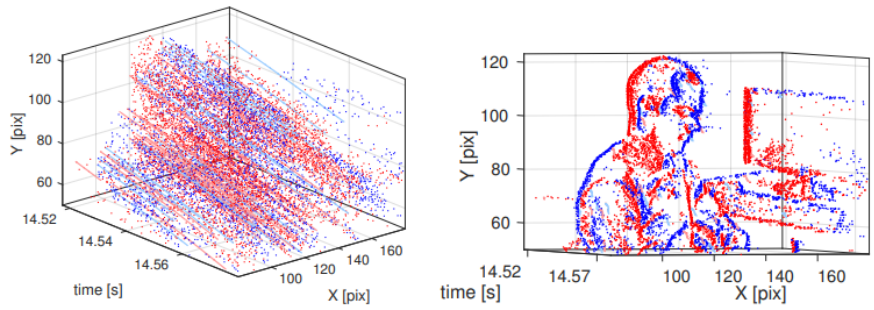

Figure 3.15: (a) The space-time region of the image plane is depicted with events (dots) caused by a moving edge pattern, and the corresponding point trajectories. The events are colored based on their polarity, blue indicating a positive event, which represents an increase in brightness, and red indicating a negative event, representing a decrease in brightness. (b) The corresponding events align along the direction of the point trajectories highlighted in (a), providing a visual representation of the edge pattern that caused them [22].

A geometric model of how points move on the image plane is created by the framework and depends on the type of estimation required to solve the computer vision problem (i.e., optical flow, depth or motion). The goal is to estimate the model's parameters using the data provided by the events. The estimation is presumably achievable because the

model parameters, which are unknown a prior, are shared by numerous events and are observable (meaning that they can be measured). To tackle the estimation problem, the framework builds candidate point trajectories according to the motion and scene models, and then uses an objective function to evaluate the level of matching between these trajectories and the event data. After this, an optimization algorithm is used to search for the point trajectories that maximize the objective function, which measures how well events are aligned along the candidate trajectories.

3.5.2 Steps of the method

1. To create an image (H), the events are warped based on the point trajectories specified by the geometric model and proposed parameters (θ). This involves translating the events along a trajectory that accounts for their space-time coordinates and other characteristics of the point-trajectory model, which is known as warping. The warp shifts each event along the trajectory that passes through it until it reaches a reference time.

2. The next step is to compute a score (f) based on the image of the warped events. An objective function f(H(E')) is computed based on the image of histogram of warped events H(E'). The researchers use the variance of H as a dispersion metric with the aim to maximize it. The objective function measures the quality of fit to the event data (E) by representing the statistics of the warped events (E') as a function of the candidate model parameters (θ). In other words, maximizing the variance of the image of warped events H(E'(θ)) favours the point trajectories that align the warped events on the image plane.

3. The objective function is optimized by tuning the parameters of the model to obtain the best possible fit to the event data. This involves finding the optimal point trajectories on the image plane that align with the event data. Various optimization algorithms such as gradient ascent or Newton's method can be used to find the best model parameters. The framework is flexible and not tied to a specific optimizer.

3.6 Event-based motion segmentation by motion compensation

The article [50] describes a proposed method for event-based motion segmentation, which aims to classify events of a space-time window into separate clusters that represent coherent moving objects or background, even in the presence of a possibly moving camera. The method is inspired by classical layered models and uses motion compensation techniques. It jointly estimates the motion parameters of the clusters and the event-cluster associations in an iterative, alternating fashion, using an objective function based on motion compensation. The proposed method is flexible and can handle different types

of parametric motions of the objects and the scene, such as translation, rotation, and zooming.

3.6.1 Method

In this study, a method was developed to improve motion segmentation in computer vision by combining classical layered models with event-based motion compensation. The former models [55] are a type of computer vision model that represents a scene as a set of layers or regions, each corresponding to a different object or surface in the scene. The idea is that each layer is a two-dimensional surface that moves independently of other layers, and the observed image is a combination of these moving surfaces.

Event-based cameras, also known as the Dynamic Vision Sensor (DVS), have independent pixels that output "events" in response to changes in intensity. The method developed in this study involves processing these events in packets to aggregate sufficient information for the estimation. The main problem addressed by this method is a motion segmentation, which involves identifying which events correspond to which objects or a background motion.

The proposed solution involves classifying events into clusters or layers representing coherent motion, with each cluster having constant motion parameters. To achieve this, the method leverages the idea of motion compensation to separate events into clusters by maximizing event alignment. This is done by warping events to a reference time and maximizing their alignment, producing a sharp image of warped events (Image of Warped Events (IWE)).

Warping events to a reference time refers to transforming the timestamps of the events in a way that aligns them to a common temporal reference frame. This is important because events in different parts of the visual field may occur at different times due to the relative motion between the camera and the scene. Maximizing their alignment refers to finding the transformation that best aligns the events across the temporal and spatial dimensions. This transformation is determined by optimizing an objective function that measures the alignment between the events in different parts of the visual field.

The goal of this process is to group events that correspond to the same object or a background motion together, which can then be used for further analysis or processing. Multiple motion models or clusters are required to achieve maximal event alignment for multiple objects with different motions. The sharpness of the IWE is used as the main cue to segment the events, identifying the events corresponding to each independently moving object as well as the object's motion parameters.

Figure 3.16 shows the output of the event-based camera, which is a sequence of events in response to changes in intensity, along with a colour image of the scene for illustration purposes. The middle block represents the iterative clustering algorithm proposed in the method, which classifies the events into clusters representing coherent motion. The right-hand side shows the segmented moving objects causing the events, which include

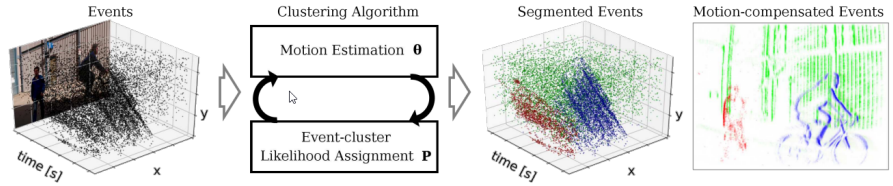

Figure 3.16: Event-based motion segmentation by motion compensation [50].

a pedestrian, a cyclist, and the camera's ego-motion. The objects are colour-coded for clarity.

The method jointly estimates the motion parameters θ and event-cluster membership probabilities P to best explain the scene, resulting in motion-compensated event images for all clusters on the right-hand side. θ represents the motion parameters of each cluster, such as the speed and direction of the object's movement. The event-cluster membership probabilities P represent the probability of each event belonging to a particular cluster.

The motion-compensated event images for each cluster show the events that correspond to each independently moving object. The colour-coded segmentation of the moving objects makes it easier to identify and track each object's motion parameters. Overall, this figure demonstrates the effectiveness of the proposed method in accurately segmenting moving objects in the scene captured by the event-based camera.

3.6.2 Conclusion

This work presents a novel method for per-event segmentation of a scene into multiple objects based on their apparent motion on the image plane. Per-event processing involves analyzing each event individually, rather than a series of frames captured at fixed time intervals. This approach allows for more efficient and accurate processing of visual information, especially in dynamic scenes with fast motion or lighting changes. The proposed method identifies the motion parameters of different objects (clusters) that cause the events while segmenting them. Additionally, the method produces motion-compensated images with a sharp edge-like appearance of the objects in the scene, which can be used for further analysis, such as a recognition.

The method outperforms two recent methods on a publicly available dataset, with as much as a 10% improvement. It can resolve small relative motion differences between clusters and achieves this using a versatile cluster model while avoiding explicit estimation of optical flow for motion segmentation, which is error-prone.

This method allows for motion segmentation in challenging conditions, such as in high-speed scenes, unlocking the outstanding properties of event-based cameras. Overall, this work presents an effective method for per-event segmentation of a scene into multiple objects, which has important implications for computer vision and image processing.

3.7 Ego-motion estimation based on fusion of images and events

The event camera is a novel bio-inspired vision sensor that produces an event stream. In this study [60], a novel data fusion algorithm called Events Aggregation and Superimposition (EAS) is proposed to merge conventional intensity images with the event stream. The fusion output is then applied to various ego-motion estimation frameworks and evaluated on a public dataset collected in low light environments.

3.7.1 Method

The implementation of EAS occurs in two stages: firstly, events are aggregated into event slices, and secondly, the corresponding event slices are superimposed on intensity images.

The event camera outputs an event stream, where the i^{th} event can be represented by a 4-dimensional vector $e_i = (x_i, y_i, t_i, p_i)$, with x_i and y_i denoting the pixel coordinates of the event, t_i representing the time stamp, and p_i representing the polarity, indicating the increase or decrease of light intensity (usually denoted as 1 or 0). In this study, due to events with opposite polarities being generated for the same edge moving in different directions, the polarity is ignored, and both types of events are treated equally.

To extract and track features effectively, a fixed number of events or events within a fixed time interval are aggregated along the time dimension, resulting in an event slice that contains sufficient information. A fixed number is preferred over a fixed time interval because the camera's speed can vary widely, leading to a significant difference in the number of events generated in a fixed time interval, making the aggregation result unstable. The aggregation process is expressed in equation (3.9), where es denotes the event slice, and $sign(.)$ is defined as follows: $sign(x) = 1$, if $x > 0$; $sign(x) = 0$, if $x = 0$; N is the number of events aggregated in an event slice. In other words, the aggregation process involves setting a pixel value to 1 if it generates any events in the event slice, and to 0 otherwise.

$$es = sign(\sum e_i), \quad i_0 \leq i < i_0 + N \tag{3.9}$$

Superimposing, also known as image fusion, is a process of combining two or more images of the same scene, taken from different sources or at different times, to create a single composite image that contains more information than any of the individual images. In the context of computer vision and image processing, superimposing usually involves aligning and blending an image obtained from a traditional camera with an image obtained from a specialized camera, such as a depth, thermal, or event camera, in order to enhance or highlight certain features of the scene. The process of superimposing can be achieved through various methods, such as image registration, intensity normalization, colour mapping, and blending.

In this case, the alignment of event slices 3.17(c) with original intensity images 3.17(a) in time is necessary for their fusion. Each intensity image has a corresponding event slice obtained at the exact moment the image is captured. Before superimposing, the event slice is first smoothed by a Gaussian filter and weighted. This weighting is determined by the value of the parameter α, which decides the degree of influence of the event slices on the fusion result. Higher α means that the event slices will have a greater impact on the final result. A threshold is set to restrict the superimposition and ensures that only pixels with values smaller than β can be enhanced by the corresponding event slice 3.17(b), meaning that pixels with values above this threshold will not be affected. This is done to ensure that the superimposition process does not cause damage to important features in the original image. An adaptive strategy to determine the weight α is used, which is set to the maximum pixel value of the intensity image with a lower bound to prevent over-amplification of small pixel values. The final fusion result 3.17(f) can be expressed mathematically using equation (3.10), where EAS represents the final fusion result, I is the original intensity image, and $G(.)$ denotes the Gaussian filter function.

$$EAS(x,y) = \begin{cases} I(x,y) + \alpha \cdot G(es(x,y)), & I(x,y) < \beta \\ I(x,y), & I(x,y) \geq \beta \end{cases} \tag{3.10}$$

The image in 3.17 shows various sets of images from a dataset. The EAS image (f) has the best quality among all the images in the set. This demonstrates the effectiveness of the EAS algorithm in improving the image quality of event camera data fused with conventional intensity images.

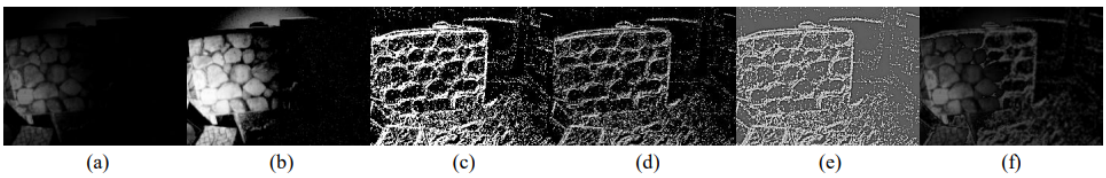

Figure 3.17: The first (a) is the original image, the second (b) is the enhanced version of the original image, the third (c) shows the event slices obtained from the event camera, the fourth (d) shows the temporal slice (TS) obtained by averaging the event slices, the fifth (e) shows the smoothed and integrated temporal slice (SITS), and the sixth (f) shows the final fused image obtained using the EAS algorithm [60].

