DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
ESCUELA SUPERIOR DE INGENIERÍA
UNIVERSIDAD DE SEVILLA

Event-based Perception for Aerial Robots: From Multirotors to Ornithopters

por

Juan Pablo Rodríguez Gómez

Graduado en Ingeniería Electrónica
Master in Artificial Intelligence and Robotics

PROPUESTA DE TESIS DOCTORAL
PARA LA OBTENCIÓN DEL TÍTULO DE

DOCTOR POR LA UNIVERSIDAD DE SEVILLA
SEVILLA, 2022

Directores
Dr.Ing. José Ramiro Martínez de Dios, Catedrático
Dr.Ing. Aníbal Ollero Baturone, Catedrático

UNIVERSIDAD DE SEVILLA

Memoria para optar al grado de Doctor por la Universidad de Sevilla

Autor:	**Juan Pablo Rodríguez Gómez**
Título:	**Event-based Perception for Aerial Robots: From Multirotors to Ornithopters**
Departamento:	**Departamento de Ingeniería de Sistemas y Automática**

V° B° Director:

José Ramiro Martínez de Dios

V° B° Director:

Aníbal Ollero Baturone

El autor:

Juan Pablo Rodríguez Gómez

A Francisca, abbiamo fatto insieme ogni passo di questo lungo percorso.

A mi mami, su inmenso amor me ha llevado a cumplir mis sueños. He perdido a mi héroe en el mundo, pero he ganado un ángel en el cielo.

A mi padre por apoyarme y aconsejarme en cada uno de mis proyectos.

Acknowledgements

I would like to thank my supervisor, Prof. J. Ramiro Martínez, for his advice and guidance. He selected me as his Ph.D. student and brought me to Seville to join this excellent group. Thank you also to Prof. Aníbal Ollero for the opportunity to join the GRVC Robotics Laboratory. It has been an honor to work at your side and learn from your broad experience in robotics.

At the GRVC, I collaborated with amazing people; without their help this Ph.D. Thesis would not have been possible. I want to express my gratitude to the members of this team. In particular, I would like to thank my colleagues Augusto Gómez and Raúl Tapia. We spent days and nights together designing and validating our event-based vision algorithms for ornithopter robots. It was a pleasure to work with the two of you. To the current and past members of the GRIFFIN team, thanks for the great time we had while working with flapping-wing robots. Special thanks to Rafael Salmoral for flying at any time, under any weather conditions, and in any circumstance.

I am very grateful to Prof. Guillermo Gallego and the members of the RIP team. It was a pleasure to learn from all of you about event-based vision. My time at TU-Berlin became one of my most valuable career experiences as a researcher.

To all my dear friends for making this journey happier and funnier. Ana, Araceli, Manu, David, Edo, Raph, Lisa, and many others, thanks!

I am very grateful to my father for his patience, support, and love.

To my wife, Francisca. It has been a long adventure that we underwent together, and this journey has now come to its end, my love. I cannot thank you enough.

Finally, I would like to dedicate this work to my mother. She was an inspiration to me. Her love and strength remain with me now.

Juan Pablo Rodríguez Gómez

This work has been supported by the Spanish National Prueba de Concepto R&D Programme ROBMIND (PDC2021-121524-I00) and HAERA (PID2020-119027RB-I00) funded by the Spanish Ministerio de Ciencia e Innovación, and by the EUROPEAN H2020 project AERIAL-CORE (H2020-2019-871479) and the ERC Advanced Grant GRIFFIN Project (ERC-2017-ADG-788247) funded by the European Commission.

Resumen

Esta Tesis Doctoral tiene como objetivo explorar las ventajas ofrecidas por las cámaras de eventos en la robótica aérea, en particular, en robots de ala batiente en los que la visión basada en eventos se encuentra prácticamente inexplorada. Esta Tesis Doctoral propone un conjunto de herramientas y algoritmos de percepción para aprovechar las ventajas de la visión basada en eventos en aplicaciones de robótica aérea. Los métodos propuestos se validan experimentalmente en dos tipos de robots aéreos: multirotores y ornitópteros. Estas plataformas suponen varios desafíos para la percepción basada en visión artificial. Por ejemplo, el movimiento ágil de los multirotores y los aleteos causados por los ornitópteros generan imágenes con desenfoque por movimiento (o en ingles *motion blur*) que pueden afectar al rendimiento de algoritmos de percepción basados en imágenes. Adicionalmente, los ornitópteros y multirrotores que realizan maniobras rápidas requieren algoritmos de percepción que actualicen rápidamente la información usada para la navegación del robot. Además, ambos tipos de plataformas tienen carga útil y potencia reducidas lo que limita el tipo y la cantidad de hardware de percepción a bordo. Equipar ornitópteros con sensores de percepción es una tarea compleja, ya que estas plataformas tienen una carga útil muy restringida y una distribución estricta de peso. Las cámaras de eventos ofrecen varias ventajas para la percepción en robótica: píxeles con resolución de microsegundos, robustez al desenfoque por movimiento, alto rango dinámico y bajo consumo de energía. Esta Tesis Doctoral se enfoca en aprovechar estas ventajas para el desarrollo de sistemas de percepción para robots aéreos y validar su uso a bordo de estas plataformas.

Primero, esta Tesis Doctoral presenta un conjunto de algoritmos de procesamiento de eventos de bajo nivel. Estos métodos tienen como objetivo contribuir a la comunidad de visión basada en eventos proporcionando un conjunto de algoritmos que procesan directamente el flujo de eventos en lugar de utilizar representaciones adicionales basadas en imágenes. Los métodos desarrollados se integran en los diferentes algoritmos de percepción de alto nivel descritos a lo largo de esta Tesis Doctoral.

En segundo lugar, esta Tesis Doctoral propone un método de guiado basado en eventos para robots aéreos. El método incluye un conjunto de algoritmos para detectar y seguir un patrón de referencia que define la configuración objetivo. Un método de control por realimentación visual basado en eventos calcula los comandos de velocidad para guiar al robot hacia el objetivo. El esquema de guiado es validado inicialmente en un multirotor y posteriormente en un ornitóptero.

Tercero, esta Tesis Doctoral presenta un método de evitación de obstaculos para robots de ala batiente. El método detecta obstáculos dinámicos aprovechando la habilidad de las cámaras de eventos para proporcionar información sobre los objetos en movimiento en la escena. Este algoritmo también utiliza una estrategia de evitación de obstáculos reactiva que evalúa posibles situaciones de riesgo de colisión y activa maniobras evasivas si es necesario. El sistema se valida experimentalmente en un robot ornitóptero.

En cuarto lugar, esta Tesis Doctoral propone un sistema de monitorización de intrusos basado en eventos para multirotores. El método incluye un algoritmo de detección de intrusos y un método que permite ajustar de manera autónoma los parámetros de dicho algoritmo. El sistema es validado en un multirotor que realiza misiones de vigilancia en escenarios con diferentes fondos y condiciones de iluminación.