3.7.2 Conclusion

The novel data fusion algorithm (EAS) to fuse conventional intensity images with the event stream output by event cameras proposed in the paper [60], was tested on two different tasks that involve estimating the movement of a camera (ego-motion estimation): one task that estimates rotation based on tracking the movement of objects in the scene, and another task that estimates both rotation and translation. The tests were conducted

on a dataset captured in low-light conditions. The results showed that EAS significantly improved the accuracy of rotation estimation compared to using only the conventional intensity images. These findings suggest that EAS is a promising approach for improving the accuracy of motion estimation in challenging lighting conditions by leveraging the unique strengths of event cameras.

3.8 Motion compensation and object detection for neuromorphic camera

The paper [54] discusses motion compensation and object detection for neuromorphic cameras. It presents two motion compensation and target detection methods and conducts a comprehensive experimental study on two datasets to explore the advantages and disadvantages of different algorithms for applicable scenarios. Specifically, the paper introduces two motion compensation methods for operating event flow based on differential neuromorphic camera. The first method involves motion compensation based on event count image and time image, while the second method involves motion compensation based on an image of warped events.

3.8.1 Motion compensation based on event count image and time image

The first method of motion compensation is a technique that utilizes two types of images, namely the "event count image" and the "time image", to estimate and compensate for motion. The event count image represents image stability, and it is calculated by summing up the number of events that occur at each pixel over a certain period. The time image, on the other hand, uses dynamic information from the event flow to create a representation of the motion. It measures the time interval between successive events at each pixel, providing information about the motion of objects in the scene.

To achieve motion compensation, a motion model is used to fit the 3D geometric characteristics of the event flow. It is used to estimate the moving system's ego-motion instead of locally calculating the image motion on a single event. This global method of motion compensation is more accurate than local methods because it accounts for the motion of the entire system rather than just the motion of individual events.

The motion compensation algorithm uses a 4-parameter global motion model to represent the global warp field, which expresses the transformation that is applied to the event flow to compensate for the movement. The event count image and time image are used to measure the error of event flow motion compensation. The algorithm tries to minimize the error between the predicted event flow and the actual event flow by adjusting the motion model parameters.

Overall, this method of motion compensation is effective in compensating for camera movement and other types of motion in the scene. By using a global model and two types of images, it can accurately estimate and compensate for motion in the entire system.

3.8.2 Motion compensation based on IWE

The second method of motion compensation estimates the parameters of a model based on the information in the event – the data point that contains the location, time, and polarity of an event in the image. Once the parameters are estimated, the event is warped for different motion models to find the point trajectory that is most suitable for the event flow. The event is then warped again according to the point trajectory described by the motion parameters, resulting in an IWE.

To determine the relative motion parameters between the neuromorphic camera and the scene, the polarity of the events is summed to obtain IWE. Before adding polarity, the event motion is realized by warping, and then the IWE corresponding to any pixel track is formed. To measure the quality of the IWE, contrast is used, which is a measure of the difference between the intensity of the event and the surrounding pixels.

The goodness of fit between the point trajectory and event is measured by using the contrast function of IWE as the objective function. By maximizing it, the point trajectory on the image plane that best matches the event can be confirmed, and the optimal parameters can be found. The contrast maximization method is used to distinguish the polarity of regions with and without events. Variance is typically used to measure the contrast of images, which is a measure of the dispersion or concentration of image values near the average intensity.

To estimate the motion parameters that can best compensate the rotational camera motion, the contrast of the rotation event image is maximized using the contrast maximization method. However, the contrast is a nonlinear function of unknown variables, and a closed solution of this problem is unlikely to exist. Therefore, an iterative nonlinear algorithm is used to optimize the contrast and make the motion compensation result optimal.

3.8.3 Object detection based on threshold

This method uses motion inconsistency to detect independent moving objects and accurately model the background. The algorithm iteratively detects and tracks objects that do not conform to the background model. To detect independent moving objects, the algorithm observes the inconsistency of time images and uses a threshold to detect pixels that move differently. Morphological operations group the pixels into objects and associate each pixel with a score representing its misalignment with the background. The score is then used to classify pixels as background or independently moving objects.

3.8.4 Object detection based on contrast maximization

This method uses a group of continuous events to detect moving objects. The event data is first segmented into clusters, each of which represents a coherent and independent moving object or the background. To accomplish this, an objective function based on

motion compensation is used to estimate the correlation between event clusters and the motion parameters of each cluster. By iteratively optimizing this objective function, the method can accurately identify and track different moving objects in the scene. Since each event carries very little information and lacks prior knowledge of the scene, the events are clustered together to gather more information and estimate the motion parameters of every cluster. Each of them represents a coherent moving object, and it is assumed that the motion parameters of the cluster are constant over time. To divide different moving objects in the scene, the event cluster association is modeled in the motion compensation framework to determine the possibility that an event belongs to a cluster. This is accomplished by optimizing the contrast of all the clusters. To solve this problem, an iterative alternating optimization method is used. This method updates the motion parameters and the event cluster association alternately to maximize the contrast of all the clusters. In each iteration, the motion parameters are updated using the gradient ascend method, while the event cluster association is updated using a probability partition law. In statistics, this law is used to calculate the probability of an event occurring given the knowledge of other related events [28]. In summary, this method uses event data to detect and track moving objects in a scene by clustering the events and estimating the motion parameters of each cluster using an objective function based on motion compensation. By iteratively optimizing this function, the method can accurately identify and track different moving objects in the scene.

3.8.5 Conclusion

The paper discusses different motion compensation and object detection methods for neuromorphic cameras. The authors propose two motion compensation methods: one based on event count images and time images and the other based on IWE. The experimental results show that both methods have different strengths and can be selected according to the needs. The former can produce motion-compensated images with clear edges, while the latter can highlight moving objects in the result. The paper also presents two object detection methods: one based on a threshold and the other based on a contrast maximization. The threshold-based method is suitable for object detection in many cases, while the other performs well for sequences with translational motion and can detect moving objects in more complex datasets.

4

Methodology

An event camera only transmits events from pixels where there has been a change in brightness intensity, so capturing an environment through an event camera on a static body will only generate events from pixels where there are moving objects. There may be some noise-generated data, but it is insignificant and can be easily removed during processing. If the object on which the event camera is mounted remains static, collision detection becomes simpler because most of the events will be part of one or more moving objects. As a result, it is only necessary to process the event data and its location to determine the number of dynamic objects and their trajectories.

However, during mission flights, UAVs equipped with event cameras can experience instability due to unfavorable conditions such as wind, resulting in additional motion. Therefore even without dynamic objects, the camera's motion generates significant events in the background. Therefore, to process events from moving objects, the algorithm needs to filter out events generated by the camera's motion. This process is known as "ego-motion compensation".

In this work, an initial approach was to replicate the detection algorithm performed in the paper [35]. This was possible because the authors of the article had made the code available online. A motion-compensation pipeline was available on GitHub in C++ language. In order to make the pipeline accessible to a wider audience, the developers also provided Python bindings for the functions. These bindings allowed the functions written in C++ to be called from Python code, enabling Python developers to utilize the motion-compensation pipeline in their projects. Python has become increasingly popular as a programming language in recent years, and for good reason. Not only is it versatile and powerful, but it also has a relatively easy-to-learn syntax that makes it accessible to beginners. Its readability and ease of use make it an attractive choice for a variety of applications. Moreover, Python has a vast library of modules and frameworks that can help streamline the development process. This makes it possible to accomplish complex tasks with minimal effort and helps to reduce the amount of code that needs to be written from scratch. Additionally, Python has a large and supportive community of developers, who are constantly creating new tools and resources that make the language even more

powerful and useful. Hence, the decision was made to utilize the available Python code instead.

Once the functions for reading data from the dataset were replicated, the next step was to prepare the ego-motion compensation pipeline. This involved creating a four-parameter model that described the distortion caused by the camera movement. The transformation of each event's coordinates was calculated using the equation (3.8).

Where $[h_x, h_y, h_z, \theta]$ are the model parameters that would later be optimized in order to find the best values so that the distortion caused by ego-motion would be as small as possible and $[x, y, t]$ are the coordinates and the timestamp of each event.

This optimization process proved to be quite complex and computationally inefficient, and another ego-motion compensation method was quickly sought that did not have the same problems.

As it was explained and researched in the state of the art, it is possible to create ego-motion compensation in a much simpler and more efficient way without using optimization processes and so much mathematical complexity, e.g. through methods that require the use of the IMU device. But for this, we had to collect our own data, as the data that was being initially used was obtained from the Internet and did not contain any IMU information.

4.1 Data

The first data sequences used were obtained from the Internet, specifically, from the article [35]. The data was obtained with the DAVIS240B bio-inspired sensor [11], which has a 3.3 mm lens and an 80-degree horizontal and vertical field of view. Most of the sequences were generated using a handheld device. They customized a Qualcomm Flight-TM platform to link the DAVIS240B sensor to the onboard computer and capture data in a realistic environment for the quadcopter sequences. The Qualcomm Flight is a complete drone reference design featuring a low-power, high-performance heterogeneous processing engine that offers 15 TOPS, long-range Wi-Fi 6 and 5G (optional) connectivity, support for 7 camera concurrency, computer vision, and vault-like security. Figure 4.1 illustrates the quadrotor + sensor platform arrangement. The fully loaded platform weighs 500 g and is powered by the Snapdragon APQ8074 ARM CPU, which has four cores running at up to 2.3 GHz.

Over 30 recordings were collected, but the sequences that we mainly used consisted of multiple recordings with 1 to 3 moving objects in normal lighting conditions. The objects are simple, with little or no texture in some cases. The objects move at different rates, either following linear paths or hitting a surface.

The article provides all datasets in two formats: text files and binary (rosbag) files. Although their contents are the same, one can be more suitable for a specific task. For this case, we used the text files since they can be loaded easily using Python. The binary

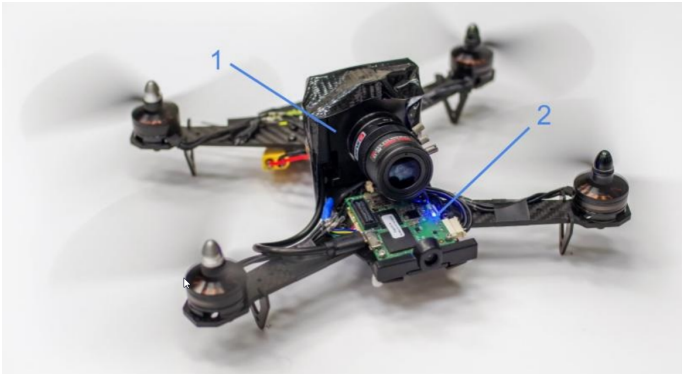

Figure 4.1: Drone used in the dataset collection. 1 – mounted DAVIS240B camera, 2 – customized Qualcomm Flight platform with onboard computer [35].

rosbag files are intended for users who are already familiar with the Robot Operating System (ROS) and for programs that will run on a real system.

The text file which contains the events is called *events.txt*. It has one event per line, with the timestamp in seconds, the x position, the y position, and the polarity, 1 being positive event and 0 being negative event.

As mentioned, this data did not have any IMU information. Therefore, we had to collect our own data and sequences to incorporate the IMU information ourselves.

4.1.1 IMU

The IMU is an electronic device/sensor that contains an accelerometer and a gyroscope, some may also contain a magnetometer or a barometer.

An accelerometer is the sensor responsible for detecting inertial acceleration, or the change in velocity over time, and is available in several configurations, including mechanical, quartz, and Micro Electro Mechanical System (MEMS) accelerometers. As demonstrated in Figure 4.2, a MEMS accelerometer is basically a mass sustained by a spring. The mass is referred to as the proof mass, and the direction in which the mass is supposed to move is referred to as the sensitivity axis. When a linear acceleration along the sensitivity axis is applied to an accelerometer, the proof mass shifts to one side, with the amount of deflection corresponding to the acceleration.

A gyroscope (figure 4.3) is an inertial sensor that measures the angular rate of an object in relation to an inertial reference frame. There are several different types of gyroscopes on the market, ranging in performance from mechanical to fiber-optic (Fibre-Optic Gyroscopes (FOGs)), ring laser (Ring Laser Gyroscopes (RLGs)), and quartz/MEMS gyroscopes. Quartz and MEMS ones are commonly employed in consumer, industrial, and tactical grade industries, respectively, whereas fiber-optic gyroscopes cover all four performance categories. Ring laser ones generally have in-run bias stabilities ranging from 1°/hour to less than 0.001°/hour, covering tactical and navigation grades. Mechanical gyroscopes are the best performing on the market, with in-run bias stabilities of less than 0.0001°/hour.

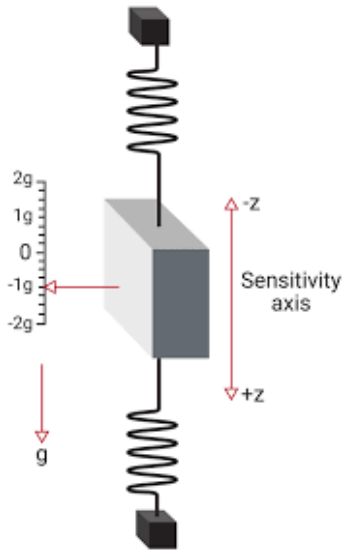

Figure 4.2: Simple accelerometer model [29].

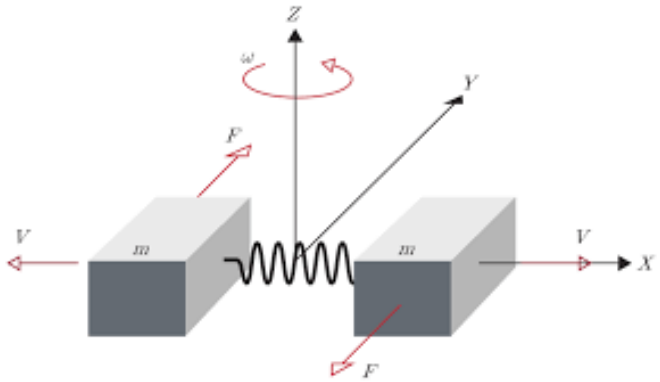

Figure 4.3: Simple gyroscope - tuning fork configuration [29].

A device known as an Inertial Measurement Unit utilizes one or more accelerometers and gyroscopes to detect linear acceleration and rotational rate. In some cases, a magnetometer is also included as a heading reference. For each of the primary axes, namely pitch, roll, and yaw, a typical setup comprises of one accelerometer, one gyroscope, and one magnetometer. IMU readings are commonly employed in Inertial Navigation Systems (INS) to determine the device's attitude, angular rates, linear velocity, and position relative to a global reference frame.

4.1.2 Recorded data

4.1.2.1 EVK1 camera

9 sequences of events and IMU data were recorded with an EVK1 HD event camera from Prophesee [17], which is a French technology company that specializes in event-based vision solutions. Prophesee's event camera evaluation kits (EVK1) are designed to provide developers with all the necessary tools and support to view and record event streams

captured by Prophesee's advanced event-based vision sensors. The sensors feature a number of unique characteristics that make them ideal for a wide range of applications, including autonomous vehicles, robotics, and industrial automation.

The camera used (figure 4.4), HD pixel-individually auto-sampling image sensor is capable of capturing visual data at high resolution and with high sensitivity, enabling developers to accurately capture and analyze visual data in real-time. The sensor also features a wide dynamic range of up to 120dB, which allows it to capture scenes with a high level of contrast, even in challenging lighting conditions.

Prophesee's event sensors use contrast detection events only, which means they are triggered by changes in the scene rather than at a fixed rate. This allows the sensors to capture and process visual data more efficiently, and enables them to operate at very high speeds.

Provided with the EVK1 evaluation kit is a D-FOV 81.5° CS mount lens, which ensures a wide field of view and is compatible with multiple camera systems. The kit also includes a power supply and data exchange with a standard USB 3.0 interface, which makes it easy to integrate the camera into existing systems and workflows.

One of the most important features of Prophesee's event sensors is their event time-stamping capability, which provides microsecond (µs) precision. This enables developers to accurately measure and track events as they occur in the scene, and to synchronize visual data with other sensor data for more accurate analysis and processing.

Figure 4.4: Prophesee EVK1 HD event camera [17].

The Prophesee EVK1 is equipped with a 6-axis IMU that combines a three-axis accelerometer and a three-axis gyroscope. The specific model of the IMU used in the EVK1 is not disclosed by Prophesee, but it is designed to provide high-precision motion tracking capabilities.

As will be explained later, the ego-motion compensation method created does not need the IMU values from the accelerometer and certainly not from the magnetometer or barometer. Only the values of the angular velocities obtained from the gyroscope are

required.

4.1.2.2 Recorded sequences

Each of the sequences recorded contains four files:

1. *data.raw* -– the recorded events.
2. data.avi — a short visualization of the recording.
3. event_data.csv — the events are stored in a text file, which has one event per row, with the timestamp in microseconds, the x position, the y position, and the polarity, "1" being positive event and "0" negative event.
4. imu_data.csv — the IMU values saved in a text file, each row has the timestamp, also in microseconds, and the 3 angular velocities (gx, gy, gz).

The 9 sequences are the following:

1. Static — a simple scenario recorded with a static camera and no moving objects in the scene. This sequence was recorded to check if the algorithm does not generate any false positives and to evaluate the algorithm's resilience to noise in the environment.
2. Dynamic Camera – a scenario recorded with a dynamic camera and no moving objects in the scene. The sequence was specifically designed to test the ego-motion compensation capabilities of the algorithm.
3. Dynamic Object -– 4 sequences recorded with a static camera and one or more moving objects in the scene. The purpose of recording these sequences was to evaluate the algorithm's ability to detect moving objects when ego-motion compensation is not required. By capturing sequences without any intentional camera movement, the algorithm's ability to accurately detect moving objects can be tested under stationary conditions. This type of testing is important for ensuring that the algorithm can operate effectively in real-world scenarios where camera movement may be minimal or non-existent.
4. Dynamic -– 3 sequences recorded with a dynamic camera and one or more moving objects in the scene. The purpose of recording these sequences was to comprehensively test the algorithm's capabilities, including its ability to perform ego-motion compensation and detect one or multiple moving objects. By capturing sequences with varying degrees of camera movement and the presence of one or more moving objects, the algorithm's ability to operate effectively in complex, real-world scenarios can be evaluated. This type of testing is critical for ensuring that the algorithm can accurately detect and track objects under a range of conditions and in different

environments. The sequences were specifically designed to provide a challenging testing environment that could accurately evaluate the algorithm's full capacities and its ability to perform under demanding conditions.

4.2 Ego-motion compensation

The ego-motion compensation algorithm collects the events from a predefined short time span, 10 ms was considered a suitable time window for our method after numerous tests and experiments. A shorter time window would make the algorithm insensitive and therefore unreliable, but with a longer time window, the number of events to process would also be larger, causing a longer delay in the algorithm's operation. This increase in events would not add any value, as it would only add events generated by the same motion as the dynamic objects but on a longer timeline.

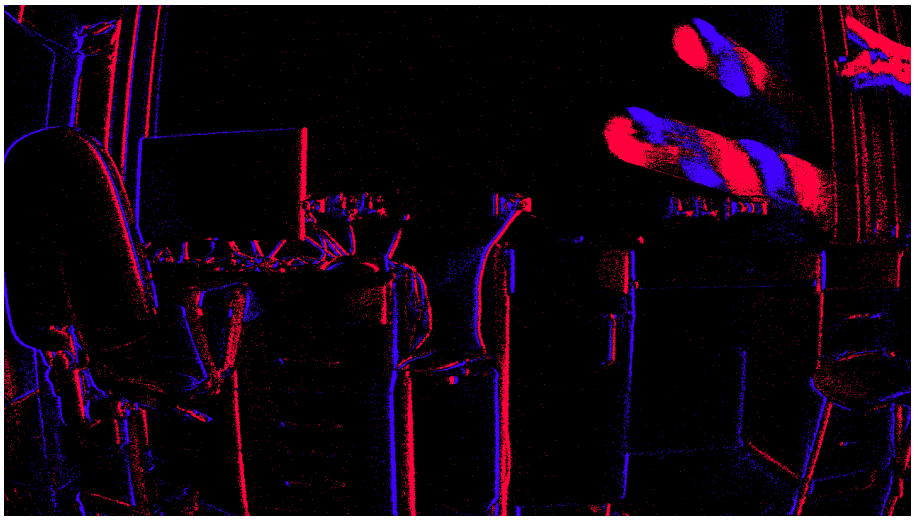

Figure 4.5: Events collected in a 10 ms time window from a scene with 3 moving objects and a moving hand. The positive events are displayed in blue and the negative events are displayed in red.

Through the UAV's rotational information, which is obtained with the IMU device, it is possible to know the correlation that exists in all the events created by the UAV's own rotation. This uniform movement of the events over the time window can be quantified by the information from the IMU. As only the events created by the rotation of the UAV itself have this uniform movement, if the algorithm can withdraw the influence of this movement, then only events created by dynamic objects will remain in the image, and the ones created by ego-motion will disappear.

After the algorithm of ego-motion compensation, it can be noticed that not all events created by the UAV's own motion disappear. This happens because the algorithm does not account for any translational motion of the UAV.

Based on the idea from the article [20], it was decided to only consider rotational motion in the algorithm's calculations. This decision was made because, with a short time

window of only 10 ms, translational motion is relatively insignificant, and incorporating it into the calculations would make the algorithm slower and more complex. Instead, it is more advantageous to filter out this movement and focus solely on compensating for the camera's rotational motion. By doing so, the algorithm can more efficiently and effectively detect and track objects without being bogged down by unnecessary calculations. This approach is in line with current research and allows for a more streamlined and optimized algorithm that can operate effectively in real-world scenarios.

The ego-motion compensation algorithm starts by calculating the rotation matrix. By selecting a time window and capturing the events, the values of the angular velocity of the IMU in the same window were also collected, and their average value was calculated. Then the rotation vector was formed by the average values of the angular velocities (4.1).

$$r = (g_x, g_y, g_z) \tag{4.1}$$

The rotation matrix was obtained using the Rodrigues algorithm and the rotation vector [41] [33]. Rodrigues' formula is an efficient technique for rotating a vector in a space given an axis and angle in three-dimensional rotation theory. The equations (4.2), (4.3) and (4.4) are used to convert a rotation vector into a rotation matrix.

$$\theta \leftarrow norm(r) \tag{4.2}$$

$$r \leftarrow r/\theta \tag{4.3}$$

$$R = \cos(\theta)I + (1 - \cos\theta)rr^T + \sin(\theta)\begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{4.4}$$

Inverse transformation can be also done easily using the equation (4.5).

$$\sin(\theta)\begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} = \frac{R - R_T}{2} \tag{4.5}$$

By multiplying the vector of an event in spatio-temporal space with the rotation matrix, we get the vector with the initial position of the event, prior to the rotation during the selected time window. A rotation vector is the most convenient and compact representation of a rotation matrix (since any rotation matrix has just 3 degrees of freedom).