Finalmente, esta Tesis Doctoral presenta dos herramientas de percepción para el desarrollo de algoritmos de visión basados en eventos para robots aéreos, especialmente ornitópteros. La primera es una arquitectura de simulación que emula las medidas de los sensores de percepción generadas durante la ejecución de trayectorias bioinspiradas de aterrizaje. La segunda herramienta describe un *dataset* grabado a bordo de un robot de ala batiente, que incluye medidas de varios sensores de percepción y datos de la posición del robot.

Abstract

This Ph.D. Thesis aims at contributing to the robot perception community by exploring the advantages of event cameras for aerial robotics, in particular on flapping-wing robots in which event-based vision is almost unexplored. It proposes a set of tools and perception algorithms to leverage the advantages of event-based vision in aerial robot applications. The proposed methods are experimentally validated in two types of aerial robots: multirotors and ornithopters. These platforms describe several challenges for vision-based perception. For instance, the agile motion of multirotors and the flapping strokes caused by ornithopters generate blurred images which may hinder the performance of frame-based approaches. Further, ornithopters and multirotors performing fast maneuvers require quick perception algorithms to update as soon as possible the perception information for robot navigation. Besides, both types of robots have limited payload and power capacity to mount and feed perception hardware. In particular, equipping ornithopter robots with additional sensors is a complex task as they have very constrained payload and strict weight distribution. Event cameras offer relevant advantages for robot perception such as microsecond pixel resolution, robustness to motion blur, high dynamic range, and low power consumption. This Ph.D. Thesis focuses on leveraging these advantages for aerial robot perception and validating the use of event-based vision on board these platforms.

First, this Ph.D. Thesis presents a set of event-based low-level processing algorithms. These methods intend to contribute to the event-based vision community by providing a set of low-level algorithms that directly process the event stream instead of using frame-based representations. The proposed methods are integrated into the different high-level perception algorithms described in this Ph.D. Thesis.

Second, this Ph.D. Thesis proposes an event-vision guidance method for aerial robots. It includes a set of event-based algorithms to detect and track a reference pattern that defines the goal configuration. An Event-Based Visual Servoing (EBVS) method computes the velocity commands to guide the robot toward the goal. The guidance scheme is initially validated in a multirotor and later in an ornithopter.

Third, a dynamic *sense-and-avoid* method for large-scale flapping-wing robots is described in this Ph.D. Thesis. It exploits the event cameras' property of triggering pixel information from moving objects to detect dynamic obstacles. The reactive avoidance policy evaluates possible collision risk situations and activates evasive maneuvers if necessary. The system is extensively evaluated in a large-scale ornithopter.

Fourth, this Ph.D. Thesis proposes an event-based intrusion monitoring system for multirotors. It includes a specific method to detect moving intrudes by analyzing the spatial-temporal information of events, and an automatic tuning method to adjust the parameters of the detection algorithm. The system is validated in a multirotor platform that performs surveillance missions in scenarios with different background configurations and illumination conditions.

Finally, this Ph.D. Thesis presents two perception tools for the development of event-based algorithms for aerial robots, especially ornithopters. The first tool is a simulation architecture that emulates the sensor measurements generated during the execution of bioinspired landing trajectories. The second tool describes a dataset collected on board a large-scale flapping-wing robot. It includes measurements from different perception sensors and ground truth robot position data.

Acronyms

APS	Active Pixel Sensor
CNN	Convolutional Neural Network
CoG	Center of Gravity
CPU	Central Processing Unit
DoF	Degrees of Freedom
DL	Deep Learning
DVS	Dynamic Vision Sensor
EBVS	Event-Based Visual Servoing
EKF	Extended Kalman Filter
Eq	Equation
FOE	Focus Of Expansion
FoV	Field of View
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GPS	Global Positioning System
GPU	Graphics Processing Unit
IBVS	Image-Based Visual Servoing
IM	Intrusion Monitoring
IMU	Inertial Measurement Unit
ISV	Intersection of the Safety Volumes
KF	Kalman Filter
LED	Light Emitting Diode

LiDAR	Laser imaging Detection and Ranging
LSD	Line Segment Detector
MAE	Mean Absolute Error
MAV	Micro Aerial Vehicle
ML	Machine Learning
No	Number of
NTP	Network Time Protocol
PBVS	Position-Based Visual Servoing
R&D	Research and Development
RGB	Red Green Blue
ROS	Robot Operating System
RTK-GPS	Real Time Kinematics Global Positioning System
SA	Simulated Annealing
SAE	Surface of Active Events
SLAM	Simultaneous Localization And Mapping
TN	True Negative
ToF	Time of Flight
TP	True Positive
TPR	True Positive Rate
TPU	Thermoplastic PolyUrethane
TTC	Time-To-Contact
UART	Universal Asynchronous Receiver-Transmitter
UAS	Unmanned Aerial Systems
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicles
VIO	Visual Inertial Odometry
V-TOL	Vertical Take-Off and Landing

Contents

Acknowledgements **vii**

Resumen **ix**

Abstract **xi**

Acronyms **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 5

1.3 Context 6

1.4 Contributions 8

1.4.1 Contribution 1: Event-vision UAS guidance 8

1.4.2 Contribution 2: Event-based dynamic sense-and-avoid for flapping-wing robots 8

1.4.3 Contribution 3: Event-based intrusion monitoring for multirotors 10

1.4.4 Contribution 4: Simulation tool for bioinspired landing perception 11

1.4.5 Contribution 5: A dataset for flapping-wing robot perception . 11

1.4.6 Contribution 6: Event-based algorithm validation on ornithopters 12

1.4.7 Contribution 7: Event-vision resources 14

1.4.8 Publications 14

1.4.8.1 Journals 14

1.4.8.2 International conferences 15

1.4.8.3 Datasets 16
1.4.8.4 Software 17
1.5 Document structure 17

2 Event-based Low-level Perception 21
2.1 Introduction 21
2.2 Related work 22
2.3 Low-level perception algorithms 25
2.3.1 Asynchronous feature tracking 25
2.3.2 Asynchronous event-based clustering 28
2.3.3 Attention priority map (APM) 29
2.3.4 Time filter 31
2.4 Experiments 32
2.4.1 Tracking 32
2.4.2 Clustering 37
2.4.3 APM and time filter 41
2.5 Conclusions 43

3 Event-based Visual Guidance for Aerial Robots 45
3.1 Introduction 45
3.2 Related work 47
3.3 General scheme 49
3.4 Line detection 51
3.4.1 Event coordinates undistortion 51
3.5 Line tracking 53
3.5.1 Event-by-event line tracker 53
3.5.2 Event-image line tracker 55
3.6 Visual servoing 56
3.6.1 Time-to-contact velocity guidance 58
3.7 Experiments 60
3.7.1 Line tracking 60
3.7.2 Multirotor landing 64

3.7.3 Ornithopter guidance 68
3.8 Conclusions 73

4 Event-based Dynamic Sense-and-Avoid for Ornithopters 77
4.1 Introduction 77
4.2 Related work 78
4.3 General scheme 82
4.4 Event-based dynamic obstacle motion estimation 84
4.5 Reactive collision evaluation strategy 88
4.6 Tail control 90
4.7 Experiments 91
4.7.1 Obstacle detection and motion estimation evaluation 92
4.7.2 *Sense-and-avoid* evaluation 95
4.8 Conclusions 99