Let's represent the rotation matrix as the following (4.6) and (4.7).

$$R = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \tag{4.6}$$

$$\begin{bmatrix} x' \\ y' \\ t_0 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ t \end{bmatrix} \tag{4.7}$$

After doing these calculations for all the events within the selected time window, the new events are now all assigned to t_0 (t_0 is the first event's time of the selected batch), going from a 3D projection to a 2D one. This process is called warping.

$$\begin{aligned} \left[x_1, y_1, t_1\right] &\rightarrow \left[x'_1, y'_1, t_0\right] \\ \left[x_2, y_2, t_2\right] &\rightarrow \left[x'_2, y'_2, t_0\right] \\ \left[x_3, y_3, t_3\right] &\rightarrow \left[x'_3, y'_3, t_0\right] \\ &etc... \end{aligned} \tag{4.8}$$

After the warping process, the algorithm faces a problem, as certain events will be transposed to positions that are not within the image limits. Using in this particular case an image with a height of 720 pixels and a width of 1280 pixels, after the calculation of the new positions, certain values could exceed these limits or fall short of them. Thus in these cases, it was decided to place these events in positions in the closest place within the limits. The figure 4.6 represents a drawing of this process. It is possible to conclude,

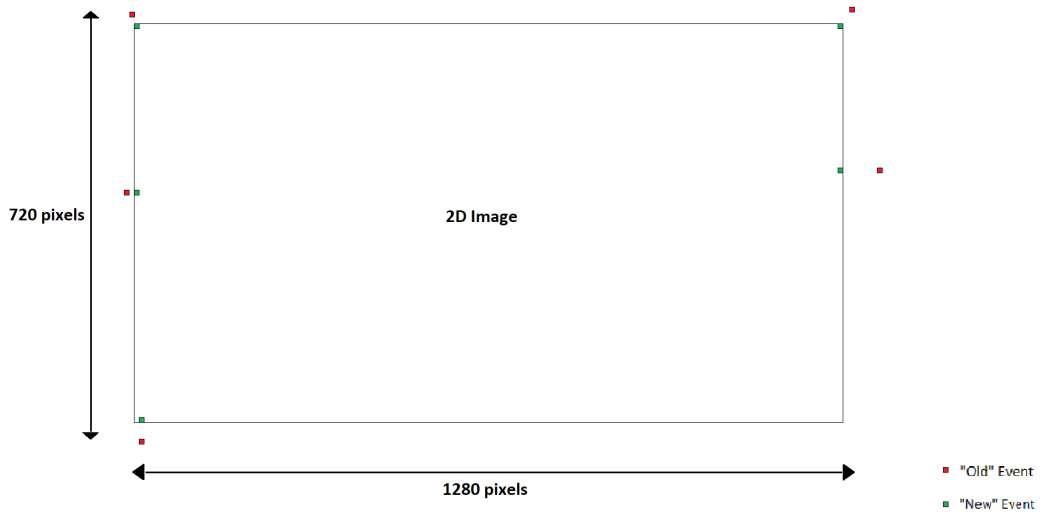

Figure 4.6: Representation of the events that after the transposition fall outside of the image limits. Red represents the position of the event after warping and green represents the position of the event after solving the boundary issue.

by comparing figure 4.5 with figure 4.7, that after the ego-motion compensation process, there is a higher density of events in the area where the dynamic objects are.

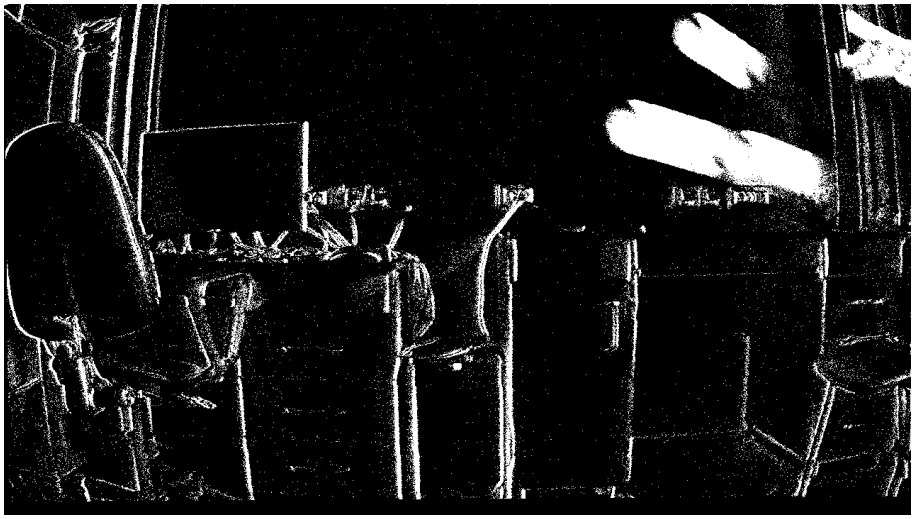

Figure 4.7: The result of the ego-motion compensation, showing in white all the pixels where there has been at least one event in the time window.

4.3 Normalized mean timestamp

The next step is to identify which pixels belong to a moving object and which belong to the background and to do so, each pixel must be assigned a score. First, from the warped events, their number in each pixel location is computed, which is called event-count pixel I_{ij}. Then, the time-pixel T_{ij} is built, which is the sum of all timestamps of all events within the pixel, and the mean timestamp in each pixel is calculated using the equation (4.9).

$$mean(T)_{ij} = \frac{T_{ij}}{I_{ij}} \tag{4.9}$$

Having the mean timestamp for each pixel, the algorithm must now calculate the average timestamp of all warped events as in equation (4.10).

$$mean(T) = \frac{T}{I} \tag{4.10}$$

The event-count image I presents the total number of events within the warped event image and the time-image T presents the sum of all timestamps of the events within the image. The given score for each pixel is computed using the equation (4.11).

$$p(i,j) = mean(T)_{ij} - mean(T) \tag{4.11}$$

The result of the equation (4.11) is called normalized mean timestamp. The normalized mean timestamp value has a range of [-1,1] (figure 4.8).

Computing a score allows to threshold the events (figure 4.9), separating static and dynamic objects (figure 4.10).

The articles [35] and [54] proposed the concept of assigning a score to each pixel and then utilizing a fixed threshold to differentiate between events generated by ego-motion

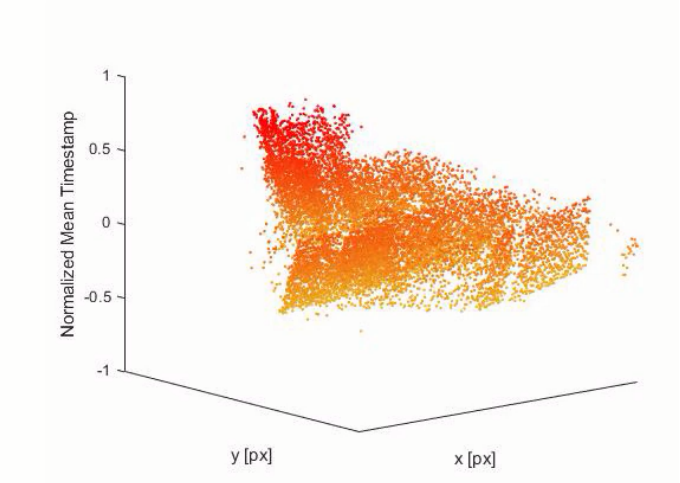

Figure 4.8: Example of the normalized mean timestamp.

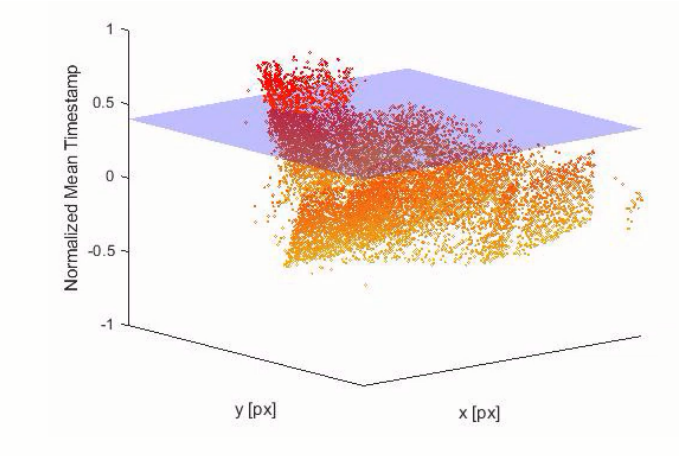

Figure 4.9: Example of a fixed threshold of the events.

and those produced by a moving object, as explained in the "Object detection based on threshold" sub-chapter 3.8.3. Following several tests, a threshold of 0,002 was set as it has a good compromise between the success of the results and the simplicity of the algorithm.

By implementing a Python loop that iterates through each pixel in the frame, it is possible to store the coordinates, the cumulative sum of events, and timestamps of the events occurring in that pixel in a 3D array (table 4.3). This makes it feasible to compute the average timestamp of the events and eventually determine the score for each pixel. Then, by implementing an *if* condition, the algorithm saves only the pixels with a score greater than a specific threshold in a string and removes the events from other pixels in the frame.

After applying thresholding, as demonstrated in the following image 4.11, the dynamic objects are now more easily discernible. However, it is possible that certain events from the static regions of the scene are not filtered out, resulting in salt-and-pepper noise

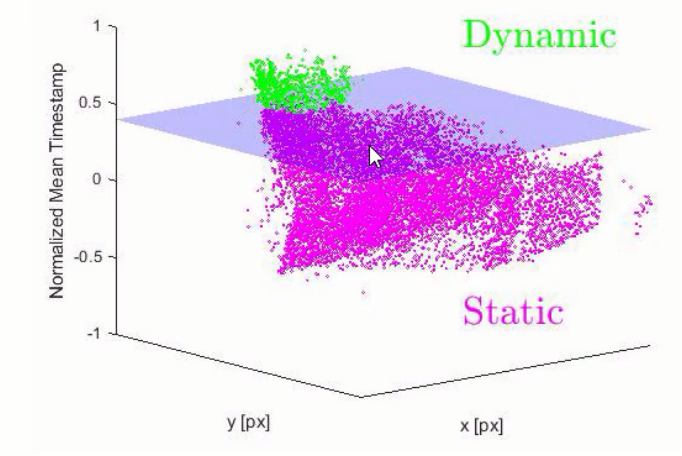

Figure 4.10: Example of separation between events from dynamic objects and the rest.

Array of pixels			
$x_{[0:total]}$	$y_{[0:total]}$	level	task
x_i	y_i	0	sum of events
x_i	y_i	1	sum of the timestamps
x_i	y_i	2	average of the timestamps
x_i	y_i	3	score
x_i	y_i	4	score > threshold ? “1” else “0”

Table 4.1: Demonstration of the array of pixels and the information stored in each of them.

that must be eliminated using morphological operations.

Figure 4.11: The result of the ego-motion compensation after thresholding.

Based on a comparative analysis of the figures 4.7 and 4.11, it can be inferred that a perceptible alteration occurs in the number of events subsequent to the thresholding procedure. A substantial portion of the events vanishes, while the remaining events persist primarily in regions where dynamic objects are present.

4.4 Morphological operations

Morphological operations are a set of image processing techniques that process images based on the shapes and structures within them. These techniques are particularly useful for extracting meaningful information from images, such as identifying objects or analyzing patterns. Morphological operations are based on the principles of mathematical morphology, which is a branch of mathematics that deals with the shapes and structures of objects [36].

Morphological operations are often used in combination with other image processing techniques, such as thresholding or edge detection, to extract and analyze features in an image. They are widely used in a variety of fields, including computer vision, medical imaging, and pattern recognition.

There are two main types of morphological operations: dilation and erosion. Dilation expands the size of an object by adding pixels to its boundaries, while erosion shrinks an object by removing pixels from its boundaries. To better understand these concepts, let's use the following image 4.12 as an example.

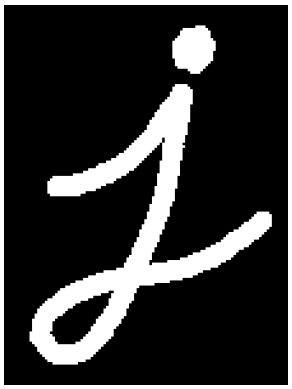

Figure 4.12: Black image with the letter "j" in it [37].

Dilation works by increasing the size of certain elements in an image, called "structuring elements", by adding pixels to their perimeter. These structuring elements can be simple shapes, such as circles or squares, or more complex, such as lines or curves (figure 4.13).

Erosion is a common technique used in image processing to reduce the size of objects or features in an image. It works by applying a small kernel or matrix over the image and setting each pixel to the minimum value within that kernel. This process removes small pixels or features from the image, effectively eroding them away (figure 4.14).

These operations can be combined and modified in various ways to achieve a range of effects, such as closing small holes or removing noise. To remove the noise, a combination of operations is used called opening. It is a technique intended to remove small objects or noise from an image. It is typically used in image processing and computer vision applications to improve the quality of an image (figure 4.15).

The opening operation consists of two steps: erosion and dilation. Erosion removes

Figure 4.13: Image after dilation transformation [37].