5 Event-based Intrusion Monitoring for Multirotors 101
5.1 Introduction 101
5.2 Related work 103
5.3 Intrusion detection 105
5.3.1 Intrusion monitoring algorithm 106
5.3.2 Selective sampling of events 108
5.3.3 Parameter dependencies 109
5.4 Auto-tuning perception scheme for intrusion monitoring 111
5.4.1 General scheme 111
5.4.2 Intrusion monitoring auto-tuning 114
5.4.3 Simulated annealing 114
5.4.4 Parameter auto-tuning 116
5.5 Experiments 117
5.5.1 Intrusion monitoring evaluation 118
5.5.1.1 Robustness validation 120
5.5.1.2 Real-time processing 121
5.5.2 Auto-tuning surveillance scheme validation 122

5.5.2.1 Performance evaluation 125
5.5.2.2 Analysis of zone sensitivity 128
5.5.2.3 Performance under low illumination conditions . . . 130
5.6 Conclusions 132

6 Datasets and Simulation 135
6.1 Introduction 135
6.2 Related work 136
6.3 ROSS-LAN: RObotic Sensing Simulation
scheme for bioinspired robotic bird LANding 140
6.3.1 Tau-theory planning 141
6.3.2 Architecture of the simulator 143
6.3.3 Datasets 145
6.4 The GRIFFIN perception dataset 148
6.4.1 The *Eye-Bird* ornithopter 149
6.4.2 Sensors 151
6.4.3 Ornithopter v.s. quadrotor flight 154
6.4.4 The GRIFFIN perception datasets 155
6.4.5 Validation 159
6.5 Conclusions 160

7 Conclusions and Future Work 163
7.1 Conclusions 163
7.2 Future work 166

References 171

List of Figures

1.1 An outdoor guidance validation experiment using the GRIFFIN *E-Flap* ornithopter. 9

1.2 The GRIFFIN *E-Flap* ornithopter robot performing a *sense-and-avoid* maneuver. 9

1.3 Intrusion monitoring results on board a quadrotor platform. 10

1.4 Block diagram of the ROSS-LAN simulation tool. 11

1.5 A dataset recording experiment using the GRIFFIN *Eye-Bird* ornithopter. 12

1.6 One of the GRIFFIN ornithopters designed and built at the GRVC Robotics Laboratory. 13

1.7 Block diagram depicting the relationship between the chapters (Ch in the diagram) of this Ph.D. Thesis. 18

2.1 (a) Grayscale frame and (b) *event image* showing the laser beams emitted over a black canvas. The *event image* was rendered by accumulating events for 250 ms. The colors on the images (red, green, blue, and magenta) indicate the correspondence between the laser beams in the frame and their equivalent in the *event image*. 34

2.2 Validation results of an experiment where a laser beam moves (red) in different directions on the canvas while being successfully tracked by the proposed algorithm (blue). The black dot represents the initial location of the laser beam, while the green dot corresponds to the location after 7 s. 35

2.3 (a) A grayscale frame and (b) its equivalent reference Mask I_M from the samples of the *shapes_6DoF* dataset. 39

2.4 Experimental results obtained by varying the value of n^C in the range [1000, 20000] using the sequences of the *shapes_6DoF* dataset. 40

2.5 (a,c) *Event images* of accumulated events within a time window of 25 ms, and (b) their cluster representation using the proposed clustering algorithm with $n^C = 1500$, and (d) $n^C = 15000$. 40

2.6 (a) Clusters of events obtained by using the proposed method, and (b) their equivalent representation after being rendered over the mask I_M. The color of each event sample (i.e., colored pixels) indicates the cluster identifier. 41

3.1 General scheme of the proposed event-based guidance approach. Green blocks represent the perception modules, while red blocks the control modules. The TTC trajectory module is optional, and its integration depends on the robot platform. 50

3.2 The *event-by-event* line tracker in one example: (a) processed input events (red and blue) and prototype coordinates of each line P_{l_i} (big green, orange, and purple points); (b) tracked lines in the image plane; (c) event Hough space representation showing the tracked lines (green, orange, and purple). 56

3.3 Undistorted *event images* along with the tracked lines in situations where different objects intersected the reference pattern. The intensity frames (without lens undistortion) on the first row provide a clear representation of the experiment. 63

3.4 Aerial platform used in the TTC guidance experiments. The quadrotor is equipped with a DAVIS346 event camera and a Khadas VIM3 for event processing. 65

3.5 Evolution of four gaps along the maneuver. (a) main ρ gap, (b) a θ gap, (c) and (d) the best and worst ρ coupled gaps. 66

3.6 (a-left) The trajectory followed by the quadrotor during the experiment. (a-right) The robot-goal distance along the maneuver. (b) Average robot spatial error in reaching the goal pose within different values of T_g. 67

3.7 (a,b) examples of the experiments performed by modifying the illumination conditions of the scene. (c,d) tracked lines describing the reference pattern at the start and goal configurations. 68

3.8 Sequences of the different validation experiments performed with the flapping-wing robot: (a) *Flapping descending*, (b) *Flapping straight*, and (c) *Flapping outdoors*. 71

3.9 Top-left) the module of the linear acceleration registered by the onboard IMU during a guidance experiment. Top-right and bottom) evolution of the line components (θ, ρ) of three tracked lines along the experiment. The flapping-wing robot was approximately launched at time 0.7 s. . . 72

3.10 Examples of lines tracked by the proposed algorithm in different scenarios: (a) lines tracked in an indoor experiment, and (b) lines tracked during an outdoor validation test. The grayscale frames were used only for visualization purposes. 72

3.11 Evolution of the velocity error $\nu(t)$ (top), and ornithopter position (bottom) during the guidance maneuver described in Figure 3.9. The purple and yellow colors correspond to the launching and flapping stages of the experiment. The robot pose was obtained from the motion capture system. 74

4.1 General diagram of the *sense-and-avoid* scheme for ornithopter flight. 83

4.2 General block diagram of the *Dynamic Object Motion Estimation* pipeline: (a) the *Time Filter* filters events generated by the background of the scene; (b) the *Corner Detector* extracts relevant features of the object; (c) the *Optical Flow* submodule computes the direction of motion of events in the image plane; and (d) the *Clustering* submodule estimates the mean flow of the dynamic object. The *event images* of each submodule are examples of the results obtained by processing events triggered within a time window of 10 ms. 84
4.3 An example of the events considered as generated by a dynamic object using the *Time Filter* submodule. (a) grayscale frame from the APS sensor of the DAVIS346; (b) *event image* rendered by accumulating events in time windows of 10 ms (black pixels). The set of events considered to belong to the moving object is shown in magenta. . . . 86
4.4 The ornithopter body is approximated by a $2H \times 2W$ volume and the obstacle is represented by a sphere of radius R'. The collision evaluation constraints are defined by ψ^* and θ^*, which are used to determine possible collisions with the ornithopter. 89
4.5 Objects of different shapes and sizes selected for the *sense-and-avoid* experiments: left) Small Box; center) Stuffed Toy; and right) FitBall. 93
4.6 Distance in the image plane between the centroid of the detected obstacle and the ground truth for all performed experiments. 94
4.7 Summary of the motion estimation experiments. (a) box plot errors and (b) mean error and standard deviation. The directional error was defined as the mean angle difference between the estimated direction of motion and the ground truth. 96
4.8 Evolution of ψ^* and θ^* by varying the distance between the obstacle and the robot (Z) for the three objects used in the experiments. . . . 97
4.9 An example of a *sense-and-avoid* outdoor experiment, which corresponds to a sequence of images that includes the robot and the obstacle position during the experimental validation. 99

5.1 Block diagram of the intrusion monitoring method. 106

5.2 The output of each module of the *IM* method rendered in an *event image*: (a) corner tracking, (b) APM, (c) event clustering, and (d) intruder detection. These results were obtained by accumulating input events and the output of each module in a time window of 25 ms. . . 107

5.3 Block diagram of the intrusion monitoring scheme. It includes several modules (e.g., *Intrusion Monitoring* and *Auto-tuning*) and submodules such as the *Object Detector*, and the *Synchronizer*. 113

5.4 Multirotor platforms used for the validation experiments. (a) DJI Flamewheel F550, and (b) a custom-made Hexarotor equipped with a PixHawk autopilot. Both platforms equip a DAVIS346 event camera mounted at $-45°$ pitch rotation. 118

5.5 Results of the intruder monitoring evaluation in four experiments: (a) daylight, (b) night, (c) multi-object, and (d) illumination changes. The white and orange points represent the set of clustered events and events corresponding to feature tracks. These images were obtained by rendering the events, and *IM* results accumulated during the last 25 ms over the grayscale frames from the APS sensor. 119

5.6 Performance results for all experiments. (a) *Accuracy*, (b) *Precision*, and (c) *Recall* as a function of γ. 122

5.7 An example of a surveillance mission including scenarios with industrial objects and nature. The frames describe the 6 DoF control points of the robot in the monitoring mission. The blue semicircular areas describe the camera field of view at each control point. 124

5.8 Snapshots of the 15 surveillance zones (ordered from left to right, and from top to bottom) of the surveillance mission shown in Figure 5.7. . 126

5.9 Evolution of the cost functions $J(\hat{\mathbf{v}})$ and $J(\mathbf{v})$ for the parameter tuning process of zone 7. $J(\mathbf{v})$ represents the performance of the solutions analyzed along the exploration of the search space. 127

5.10 The evolution of $J(\hat{\mathbf{v}})$ for each zone during the tuning process. 127

6.1 Different trajectories χ for $0 < k_\tau \leq 1$ closing the gap at $t = 0$. For $k_\tau = 1$ (light blue curve) the gap closes with $\dot{\chi}$ constant leading to a collision. 142

6.2 Block diagram of ROSS-LAN. 145

6.3 A set of normalized trajectories describing bioinspired simulated trajectories. The magnitude and direction of the vectors describe the robot's velocity, while the tail of each vector corresponds to the robot's position. 146

6.4 Perspective views of the simulation scenarios used to record the dataset. (a) A warehouse and (b) an oil refinery 147

6.5 A visual representation of some of the sensing measurements collected in the datasets. (a,c) Grayscale frames along with the simulated events accumulated in time windows of 10 ms. (b,d) The point clouds from the LiDAR 148

6.6 Distribution of the perception hardware on the GRIFFIN *Eye-Bird* in an experimental scenario: (A) Leica GRZ101 MiniPrism, (B) DAVIS346 camera, (C) battery, (D) the flapping mechanism, (E) VectorNav VN-200 IMU, (F) Khadas VIM3 board, and (G) tail servos. 150

6.7 The GRIFFIN *Eye-Bird* ornithopter with the reference frames of each sensor: (V) VectorNav, (C) DAVIS346 APS and DVS, (I) DAVIS346 IMU, and (W) world reference frame either from the TotalStation or the OptiTrack. The DAVIS346 was mounted upside down, affecting frames (C) and (I). The TotalStation prism is located at coordinate (0, 37.2, 0) mm at frame (I). 153

6.8 The quadrotor platform and the *Eye-Bird* flying together in an experiment. 155

6.9 The number of events per millisecond generated in *Eye-Bird* and the quadrotor while describing a similar trajectory: (a-b) Frames of accumulated events at 40Hz on the quadrotor (a) and on the ornithopter (b); and (c) the number of events accumulated every millisecond in both platforms. 156

6.10 Event generation describing the flight stages of the ornithopter – *launching*, *flapping*, and *landing*. Top) the events triggered per millisecond, down) and $\|\vec{r}\|$, the norm of the position vector in dataset *Hills Base 3*. 157
6.11 Some of the perception data collected in the dataset along with a picture of each scenario: (a) *Soccer*, (b) *Hills*, (c) *People* flight in the *Soccer* scenario, (d) *Testbed*. 158
6.12 Robot trajectories estimated using ROVIO (in blue) against the ground truth (in magenta) in the flights *Testbed ArUco1* (Left), *Hills ArUco1* (Right-up), and *Soccer ArUco1* (Right-down). 161

Chapter 1

Introduction

1.1 Motivation

Multirotors are the most used aerial platforms in robotics [1]. Compared to other aerial vehicles (e.g., fixed-wing platforms), they are preferred due to their easy deployment, relatively low cost, extended payload, and hovering capabilities. Many works have reported the benefits of integrating perception sensors and algorithms on multirotors in applications such as surveillance [2] [3], inspection [4] [5] [6], object transportation and delivery [7] [8] [9], maintenance [10] [11], and autonomous navigation [12] [13] [14], among many others. However, many of these applications enclose relevant challenges for the onboard perception systems by being subject to fast motions, high vibrations, and strong changes in the illumination conditions (e.g., passing from an indoor scenario to an outdoor space). For instance, the sensing rate of the majority of commercial Laser imaging Detection and Ranging (LiDAR) sensors ($\leq 20\,\text{Hz}$ [15]) limits their use in applications where the robot moves at high velocities. Moreover, perception algorithms relying on frame-based cameras are particularly prone to the aforementioned conditions. These sensors suffer from motion blur, which hinders the performance of traditional image-based perception methods. Besides, the typical 60 dB dynamic range of standard cameras [16] limits their use in applications where the robot flies in environments describing large variations in the illumination conditions [17]. Moreover, although multirotors report a higher payload compared to other aerial platforms (e.g., fixed-wing and flapping-wing robots), they require sensors with low power consumption to enlarge their flight time in applications such as inspection [18] and monitoring [19]. The aforementioned challenges suggest the integration of novel sensors providing low latency sensing, high dynamic range, and low power consumption to solve the sensing issues presented on multirotor flight. Finally, there are relevant open challenges for multirotors platforms. First, they require efficient schemes to optimize their energy consumption to enlarge their flight time. Second, although the current multirotor technology aims at providing safety platforms for human-robot interaction, these platforms still represent a potential danger due to their rotary propellers. Current research on aerial robot design focuses on the development of novel user-friendly platforms which represent the lowest possible danger to their users.