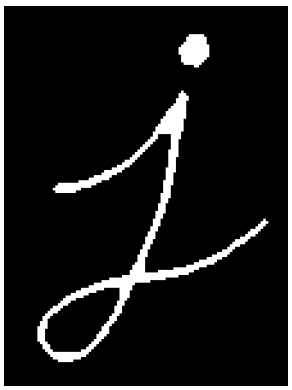

Figure 4.14: Image after erosion transformation [37].

Figure 4.15: Image after opening transformation [37].

small objects or noise from the image by eroding away the pixels around the object. This is done by using a kernel, which is a small matrix of values that is applied to the image. The kernel slides over the image and compares each pixel to its neighbors. If the pixel is surrounded by a majority of pixels with a value of 1 (in a binary image) or 255 (in a greyscale image), then it is retained. If the pixel is surrounded by a majority of pixels with a value of 0, then it is eroded away.

Dilation then follows erosion, and it expands the remaining pixels in the image. This is done by using the same kernel as erosion, but in this case, the kernel looks for pixels with a value of 1 and expands them to include the surrounding pixels. This helps to restore the size and shape of the objects in the image after they have been eroded.

Opening is a useful technique for removing small objects or noise from an image, as it preserves the larger objects and features while removing the smaller ones. It is often used in combination with other morphological operations, such as closing and thinning, to further improve the quality of an image.

To eliminate the noise, the method is to simply execute the function *cv2.morphologyEx()*, which is a function in the OpenCV library that allows for the application of morphological transformations to images.

The function takes in three main arguments: the source image, the morphological operation to be performed, and the structuring element. The morphological operation can be one of several options, including erosion, dilation, closing, gradient, top hat and the one applied here, opening.

The structuring element is a simple shape, such as a square or a circle, that is used to perform the morphological operation. It is defined by its size and shape. The structuring element used here was a 4×4 kernel of 1s (4.12).

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.12}$$

Overall, *cv2.morphologyEx()* is a powerful tool for image processing and analysis and can be used in a wide range of applications, from object detection and segmentation to image enhancement and restoration.

Following the application of the morphological operation and subsequent noise removal, the algorithm produces a clear image 4.16 in which the events from the dynamic objects are the only events present on the frame, eliminating the presence of salt-and-pepper noise.

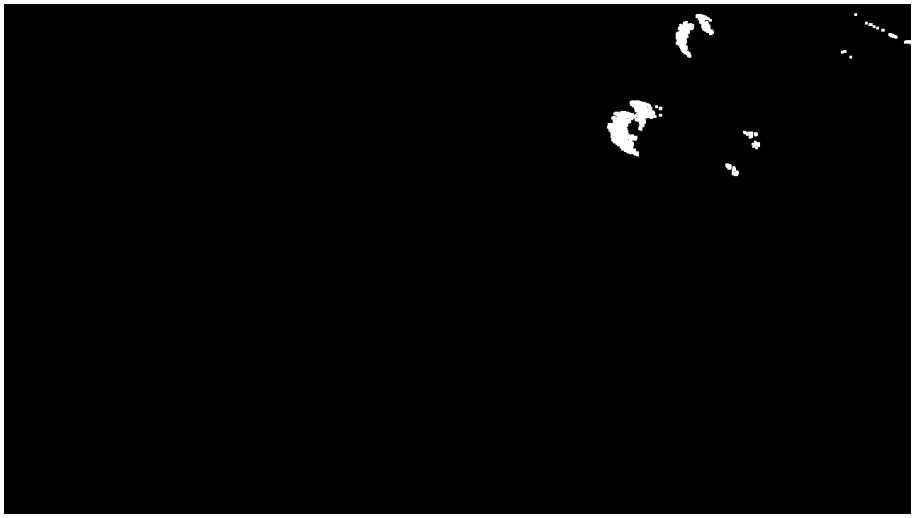

Figure 4.16: The result of the morphological operation – opening.

It has been previously noted that the scene contains three dynamic objects and a moving hand. Therefore, it is important that the algorithm is able to distinguish between these four dynamic bodies in order to detect potential collisions with the UAVs.

4.5 Clustering

Clustering is a widely used data analysis technique that involves grouping data points into distinct categories or clusters based on shared characteristics or patterns [1]. An illustration of the concept being discussed is presented in the accompanying figure 4.17. This technique is commonly used in a variety of fields, including marketing, finance, and biology, to identify patterns and trends in large datasets. In the field of image analysis and computer vision, clustering can be used to detect objects in a dataset by identifying clusters of data points that represent distinct elements. Clustering is particularly useful for detecting objects in datasets with high levels of noise or missing data, as it is able to handle these issues and identify clusters of data points that represent objects. It is also useful for objects that may be difficult to detect using other methods, such as the ones that are partially occluded or have irregular shapes. There are a variety of clustering algorithms available, including K-means and DBSCAN, each with its own strengths and limitations.

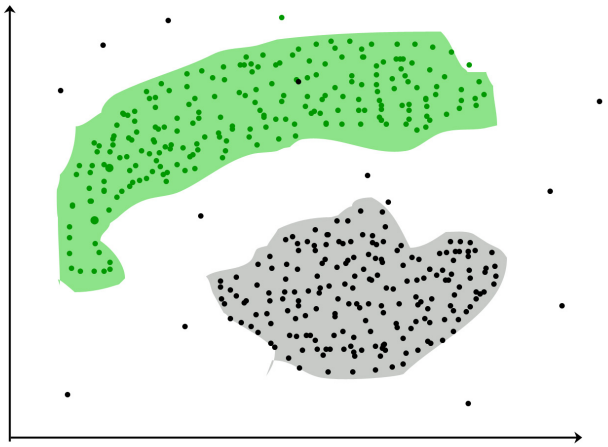

Figure 4.17: Clustering – assigning data points to specific categories or groups based on shared characteristics or patterns [9].

After conducting a thorough evaluation of various clustering algorithms, it was determined that Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [16] was the most suitable method for the object detection task in this case. This decision was based on the following advantages of DBSCAN:

1. Ability to identify clusters of data points representing objects with irregular shapes or partial occlusion: DBSCAN's density-based approach allows it to identify clusters of data points that may be difficult to detect using other methods, such as K-means, which are more sensitive to the shape and distribution of the data.

2. No need to specify the number of clusters: DBSCAN does not require the user to specify the number of clusters beforehand, making it particularly useful for datasets with an unknown number of clusters or when the number of clusters is expected to vary. This allows for more flexibility and adaptability in the clustering process.

Ultimately, the use of DBSCAN for object detection was deemed to be the most appropriate based on its ability to handle irregularly shaped and partially occluded objects and its lack of a requirement for the user to specify the number of clusters beforehand.

DBSCAN is a popular method for detecting objects in a dataset, particularly in image analysis and computer vision applications. It is an unsupervised machine learning algorithm that is used to identify clusters of data points based on shared characteristics or patterns.

The DBSCAN algorithm works by identifying dense regions of data points, known as core points, and then expanding these clusters to include nearby points that meet a certain density threshold. Points that do not meet this threshold or are not connected to a cluster are classified as noise (figure 4.18).

There are two key parameters in DBSCAN -– *epsilon* and *minPts*. *epsilon* determines the radius around each data point in which other points are considered part of the same cluster. *MinPts* determines the minimum number of points required within this radius to form a cluster. These parameters can be adjusted to achieve the desired level of granularity in the clusters.

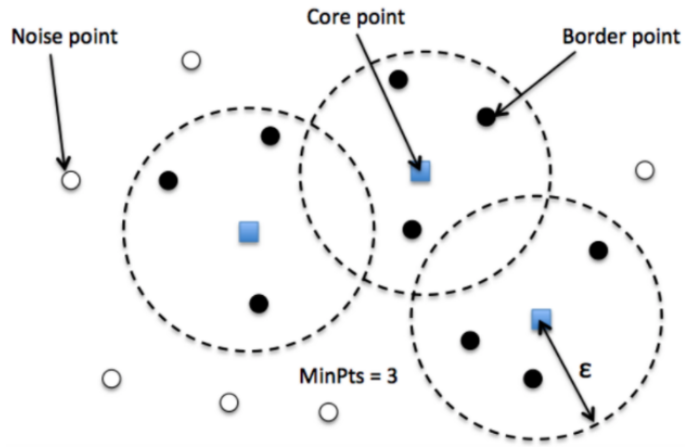

Figure 4.18: Density-Based Spatial Clustering of Applications with Noise [13].

To implement the DBSCAN algorithm, the following steps are followed:

1. Select appropriate values for *epsilon* and *minPts*.

2. Identify all data points that have at least *minPts* number of points within their *epsilon* radius. These points are considered core points.

3. For each core point, create a cluster and add all points within its *epsilon* radius to the cluster, including other core points.

4. Repeat step 3 for all points in the cluster until no more points can be added.

5. Repeat steps 2-4 for all remaining data points until all points have been either assigned to a cluster or classified as noise.

The *sklearn.cluster* module [12], which is a part of the popular *scikit-learn* machine learning library in Python, was utilized to perform the steps of the DBSCAN algorithm. This module contains a range of clustering algorithms, including DBSCAN.

To implement DBSCAN using *sklearn.cluster*, the following steps can be followed:

1. Import the DBSCAN class from *sklearn.cluster*.

2. Initialize a DBSCAN object with the desired values for epsilon and MinPts.

3. Fit the DBSCAN object to the dataset using the fit method.

4. Use the fit object's labels attribute to access the clusters and the fit object's core sample indices attribute to access the core points.

A visual representation of the implemented code for the DBSCAN algorithm is provided in 4.19. Through a process of experimentation and evaluation, it was determined that a value of 2 for epsilon and 50 for MinPts resulted in the best performance.

```
from sklearn.cluster import DBSCAN

# Initialize DBSCAN object with epsilon=50 and minPts=2
dbscan = DBSCAN(eps=50, min_samples=2)

# Fit DBSCAN to the dataset
dbscan.fit(X)

# Access the clusters and core points
clusters = dbscan.labels_
core_points = dbscan.core_sample_indices_
```

Figure 4.19: Example of the implemented code for the DBSCAN algorithm.

sklearn.cluster also includes a number of other clustering algorithms, such as K-means and agglomerative clustering, which can be used depending on the characteristics of the dataset and the desired level of granularity in the clusters.

Once the clusters have been identified using the DBSCAN algorithm, the next step is to determine the boundaries of each one. This can be achieved by fitting a bounding box around each cluster, with the center of the box representing its center. The coordinates of the center of each cluster can then be output as the result of the algorithm.

In this particular example, there are a total of 4 clusters identified by the DBSCAN algorithm 4.20.

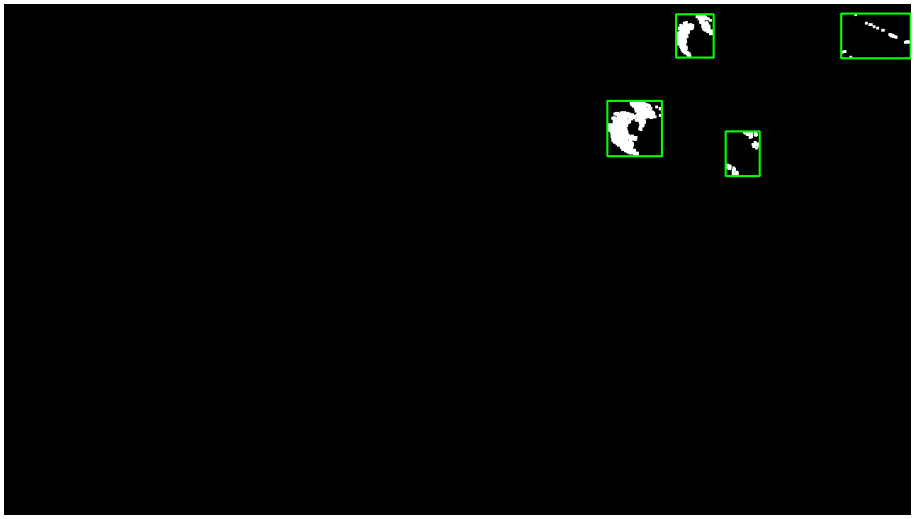

Figure 4.20: Result of the clustering in the frame.