In the last years, there has been a significant research effort in the development of aerial vehicles that solve the above challenges. For instance, soft aerial robots [20] [21] [22] use flexible, soft, energy-storing, and adaptive structures and materials to save energy [23], increase user safety [24], extend time-of-flight [25], among others. Moreover, aerial platforms with morphing capabilities modify their geometry to improve their flight stability and maneuverability [26], and enhance their versatility to perform different tasks [27] [28] [29]. Flapping-wing robots, also known as ornithopters, are bioinspired aerial robots that mimic the wing motion of birds and insects during flight. Some of these platforms integrate soft materials and morphing mechanisms to perform perching [30] and improve their flight performance [31] [32]. Ornithopters generate lift and trust by flapping their wings. They are considered less dangerous than multirotors by being manufactured with lightweight materials and using soft wings instead of rigid propellers. Additionally, there is a relevant R&D interest in developing lightweight ornithopters that report lower power consumption [33]. The current research in ornithopter technology also focuses on the development of novel flying and perching mechanisms [34] [35] [36], the validation of their aerodynamic models [37] [38], and the design of novel flight controllers [39], among others.

However, there is still a huge gap to close between the implementation and validation of perception systems for these platforms and the development of flapping-wing robots with fully autonomous capabilities. Ornithopters present several challenges to robot perception. First, the restricted payload of these platforms together with their complex weight distribution limit the installation of perception sensors and processing units. The majority of works integrating perception algorithms in flapping-wing robot applications use either external sensors [40], or specialized processing units that directly gather and compute the perception information [41]. Additionally, the flapping strokes produced by ornithopters and their agile motion represent additional challenges. For instance, the strong flapping strokes exerted by large-scale ornithopters produce motion blur in intensity images [42], which limits their use with standard frame-based computer vision algorithms for robot perception. Besides, performing agile maneuvers in outdoor scenarios tends to produce large variations in the illumination conditions [43]. This affects standard frame-based cameras due to their limited dynamic range to adapt to these conditions. Further, the fast flight velocities of medium-scale and large-scale ornithopters (e.g., $16\,\mathrm{m\,s^{-1}}$ [44]) suggest the use of perception sensors and methods that provide low-latency perception estimations. Finally, ornithopter robots mount lightweight batteries with limited power capacity (e.g., $0.6\,\mathrm{W}$ [41] and $7.5\,\mathrm{W}$ [34]), which restricts the use of perception sensors with high power requirements (e.g., LiDARs and depth cameras). The previously presented challenges for flapping-wing robot perception indicate the use of novel sensors and methods to provide robust and low latency perception estimations for ornithopter robots.

During the last decades, advances in vision sensor technology have led to novel devices trying to imitate the behavior of human and animal visual systems. This is the case with event cameras, which are neuromorphic sensors that mimic the neural architecture of the eye. Unlike frame-based cameras that measure absolute brightness at a constant frame rate, event cameras measure asynchronous per-pixel brightness changes in the scene. An event is triggered when a brightness increment in a pixel exceeds a predefined magnitude at a specific time. Hence, event cameras asynchronously provide pixel information with high temporal resolution. The event stream may be interpreted as a sequence of asynchronous data representing changes in brightness on individual pixels of the sensor array. Events are mainly transmitted using the address-event representation (AER) protocol [45] [46]. Event cameras offer several advantages compared to standard frame-based sensors including high temporal resolution, high dynamic range, robustness to motion blur, and low power consumption. Their high temporal resolution is due to their analog circuitry, and digital read-out (e.g., using a 1 MHz clock [47]) that allows triggering timestamped events with microsecond resolution. Thus, event cameras are robust to motion blur by capturing per-pixel information of very fast motions, different from standard frame-based sensors. Moreover, the logarithmic functioning scale of their photoreceptors together with their per-pixel operation provide a high dynamic range (i.e., [90-140 dB] [16]) compared to the 60 dB of traditional image-based cameras. This feature makes event cameras robust to strong variations in illumination, from daylight to pitch-dark conditions. Additionally, these sensors report low power consumption compared to other vision sensors by transmitting only per-pixel brightness variations instead of redundant intensity frame data. Different works [48] [49] [50] have reported power consumptions of $\sim$ 100 mW using event camera systems directly interfaced to an embedded processor. Recently, a complete survey on event-based vision has been published [16]. It briefly explains the principle of operation of event cameras, highlights the advantages offered by these sensors compared to traditional cameras, and describes their applications in robotics, neuromorphic computing, and computer vision.

Event cameras have been used in different computer vision and robotic applications, that benefit from the outstanding properties of these sensors. Some of these works regard to feature tracking [51], optical flow estimation [52], 3D reconstruction [53], visual internal odometry [54], and image reconstruction [55], among others. The aforementioned properties suggest event cameras as a suitable sensor for multirotor and flapping-wing robot perception by addressing several of the challenges entailed during the flight of these platforms.

This Ph.D. thesis focuses on exploring the use of event-based vision methods in two types of aerial robots; multirotors, and ornithopters. The advantages of event cameras compared to other vision sensors suggest them as a suitable solution for aerial robot perception. However, event cameras represent a new paradigm for robot perception as most state-of-art vision-based algorithms input frames instead of single-pixel information. Thus, the event stream requires further adaptations for being processed with traditional methods, and novel algorithms are needed to directly exploit the advantages of event cameras. Additionally, despite there is strong interest in developing flapping-wing platforms with perception capabilities, there are few ornithopter platforms that have mounted perception sensors [56]. This Ph.D. Thesis also aims to contribute toward the use of event-based sensors on flapping-wing robots.

1.2 Objectives

Event cameras offer several outstanding properties suitable to deal with the perception challenges caused by multirotor and flapping-wing robot flight. However, the current state-of-the-art reports few works that leverage these advantages on the perception of board aerial robots. Besides, to the best of our knowledge the use of event-based vision on flapping-wing robots is a novelty, which has been preliminary studied in our previous work [42]. The main objective of this Ph.D. Thesis is to explore the use of event-based vision on aerial robots by proposing a set of software algorithms and resources that contribute to the development of event-vision methods and systems suitable for multirotors and ornithopter robots.

The research enclosed in this Ph.D. Thesis proposes a set of low-level event-vision algorithms that leverage the asynchronous generation and microsecond resolution of event cameras. These algorithms intend to serve as low-level modules for more complex event-based perception methods. Furthermore, this Ph.D. Thesis presents a set of high-level perception schemes and algorithms for aerial robot perception. These methods propose event-based solutions for aerial robot applications where event cameras gain significant relevance by dealing with some of the challenges presented during the flight of multirotor and ornithopter platforms. Moreover, this research proposes a simulation tool to emulate bioinspired landing trajectories to narrow the gap toward the development of perception algorithms for bioinspired aerial robots. Finally, the current event-vision literature reports a few datasets including event information collected on board aerial platforms. This Ph.D. Thesis provides a set of event datasets recorded on board multirotors and flapping-wing robots to pave the way toward the study and development of specific event-vision methods for these platforms.

The GRVC Robotics Laboratory is a well-known research group for its work on aerial robotics research, offering a unique opportunity to develop perception systems for aerial robots. Multirotors are the main platforms used in the group to develop diverse robotic applications due to their versatility and relative low cost. Moreover, this Ph.D. Thesis is mainly enclosed within the context of the GRIFFIN ERC project that focuses on the development of flapping-wing robots with diverse robot capabilities. Thus, offering an ideal environment for designing and validating event-based algorithms on ornithopter platforms.

1.3 Context

This Ph.D. Thesis has been developed within the context of several projects funded by the European Commission, and by the Spanish Ministry for Science and Innovation. Each of these projects are related to the development of perception systems for aerial robots. The Ph.D. candidate designed, implemented, and validated the developments presented in this Thesis within the GRVC Robotics Laboratory of the University of Seville.

- GRIFFIN [1] (ERC-2017-ADG-788247). The ERC Advanced Grant GRIFFIN project aims at developing flying robots with dexterous manipulation capabilities. The robots will use foldable wings with flapping capabilities. The GRIFFIN robots will mount sensors and boards to run autonomous and reactive perception methods. New software tools will be developed to facilitate the design and implementation of these complex robots and their perception algorithms. The GRIFFIN flapping-wing robots will be able to land autonomously using vision-based perception. The GRIFFIN project is the main funder of this Ph.D. Thesis. It mainly motivates this research and provides the ornithopter robots used to

[1] https://griffin-erc-advanced-grant.eu/

validate the proposed perception algorithms. This Ph.D. Thesis describes several of the perception methods developed on the GRIFFIN project by exploring the advantages of event cameras for flapping-wing robot perception.

- H2020 AERIAL-CORE [2] (H2020-2019-871479). The inspection and maintenance of large infrastructure is a big challenge for engineers and represents a major economic activity. Robot technologies can help increase efficiency, reduce costs, and keep workers out of hazardous conditions. The EU-funded AERIAL-CORE project develops an integrated aerial cognitive robotic system to assist human workers in inspection and maintenance activities. Specifically, it will integrate aerial robots for long-range and very accurate inspection of the infrastructure capability. Perception algorithms will be developed to detect operators and power lines for inspection and maintenance tasks. This Ph.D. Thesis provides a basis for the development of these perception methods using event-based vision together with traditional computer vision methods.

- ROBMIND (PDC2021-121524-I00). This project from the Spanish National Prueba de Concepto R&D Programme will develop novel robotic prototypes for the inspection and maintenance of industrial plants. It will develop robotic prototypes for contact and contactless inspection as well as simple maintenance tasks, allowing industrial companies to adopt more agile and frequent predictive maintenance. Perception is particularly sensitive in complex environments with a lack of structure. Critical aspects are accuracy and robustness against lack of features in many industrial settings, e.g., inside a vessel or a storage tank. These industrial scenarios report low illumination conditions that increase the perception challenges for inspection and maintenance. This Ph.D. Thesis provides an initial step toward the development of perception methods for the inspection of industrial scenarios under low illumination conditions.

- HAERA (PID2020-119027RB-I00). This project funded by the Spanish National RETOS R&D Programme aims at developing flapping-wing platforms to collect

[2]https://aerial-core.eu/

samples and monitor the environment. The robot platforms integrate different perception and control modules that allow them to perform autonomous flight and monitoring operations. Besides, these platforms integrate specific mechanisms for takeoff and landing in the water. This Ph.D. Thesis serves as an initial study to explore the advantages of using event-based cameras for aerial monitoring tasks.

1.4 Contributions

This section summarizes the contributions of this Ph.D. Thesis.

1.4.1 Contribution 1: Event-vision UAS guidance

The problem of visual guidance for aerial robotics has been extensively studied in the literature. One of its main limitations is the given by the restriction imposed by frame-based cameras such as limited dynamic range, fixed frame rate, and motion blur. This contribution presents a guidance method for aerial robots using event-based vision. The method also includes a bioinspired time-to-contract trajectory generator to mimic the trajectories followed by birds while landing. The method has been initially validated in a multirotor platform and afterward on an ornithopter robot. To the best of our knowledge, this research describes the first event-based guidance method adapted for a flapping-wing platform. The results of this research were published in [57] and [58]. Figure 1.1 shows an experiment in which the *E-Flap* ornithopter flies toward its goal using the proposed guidance system.

1.4.2 Contribution 2: Event-based dynamic sense-and-avoid for flapping-wing robots

Avoiding dynamic obstacles with large-scale flapping-wing robots requires fast perception processing and response to guide the ornithopter toward a collision-free direction before colliding with the obstacle. Event cameras trigger events due to the changes of illumination in the scene which in many cases are caused by the movement of objects

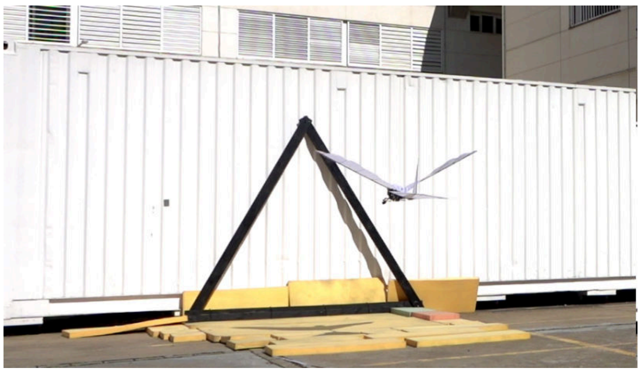


Figure 1.1: An outdoor guidance validation experiment using the GRIFFIN *E-Flap* ornithopter.

in the scenario. Thus, event cameras directly provide pixel information from moving objects. The research associated with this contribution focuses on exploiting the properties of event cameras for the sense-and-avoid (i.e., obstacle avoidance) problem with large-scale ornithopters. It proposes a sense-and-avoid pipeline which is evaluated in several experiments with the *E-Flap* ornithopter. The results show the outstanding capabilities of the pipeline for evading dynamic obstacles. The research associated with this contribution lead to the publication in [59]. Figure 1.2 shows an example of a sense-and-avoid experiment in which the *E-Flap* ornithopter evades a dynamic obstacle.

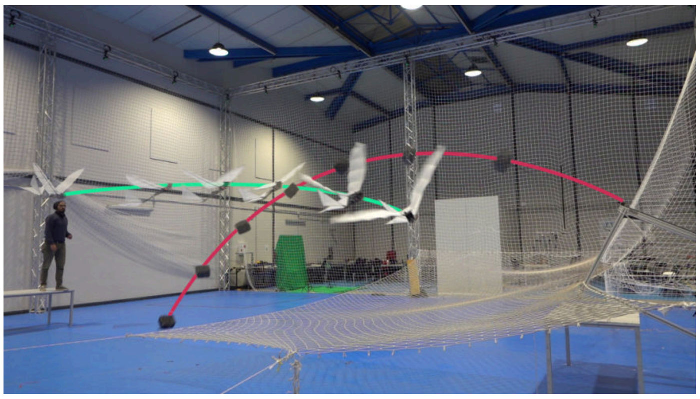


Figure 1.2: The GRIFFIN *E-Flap* ornithopter robot performing a *sense-and-avoid* maneuver.