The position of the center of each cluster can be obtained by calculating the mean coordinates of the corners of the bounding box. This method involves computing the sum of the x and y coordinates of each corner and dividing the result by the total number of corners. By doing so, one can obtain an accurate estimate of the center of the cluster 4.21.

```
Center (height,width) of the Moving Object 1 : ( 44.0 , 1230.0 )
Center (height,width) of the Moving Object 2 : ( 44.0 , 974.0 )
Center (height,width) of the Moving Object 3 : ( 175.0 , 889.0 )
Center (height,width) of the Moving Object 4 : ( 210.0 , 1042.0 )
```

Figure 4.21: Illustration of the output of the coordinates of the center of each cluster corresponding to a dynamic object.

5

Experimental results

This chapter presents the experimental results of a proposed collision detection system for UAVs using event cameras. The system's performance is evaluated through a series of experiments conducted in various scenarios and is compared with other algorithms previously discussed in the state of the art. The chapter also provides an explanation of the technical choices that were made in the proposed system.

The evaluation experiments and results for this algorithm were conducted on a system equipped with an Intel Core i7-7700HQ processor, clocked at 2.80 GHz. The operating system utilized was Windows 10 Home 64-bit. The system also featured 16 GB of RAM and an NVIDIA GeForce GTX 1050 graphics card.

5.1 Success rate

To evaluate the effectiveness of the collision detection system, a series of experiments were conducted using the proposed algorithm.

A new function was created as a derivative of the algorithm to analyze the entire sequence. The analysis involved iterating through the sequence at 10 millisecond intervals, starting from the beginning and continuing to the end. This process was repeated for all 9 recorded sequences, resulting in the analysis of over 3000 event-based windows containing diverse dynamic objects.

The success rate for all 9 sequences, as well as the total success rate, was calculated using two indicators, as presented in table 5.1. The first indicator expressed the success rate of the algorithm in detecting the area where the dynamic objects are (Detection area), and the second indicator, which is less detailed, was used to evaluate the algorithm not only in detecting the area where the dynamic objects are but also their precise number in the scene (Each object). This was done to keep the algorithm simple, as equating the success rates of the two instants would make the algorithm more complex, less efficient, and more expensive. These results provide insight into the effectiveness of the proposed system in detecting dynamic objects and their locations in the scene in real time.

For example, the clustering process may detect two objects very close to each other

Success rate		
Sequence	Detection area [%]	Each object [%]
Dynamic	100,00	98,49
Dynamic2	100,00	98,41
Dynamic3	99,34	98,67
Dynamic Camera	100,00	100,00
Static	100,00	100,00
Dynamic Object	100,00	94,09
Dynamic Object2	96,75	91,87
Dynamic Object3	100,00	100,00
Dynamic Object4	99,51	99,51
Total	99,51	97,89

Table 5.1: Success rates for all 9 sequences

when in reality there is only one, and this would not complicate the success of the avoidance algorithm, as it only needs to know the area where there the dynamic objects are to track their path and, consequently, examine if there is a danger of collision with the UAVs.

As seen in 5.1, the first image depicts the scene with all the events, and it is evident that there is only one dynamic object. However, upon examination of the last picture, it becomes clear that the algorithm has identified two objects in the same area where in fact there is only one moving.

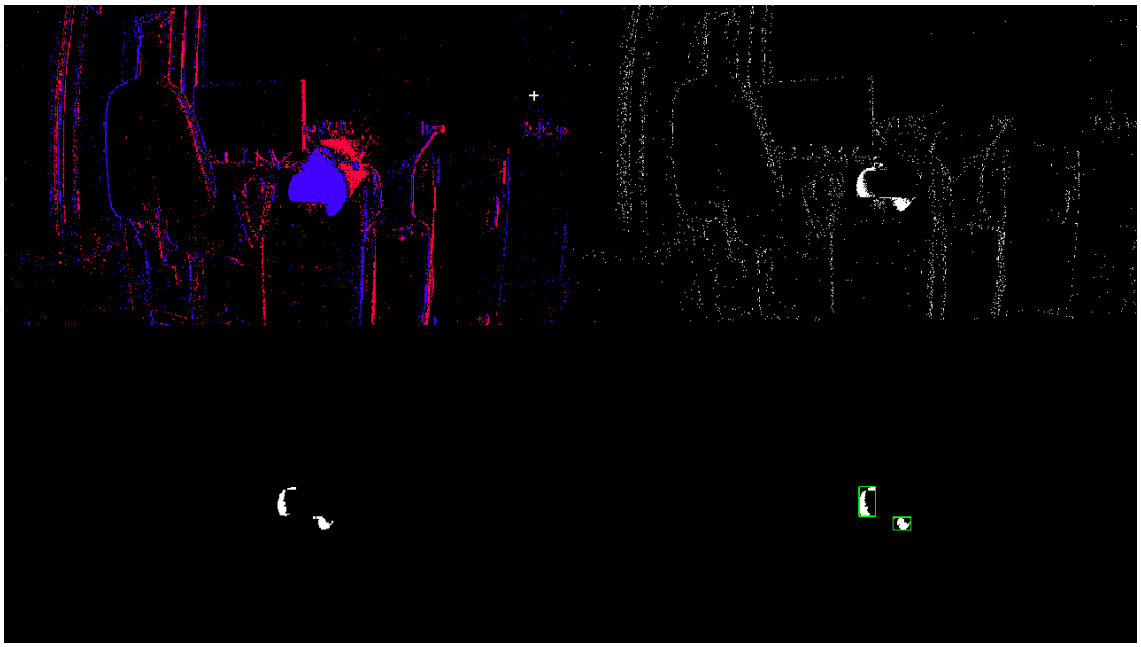

Figure 5.1: Display of the four phases to detect dynamic objects.

5.2 Computational cost

In order to accurately determine the computational cost of the algorithm under examination, it will be necessary to evaluate both the processing time and memory usage of each individual function. This can be achieved through the utilization of a profiler specifically designed for the Python programming language. The profiler provides detailed information on the performance of each function, including the amount of time required for execution and the amount of memory needed for it. This information can then be used to identify and optimize any areas of the algorithm that may be causing excessive computational cost. By utilizing this approach, it is possible to accurately determine the overall computational cost of the algorithm and make any necessary adjustments to improve performance.

The Python programming language includes a built-in profiler called *cProfile* [51], which can be used to measure the performance of code and identify any potential bottlenecks. The profiler can be invoked by calling the *cProfile* module and passing the script to be profiled as an argument. It will then generate a report that includes statistics on the execution time and memory usage of each function in the script.

The *cProfile* module provides several options for controlling the output of the profiler, including the ability to sort the report by different metrics such as total time or cumulative time. Additionally, it is also possible to specify a certain number of function calls to the profile, which is useful for targeting specific sections of code.

The profiler report generated by *cProfile* provides a wealth of information on the performance of the code. For example, it shows the number of times a function was called, the amount of time spent inside, the percentage of the total execution time that was spent in it, the amount of memory used by the function and the number of primitive function calls that were made. Additionally, it also provides the ability to view callers of a function. The graph 5.2 illustrates the average processing time in milliseconds for each primary function. It is evident that the ego-motion function requires the most time, taking approximately one second to process a 10-millisecond time window.

In addition to the built-in profilers, there are also several third-party ones available for Python such as *line-profiler* [30] and *memory-profiler* [34]. They can provide more detailed information on the performance of code and can be useful for identifying specific areas of an application that may be causing performance issues. For example, *line-profiler* can be used to measure the performance of each line of code and *memory-profiler* can be used to measure the memory usage of a script. So the latter was used to determine the memory usage of each function.

memory-profiler is a third-party Python package that allows developers to measure the memory usage of their code. It can be used to identify leaks, track the memory usage of specific functions and understand how the usage changes over time. The package then generates a report that shows the memory usage of each function, including the amount that was allocated (e.g. blocks) and freed. The report can also be used to track how

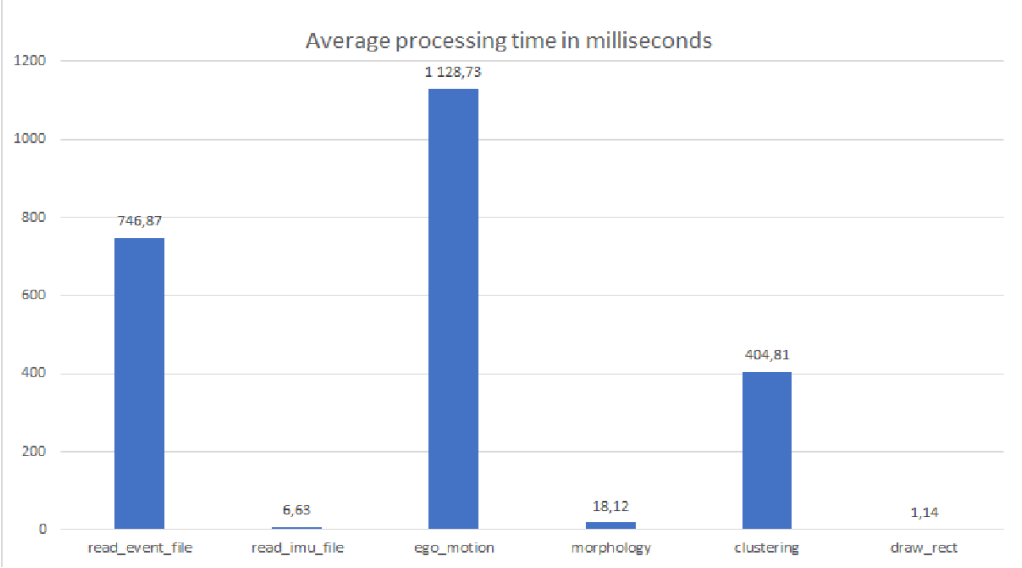

Figure 5.2: Average processing time of the functions in the algorithm.

memory usage changes over time and identify the specific lines of code that are causing the leaks.

The memory package indicates that the majority of the functions do not significantly impact the algorithm's memory usage, with the exception of the *read_event_file* function. As expected, this function's memory usage increases proportionally with the number of events extracted from the dataset. On average, the algorithm's requirement is approximately 800 mebibytes (MiB).

5.3 Comparative analysis

In this section, the algorithm is analyzed in terms of any modifications made to it and the reasoning behind those decisions.

The ego-motion function is a crucial aspect of the algorithm, however, it is also the most time-consuming one. In order to improve efficiency, a different approach was proposed to handle situations where the new positions of events, after the ego-motion, fall outside the image limits. Instead of transposing the new positions to the nearest pixel within the image limits, as it was mentioned in the "Methodology" chapter (4) and "Ego-motion compensation" sub-chapter (4.2), the decision was made to simply ignore events that fall outside the image limits. This decision was made with caution as it was feared that it would negatively impact the performance and success rate of the algorithm, as each event belonging to a dynamic object is an important piece of information.

However, after analyzing results from the sequence "Dynamic 2", it was found that the success rate did not change when evaluating the sequence with this modification. Furthermore, the processing time of the ego-motion decreased when ignoring events outside the image limits. On average, the decrease was 100 ms. As the average processing

time of the ego-motion is around 1 second, this modification improved efficiency by 10%.

Figure 5.3 illustrates the processing time of the ego-motion function for over 60 time windows (10 ms each) of the sequence "Dynamic 2". As expected, it was found that as the number of events increases, the processing time of the ego-motion increases in a linear manner. The blue line represents the processing time of the ego-motion when transposing new events to the nearest pixel, while the orange line represents the more efficient approach of ignoring those events.