1.4.3 Contribution 3: Event-based intrusion monitoring for multirotors

This contribution describes an event-based intrusion monitoring system for multirotors. The high dynamic range and the robustness to motion blur of event cameras make them a suitable candidate for intrusion monitoring and surveillance on board aerial robots. The high dynamic range of these sensors enables performing the monitoring under different illumination conditions without requiring additional sensors. The perception module of the system is described by several event-processing algorithms which extract and analyze events to detect intruders in the scene. The system includes an automatic tuning method to adjust the parameters of the event-based algorithms for different scenarios and conditions. The system has been tested in different multirotor platforms in several experiments varying the weather, scenes, and light conditions. The research enclosed by this contribution is described in [60], [3], [61], and [62]. Figure 1.3 shows an example of a monitoring mission in which an intruder is detected by the proposed method.

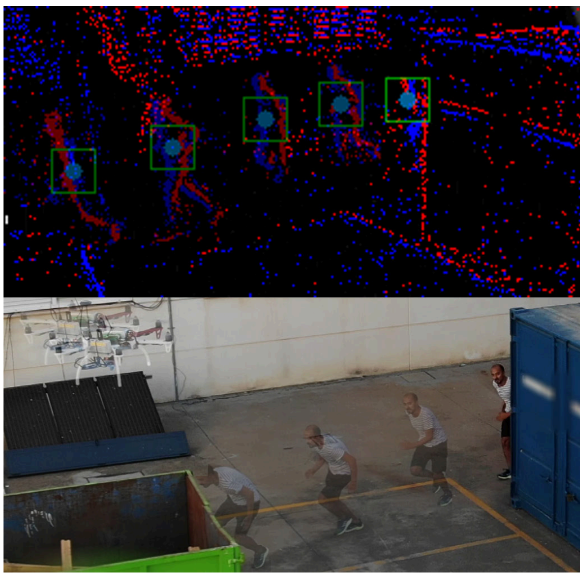


Figure 1.3: Intrusion monitoring results on board a quadrotor platform.

1.4.4 Contribution 4: Simulation tool for bioinspired landing perception

This contribution describes a perception simulation tool to retrieve the perception sensor measurements collected while performing bioinspired landing trajectories. The simulation tool integrates several relevant perception sensors for aerial robotics such as LiDARs, Lasers, Inertial Measurement Units (IMUs), traditional cameras, and event cameras. Besides, the tool includes a trajectory generator that provides bioinspired landing trajectories similar to those performed by birds during landing. A dataset with some of the sensing samples recorded in different landing experiments is publicly available. The research enclosed within this contribution corresponds to publications [63] and [64]. Figure 1.4 shows the block diagram of the proposed simulation tool.

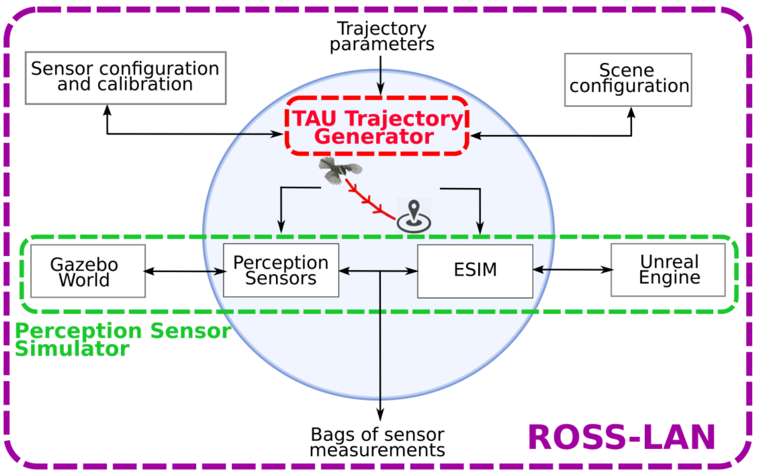


Figure 1.4: Block diagram of the ROSS-LAN simulation tool.

1.4.5 Contribution 5: A dataset for flapping-wing robot perception

Currently, the lack of available ornithopters together with the payload restrictions of these platforms limit the study of perception algorithms for flapping-wing perception. Allocating perception sensors and computing hardware on board a flapping-wing robot is a challenging task, the weight of the perception hardware must satisfy the payload specifications of the platforms without harming the robot's maneuverability and flight performance. This contribution describes the first perception dataset for flapping-wing perception. It integrates sensor measurements from a frame-based camera, two IMUs, an event camera, and ground truth pose samples from external sensors (e.g., a TotalStation and a motion capture system). The dataset has been recorded on board the *Eye-Bird* ornithopter, a flapping-wing robot designed and manufactured at the GRVC Robotics Laboratory. The dataset has been validated with a state-of-the-art Visual Inertial Odometry (VIO) method. The work related to this contribution is described in publications [42] and [43]. Figure 1.5 depicts the *Eye-Bird* robot flying during a dataset collection experiment.

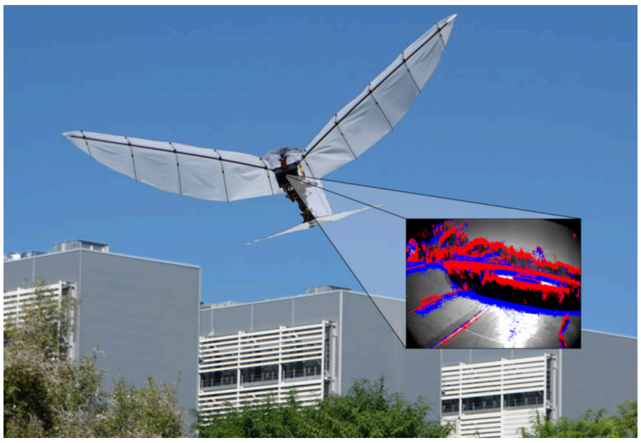


Figure 1.5: A dataset recording experiment using the GRIFFIN *Eye-Bird* ornithopter.

1.4.6 Contribution 6: Event-based algorithm validation on ornithopters

In the last few years, the interest in using event cameras for computer vision and robotics has increased exponentially. Currently, the development of lighter and high-resolution event-based vision sensors is constantly improving. However, there is still a huge gap to close toward the use of event cameras in robotics due to their limited commercial availability and relatively new state-of-the-art. Additionally, the development of event-vision methods for ornithopter platforms is mainly constrained by their weight and power limitations. This contribution refers to the experimental work developed in this Ph.D. Thesis by integrating event cameras and validating event-based perception methods in ornithopter robots. This is a challenging task due to the complexity of integrating sensors and processing boards on flapping-wing robots with strict payload and weight distribution specifications. Further, performing experiments with these platforms requires planning and coordination. Several experiments were conducted in challenging conditions where standard frame-based cameras may fail such as in low-illumination conditions and by performing fast motions with the robot. The majority of the articles enclosed in this research took part in this contribution [64], [42], [43], [57], [58], and [59]. Besides, this research also includes experiments performed with diverse multirotor platforms, which led to additional publication [60], [3], [61], and [62].