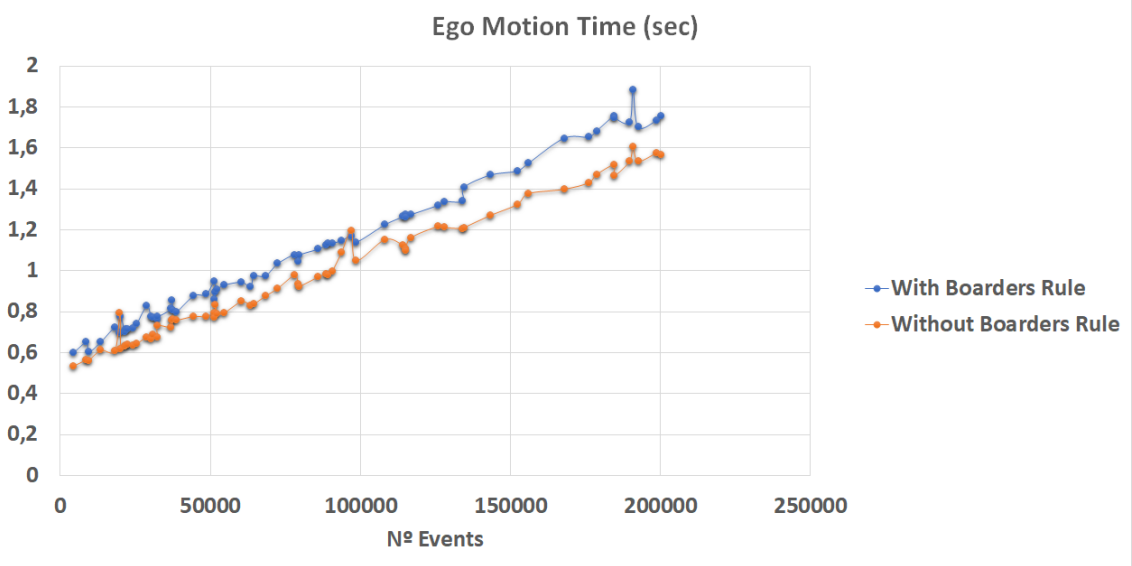

Figure 5.3: Comparison of the processing time of the ego-motion with two different approaches.

A 10 ms time window provides a balance between accuracy and responsiveness, making it a reasonable compromise for many scenarios. In this context, the paragraph explores the reasons why a 10 ms time window is often considered the best option for collision detection algorithms using event cameras for UAVs.

Upon analyzing the average processing time in seconds for time windows of 1, 5, 10, 50, and 100 ms in the sequence "Dynamic 2", it becomes evident that the larger the time window, the longer the processing time, and this occurs in a linear manner 5.4. It is essential to select a smaller time window as an excessively long time window generates an excessive number of events to process, thereby slowing down the algorithm without providing any significant additional value, since the additional events are generated by past obstacle motion history. However, choosing a too-short time window results in insufficient information being collected for reliable detection. In an experiment conducted using time windows of 1 and 5 ms to run the detection algorithm, no object was detected.

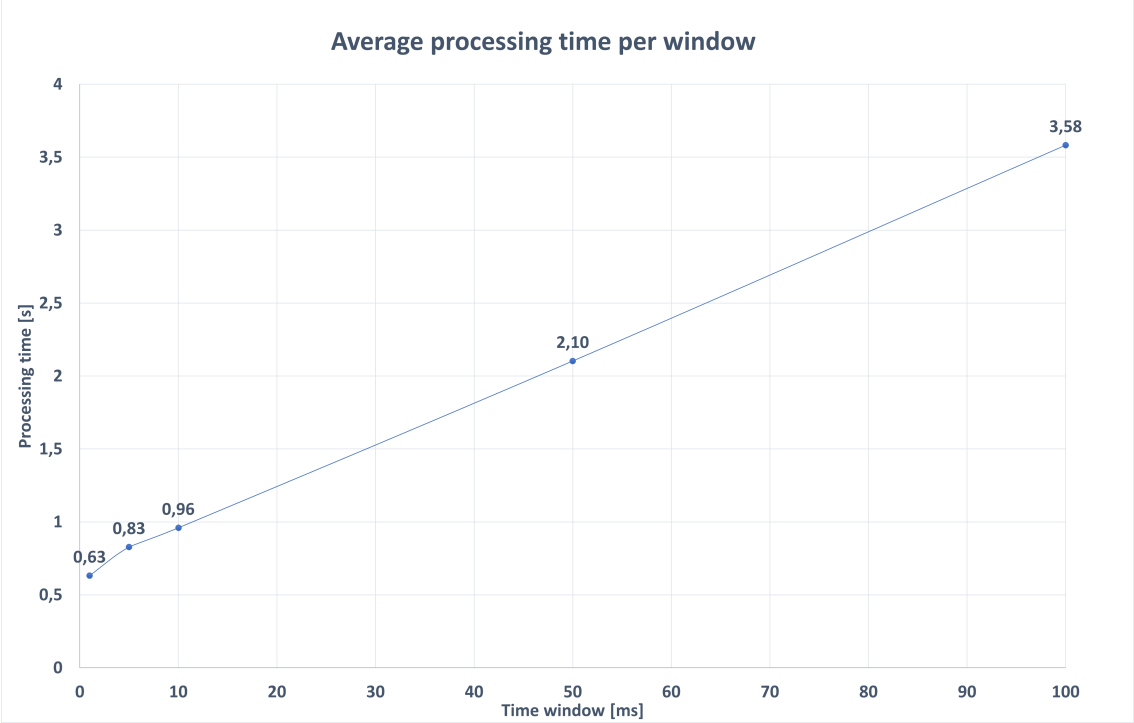

Figure 5.4: Average processing time of the algorithm per time window.

6

Conclusion and Future Work

6.1 Conclusion

In conclusion, this dissertation presents a comprehensive investigation into the collision detection problem for Unmanned Aerial Vehicles using event cameras. The primary objective of this research was to address the limitations of traditional cameras, such as high computational requirements and low frame rates, by exploring the potential of event cameras in enhancing the performance of UAVs collision detection.

The research methodology involved developing an event-based collision detection algorithm that incorporates IMU information for motion compensation. The proposed algorithm was then evaluated using real-world datasets, and the results demonstrated its superior performance compared to traditional methods in terms of accuracy, speed, and robustness to changing lighting conditions and object motion.

Furthermore, this study identified the potential applications of event cameras in enhancing the autonomy of UAVs by enabling them to navigate complex environments and avoid collisions with obstacles in real-time. The proposed algorithm can also be extended to other applications such as robotics, autonomous vehicles, and virtual reality.

One of the unique aspects of this thesis is that it presents, to the best of the knowledge of the author, the only collision detection algorithm for event cameras developed in Python that incorporates IMU information. The proposed algorithm and all datasets used in this study are publicly accessible for future comparisons and research, as mentioned in the "Supplementary material" sub-chapter (1.2). This will allow other researchers to build upon the proposed algorithm and enhance its capabilities for specific applications.

In addition, this thesis emphasizes the importance of making research results openly available to the scientific community. The proposed algorithm and datasets will be published online, allowing researchers to reproduce and expand on this work.

Overall, the findings of this dissertation contribute to the advancement of collision detection for UAVs and provide insights into the potential of event cameras in enhancing the performance of robotics and autonomous systems. Future research can focus on optimizing the proposed algorithm for specific applications and exploring the use of

event cameras in other areas of computer vision and machine learning. The availability of the proposed algorithm and datasets will further facilitate research in this area, leading to the development of more advanced collision detection methods and ultimately improving the safety of UAVs operations.

6.2 Future work

In this chapter, the limitations of the current system are identified and potential directions for further development are suggested. Additionally, challenges and opportunities in the field are highlighted, along with recommendations for future research. The chapter aims to provide a roadmap for researchers and engineers to advance the technology and create safer and more efficient UAVs.

6.2.1 Optimization

When it comes to improving an algorithm's performance, various parameters can be tweaked to achieve better results. The choices made in parameter selection can have a significant impact on the effectiveness of the algorithm. The next segment explores the various parameters that can be modified and the techniques that can be employed to optimize them, with the ultimate goal of creating more efficient and accurate algorithms.

6.2.1.1 Adaptive time window

The time window is a crucial factor in collision detection algorithms. While a 10 ms time window has been effective in many scenarios, it may not be optimal for all situations. The complexity of the scene and the number of dynamic objects can impact the algorithm's performance. Hence, an adaptive time window algorithm that adjusts itself based on the scene and the number of dynamic objects can be proposed to enhance the algorithm's accuracy and responsiveness.

The proposed adaptive time window algorithm could begin with a 10 ms fixed value that provides a good balance between accuracy and responsiveness. However, the algorithm would adjusts the time window dynamically as the scene and number of dynamic objects change. This adjustment could be done using computer vision techniques and machine learning algorithms that analyze the scene and identify the number of dynamic objects. If the scene is more complex, the algorithm would reduce the time window to ensure sufficient time for obstacle detection. Conversely, if the scene is less complex, the algorithm would increase the time window to improve accuracy and reduce false alarms.

In a cluttered environment with many obstacles, the adaptive time window algorithm would reduce the number of false alarms and improve the UAVs' maneuverability. In contrast, in a more open environment with fewer obstacles, the adaptive time window algorithm would increase obstacle detection accuracy and reduce the UAVs' computational load.

6.2.1.2 Clustering analysis

An additional aspect that would be worthwhile to investigate is whether the clustering algorithm utilized, DBSCAN, is the most effective one for this task, by conducting a thorough comparison and analysis.

Other clustering algorithms that can also be well-suited for object detection with event cameras are mentioned below:

1. Ordering Points To Identify the Clustering Structure (OPTICS): This is a density-based clustering algorithm that can handle noise and outliers well. OPTICS can also detect clusters of different sizes and shapes, which can be useful for object detection with event cameras [49].

2. Mean-Shift: This is a non-parametric clustering algorithm that can identify clusters by locating the modes of a density function. Mean-Shift can be used to detect objects in event camera data by finding the modes of the event density in space and time [48].

3. K-Means: This is a popular clustering algorithm that can group events into K clusters based on their similarity in space and time. K-Means can be useful for object detection with event cameras if the number of objects in the scene is known beforehand [47].

The choice of an appropriate clustering algorithm is determined by the specific requirements of the object detection task, along with the characteristics of the event camera data. The selection can depend on whether a more accurate or faster algorithm is desired. It is recommended to experiment with different algorithms and parameter configurations to identify the most effective clustering approach for a given application. Hence, it may be beneficial to analyze and modify the clustering algorithm based on the specific requirements of the object detection task and the event camera data.

6.2.1.3 Adaptive threshold

Another potential method for optimizing the algorithm could involve replacing the fixed threshold, which was previously discussed in the "Normalized Mean Timestamp" subchapter (4.3), with a dynamic function that can calculate the optimal threshold value based on various factors such as the type of scene being observed, the quantity of events detected, and the velocity of the camera. By implementing a threshold function that can adapt to these diverse scenarios, the algorithm can improve its accuracy and efficiency in detecting objects with greater consistency and reliability.

6.2.1.4 Filters

The use of filters can help to reduce the number of events, particularly noise, and optimize the algorithm's processing time. Two common filters used in this regard are Activity Filters [46] and Polarity Filters [45].

Activity Filters work by eliminating inactive events that do not have any other similar events occurring in their 8-point neighbourhood. By filtering them out, the number of events being processed is reduced, resulting in a decrease of processing time.

Polarity Filters, on the other hand, work by propagating events of a specific polarity. This means that only positive or negative events are processed, and never both together. As a result, the number of events processed is reduced by approximately 50%. By reducing the number of events being processed, the algorithm can be optimized, resulting in improved performance and reduced processing time.

6.2.1.5 C functions

One possible direction for future work is to use PyObject [10]. This idea has been inspired by the code from the article [35]. PyObject can be used to interface Python with C code, which is known for its speed and efficiency. By migrating certain functions to C, the collision detection algorithm can be processed more efficiently, resulting in faster response times. One function that would benefit from migration to C is the ego-motion function, according to the graph presented in 5.2, which is the most time-consuming in the algorithm. Using PyObject and C functions can also make the code more modular and easier to maintain.

6.2.2 Hardware implementation

Eventually, the algorithm can be implemented in an Field-Programmable Gate Array (FPGA). Implementing a collision detection algorithm using event cameras for UAVs in an FPGA can be beneficial because these platforms offer high processing power and low latency, which are essential for real-time collision detection [24] [15] [8]. Additionally, FPGA-based implementations can be more energy-efficient compared to software-based implementations, because they are designed to perform specific tasks using dedicated hardware, rather than relying on a general-purpose processor to execute instructions.

When executing software-based collision detection algorithms, the processor must execute a large number of instructions, which can consume a significant amount of power. In contrast, FPGAs are optimized for parallel processing, which enables them to perform multiple tasks simultaneously.

For example, reading the data requires sequential execution. Nonetheless, in certain aspects, FPGA parallelization plays a critical role:

1. The algorithm can process multiple time windows simultaneously, which means that it can start processing the next time window without waiting for the current

one to finish.

2. It is also feasible to parallelize the processing of multiple events within a time window. For example, when creating the warped image in the ego-motion function, multiple events can be processed simultaneously.

3. While processing older events for ego-motion and even older events for visualizing the results, new events can be simultaneously read. In contrast, such simultaneous processing is not possible with a Central Processing Unit (CPU).

The ability to process data in parallel reduces latency, which is the time needed to process data from input to output. As a result, this lowers power consumption and enables UAVs, for example, to fly for longer periods of time.

Bibliography

[1] C. C. 1. Aggarwal and C. K. 1. Reddy. *Data Clustering : Algorithms and Applications*. ISBN: 978-1-315-37351-5 (cit. on p. 54).

[2] N. Ahmad et al. "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* (2013), pp. 256–262. ISSN: 23154535. DOI: 10.12720/ijsps.1.2.256-262 (cit. on p. 16).

[3] S. Baker and I. Matthews. *Lucas-Kanade 20 Years On: A Unifying Framework*. 2004, pp. 221–255. URL: http://www.ri.cmu.edu/projects/project (cit. on p. 19).

[4] P. A. Bardow, S. Leutenegger, and P. A. Davison. *Estimating General Motion and Intensity from Event Cameras*. 2018 (cit. on p. 9).

[5] R. Benosman et al. "Event-based visual flow". In: *IEEE Transactions on Neural Networks and Learning Systems* 25 (2 2014-02), pp. 407–417. ISSN: 2162237X. DOI: 10.1109/TNNLS.2013.2273537 (cit. on p. 25).

[6] P. Bernhard and M. Deschamps. "Kalman 1960: The birth of modern system theory". In: *Mathematical Population Studies* 26.3 (2019), pp. 123–145. eprint: https://doi.org/10.1080/08898480.2018.1553393. URL: https://doi.org/10.1080/08898480.2018.1553393 (cit. on p. 19).

[7] D. Borrmann et al. "The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design". In: *3D Research* 2 (2 2011), pp. 1–13. ISSN: 20926731. DOI: 10.1007/3DRes.02(2011)3 (cit. on p. 23).

[8] A. Cardona, L. Garelli, and J. M. Gimenez. *DRONE AUTONOMOUS NAVIGATION BY HARDWARE IMAGE PROCESSING*. 2019, pp. 5–7. URL: http://www.lac.inpe.brhttp://www.ieav.cta.br/://www.amcaonline.org.ar (cit. on p. 68).

[9] *Clustering in Machine Learning - GeeksforGeeks*. Accessed: Oct. 10, 2022. URL: https://www.geeksforgeeks.org/clustering-in-machine-learning/ (cit. on p. 54).

[10] *Common Object Structures — Python 3.11.2 documentation*. Accessed: Mar. 2, 2023. URL: https://docs.python.org/3/c-api/structures.html (cit. on p. 68).

[11] *DAVIS 240* (cit. on p. 40).

[12] *DBSCAN Algorithm Clustering in Python | Engineering Education (EngEd) Program | Section*. Accessed: Oct. 10, 2022. URL: https://www.section.io/engineering-education/dbscan-clustering-in-python/ (cit. on p. 56).

[13] *DBSCAN Clustering Algorithm in Machine Learning - KDnuggets*. Accessed: Oct. 10, 2022. URL: https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html (cit. on p. 55).

[14] D. Detone, T. Malisiewicz, and A. Rabinovich. "Deep Image Homography Estimation". In: () (cit. on p. 26).

[15] S. Ehsan and K. D. Mcdonald-Maier. *On-Board Vision Processing For Small UAVs: Time to Rethink Strategy* (cit. on p. 68).

[16] M. Ester et al. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. 1996. URL: www.aaai.org (cit. on p. 54).

[17] *EVK1 - HD - Product brief*. Accessed: Mar. 2, 2023. URL: https://support.prophesee.ai/portal/en/kb/articles/prophesee-evauation-kit-1-evk-1-gen4-hd#Highlights (cit. on pp. 42, 43).

[18] *Exit seminar - Google Slides*. Accessed: Jun. 1, 2022. URL: https://docs.google.com/presentation/d/18ZKiLXSx8F8pb_8PjALfAECDDfbQ3ux3w_q5mevAwx0/edit#slide=id.g7d90b5d6f1_1_35 (cit. on p. 6).

[19] D. Falanga, S. Kim, and D. Scaramuzza. *How Fast is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid*. 2019. URL: http://youtu.be/sbJAi6SXOQw (cit. on p. 7).

[20] D. Falanga, K. Kleber, and D. Scaramuzza. *Dynamic obstacle avoidance for quadrotors with event cameras*. 2020, p. 9712. URL: https://www.science.org (cit. on pp. 16–18, 21, 22, 45).

[21] *Frame- vs Event-based cameras - the curious case of the spinning dot - YouTube*. Accessed: Jun. 1, 2022. URL: https://www.youtube.com/watch?v=2tLGNFlS7bM (cit. on p. 6).

[22] G. Gallego, H. Rebecq, and D. Scaramuzza. "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation". In: (). URL: https://youtu.be/LauQ6LWTkxM?t=25 (cit. on p. 29).

[23] G. Gallego et al. "Event-Based Vision: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44 (01 2022-01), pp. 154–180. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2020.3008413. URL: http://www.gradoffice.caltech.edu/current/clauser (cit. on p. 6).

[24] P. Hobden, S. Srivastava, and E. Nurellari. "FPGA-Based CNN for Real-Time UAV Tracking and Detection". In: *Frontiers in Space Technologies* 3 (2022-05). DOI: 10.3389/frspt.2022.878010 (cit. on p. 68).

[25] *How Cameras Work | HowStuffWorks*. Accessed: Jun. 1, 2022. URL: https://electronics.howstuffworks.com/camera.htm (cit. on p. 5).

[26] *How to Create Motion Blur in Premiere Pro CC - Motion Array*. Accessed: Jun. 13, 2022. URL: https://motionarray.com/learn/premiere-pro/premiere-pro-motion-blur-tutorial/ (cit. on pp. 9, 10).

[27] *K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint*. Accessed: Jan. 8, 2023. URL: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning (cit. on p. 22).

[28] *Law of Total Probability | Partitions | Formulas*. Accessed: Mar. 2, 2023. URL: https://www.probabilitycourse.com/chapter1/1_4_2_total_probability.php (cit. on p. 37).

[29] *Learn about MEMS accelerometers, gyroscopes, and magnetometers · VectorNav*. Accessed: Mar. 2, 2023. URL: https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-mems (cit. on p. 42).

[30] *line-profiler · PyPI*. Accessed: Mar. 25, 2023. URL: https://pypi.org/project/line-profiler/ (cit. on p. 61).

[31] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[32] A. Ma et al. *Artificial Potential Field Algorithm Implementation for Quadrotor Path Planning*. 2019. URL: www.ijacsa.thesai.org (cit. on p. 20).

[33] J. E. Mebius. *DERIVATION OF THE EULER-RODRIGUES FORMULA FOR THREE-DIMENSIONAL ROTATIONS FROM THE GENERAL FORMULA FOR FOUR-DIMENSIONAL ROTATIONS*. 2007 (cit. on p. 46).

[34] *memory-profiler · PyPI*. Accessed: Mar. 25, 2023. URL: https://pypi.org/project/memory-profiler/ (cit. on p. 61).

[35] A. Mitrokhin et al. "Event-based Moving Object Detection and Tracking". In: (). URL: http://prg.cs.umd.edu/BetterFlow.html (cit. on pp. 27, 28, 39–41, 48, 68).

[36] *Morphological Image Processing*. Accessed: Jan. 8, 2023. URL: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm (cit. on p. 51).

[37] *Morphological Transformations — OpenCV-Python Tutorials beta documentation*. Accessed: Oct. 10, 2022. URL: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html (cit. on pp. 51, 52).

[38] E. Mueggler et al. *Lifetime Estimation of Events from Dynamic Vision Sensors*. URL: http://rpg.ifi.uzh.ch. (cit. on p. 24).

[39] T. Nguyen et al. "Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model". In: () (cit. on p. 26).

[40] D. Nistér, O. Naroditsky, and J. Bergen. "Visual odometry". In: vol. 1. 2004. DOI: 10.1109/cvpr.2004.1315094 (cit. on p. 14).

[41] E. Piña. "Rotations with Rodrigues' vector". In: *European Journal of Physics* 32 (5 2011-09), pp. 1171–1178. ISSN: 01430807. DOI: 10.1088/0143-0807/32/5/005 (cit. on p. 46).

[42] H. Rebecq, D. Gehrig, and D. Scaramuzza. "ESIM: an Open Event Camera Simulator". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by A. Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018-29–31 Oct, pp. 969–982. URL: https://proceedings.mlr.press/v87/rebecq18a.html (cit. on p. 27).

[43] N. J. Sanket et al. "EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras". In: (2019-06). URL: http://arxiv.org/abs/1906.02919 (cit. on pp. 25–27).

[44] D. Scaramuzza. "Tutorial on Event-based Cameras". In: (). URL: http://rpg.ifi.uzh.ch/research_dvs.html (cit. on pp. 6, 10).

[45] *SDK Core Algorithms — Metavision Intelligence Docs 3.1.2 documentation*. Accessed: Mar. 16, 2023. URL: https://docs.prophesee.ai/stable/metavision_sdk/modules/core/api/algorithms.html (cit. on p. 68).

[46] *SDK CV Algorithms — Metavision Intelligence Docs 3.1.2 documentation*. Accessed: Mar. 16, 2023. URL: https://docs.prophesee.ai/stable/metavision_sdk/modules/cv/api/algorithms.html (cit. on p. 68).

[47] *sklearn.cluster.KMeans — scikit-learn 1.2.2 documentation*. Accessed: Mar. 2, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (cit. on p. 67).

[48] *sklearn.cluster.MeanShift — scikit-learn 1.2.2 documentation*. Accessed: Mar. 2, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html (cit. on p. 67).

[49] *sklearn.cluster.OPTICS — scikit-learn 1.2.2 documentation*. Accessed: Mar. 2, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html (cit. on p. 67).

[50] T. Stoffregen et al. "Event-Based Motion Segmentation by Motion Compensation". In: (). URL: https://youtu.be/0q6ap (cit. on pp. 30, 32).

[51] *The Python Profilers — Python 3.11.2 documentation*. Accessed: Mar. 25, 2023. URL: https://docs.python.org/3/library/profile.html (cit. on p. 61).

[52] D. R. Valeiras et al. "Neuromorphic event-based 3D pose estimation". In: *Frontiers in Neuroscience* 9 (JAN 2016). ISSN: 1662453X. DOI: 10.3389/fnins.2015.00522 (cit. on p. 19).

[53] V. Vasco, A. Glover, and C. Bartolozzi. *Fast Event-based Harris Corner Detection Exploiting the Advantages of Event-driven Cameras* (cit. on p. 24).

[54] Y. Wan et al. "Motion compensation and object detection for neuromorphic camera". In: *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2021, pp. 3420–3427. DOI: 10.1109/BIBM52615.2021.9669816 (cit. on pp. 35, 48).

[55] J. Y. A. Wang and E. H. Adelson. "Layered Representation for Motion Analysis". In: (1993), pp. 361–366 (cit. on p. 31).

[56] *What is Computer Vision? | IBM*. Accessed: Jun. 1, 2022. URL: https://www.ibm.com/topics/computer-vision (cit. on p. 5).

[57] *What is data redundancy in computer vision?* Accessed: Jun. 13, 2022. URL: https://www.lightly.ai/post/what-is-data-redundancy-in-computer-vision (cit. on p. 9).

[58] *What is HDR (high dynamic range) in photography? - Adobe*. Accessed: Jun. 13, 2022. URL: https://www.adobe.com/creativecloud/photography/hub/guides/what-is-hdr-photography (cit. on p. 8).

[59] *What is low latency camera streaming? – e-con Systems*. Accessed: Jun. 13, 2022. URL: https://www.e-consystems.com/blog/camera/technology/why-low-latency-streaming-is-an-important-feature-for-many-embedded-vision-applications/ (cit. on p. 8).

[60] L. Yang. "Ego-motion Estimation Based on Fusion of Images and Events". In: () (cit. on pp. 33, 34).

[61] J. N. Yasin et al. "Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches". In: *IEEE Access* 8 (2020), pp. 105139–105155. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3000064 (cit. on pp. 11–13, 21).