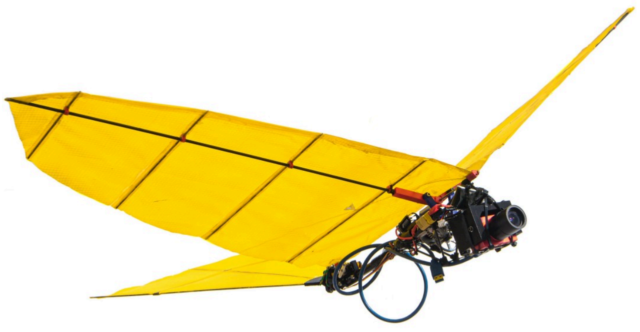


Figure 1.6: One of the GRIFFIN ornithopters designed and built at the GRVC Robotics Laboratory.

1.4.7 Contribution 7: Event-vision resources

This contribution corresponds to the set of event-vision algorithms, datasets, and tools publicly released during the development of this Ph.D. Thesis. These resources are available online to contribute to the event-perception community. The algorithms perform *event-by-event* processing adapting to the asynchronous event generation without requiring any additional pre-processing step to input the event stream. Moreover, the datasets intend to pave the way for the development of event-vision algorithms for robotics. Although the interest in using event cameras in robotic applications has increased in the last decade, there are still few event-vision available datasets compared to the number of perception datasets containing intensity frames from standard cameras. These resources are part of the research conducted in publications [64] [60] [3] [43], and [59].

1.4.8 Publications

The research described in this Ph.D. Thesis has been published in 8 conferences and 3 Journals. Among the most important conferences where the work on this research has been presented there are the IEEE International Conference on Robotics and Automation (ICRA) (4 contributed papers), and the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2 contributed papers).

This Ph.D. Thesis also produced publicly available resources including 3 datasets and 3 open-code algorithms, which are left available to the community.

1.4.8.1 Journals

- J. P. Rodríguez-Gómez, R. Tapia, M. d. M. G. Garcia, J. R. Martínez-d. Dios and A. Ollero, *Free as a Bird: Event-Based Dynamic Sense-and-Avoid for Ornithopter Robot Flight.* IEEE Robotics and Automation Letters 7.2 (2022), pp. 5413-5420. Links: Video, [59].

- J.P. Rodríguez-Gómez, R. Tapia, R, J. Paneque, P. Grau, A. Gómez Eguíluz, J.R. Martínez-de Dios and A. Ollero. *The GRIFFIN Perception Dataset: Bridging* *the Gap Between Flapping-Wing Flight and Robotic Perception.* IEEE Robotics and Automation Letters 6.2 (2021), pp. 1066–1073. Links: Dataset, Video, Code, [43].

- J. P. Rodríguez-Gómez, A. Gómez Eguíluz, J.R. Martínez-De Dios, and A. Ollero. *Auto-Tuned Event-Based Perception Scheme for Intrusion Monitoring With UAS.* IEEE Access 9 (2021), pp. 44840–44854. Links: Video, [3].

1.4.8.2 International conferences

- J.P. Rodríguez-Gómez, A. Gómez Eguíluz, J.R. Martínez-de Dios, and A. Ollero *ROSS-LAN: RObotic Sensing Simulation Scheme for Bioinspired Robotic Bird LANding.* Robot 2019: Fourth Iberian Robotics Conference. Springer International Publishing, 2020, pp. 48–59. Links: Dataset, [64].

- J.P. Rodríguez-Gómez, Cicco, M. D., Nardi, S., and Nardi, D *Mapping Infected Crops Through UAV Inspection: The Sunflower Downy Mildew Parasite Case.* International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer Cham, 2019, pp. 495–503. [63].

- A. Gómez Eguíluz, J.P. Rodríguez-Gómez, J.L. Paneque, P. Grau, J.R. Martínez de Dios and A. Ollero. *Towards flapping wing robot visual perception: Opportunities and challenges.* Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS). 2019, pp. 335–343. [42].

- J.P. Rodríguez-Gomez, A. Gómez Eguíluz, J.R. Martínez-de Dios, and A. Ollero. *Asynchronous event-based clustering and tracking for intrusion monitoring in UAS.* IEEE International Conference on Robotics and Automation (ICRA). 2020, pp. 8518–8524. Links: Dataset, Code, [60].

- J.R. Martínez-de Dios, A. Gómez Eguíluz, J.P. Rodríguez-Gómez, R. Tapia and A. Ollero. *Towards UAS Surveillance using Event Cameras.* IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). 2020, pp. 71–76. [61].

- A. Gómez Eguíluz, J.P. Rodríguez-Gómez, J.R. Martínez-de Dios, and A. Ollero. *Asynchronous Event-based Line Tracking for Time-to-Contact Maneuvers in UAS.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2020, pp. 5978–5985. Links: Video, [57]

- J.P. Rodríguez-Gómez, R. Tapia, A. Gómez Eguíluz, J. R. Martínez-de Dios and A. Ollero. *UAV human teleoperation using event-based and frame-based cameras.* 2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO). 2021, pp. 1–5. [62].

- A. Gomez Eguiluz, J.P. Rodriguez-Gomez, R . Tapia, J. Maldonado, J.A. Acosta and J.R. Martinez-de Dios and A. Ollero. *Why fly blind? Event-based visual guidance for ornithopter robot flight.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2021, pp. 1935–1942. Links: Video, [58].

- J.P. Rodriguez-Gomez, G. Gallego, J.R. Martinez-de Dios and A. Ollero. *Stabilizing Event Data on Flapping-wing Robots for Simpler Perception.* Workshop on Challenges of Flapping-wing aerial robots of the IEEE International Conference on Robotics and Automation (ICRA) 2022. [65].

1.4.8.3 Datasets

- *The event dataset for intrusion monitoring.* The dataset contains events and grayscale frames recorded in several intrusion monitoring situations. It was recorded on a multirotor flying in industrial scenarios under different illumination conditions. This dataset is part of the contributions of the work in [60]. Dataset

- *The perception dataset for bioinspired robot landing.* A dataset including synthetic sensing information from several perception sensors while simulating bioinspired landing trajectories. This dataset is part of the contributions of the work in [64]. Dataset

- *The GRIFFIN Perception Dataset.* A perception dataset recorded on board a large-scale flapping-wing robot. It includes measurements from different sensors relevant to flapping-wing perception. The dataset is enclosed in the contributions of the paper in [43]. Dataset

1.4.8.4 Software

- *dvs_data_tools*. A software tool to visualize event data in *event images* together with a set of scripts to transform *rosbag* files into text files and frames. The tool is set to receive events from either a Dynamic Vision Sensor (DVS) or a DAVIS camera. Code

- *grvc_ef_tracker*. A feature tracking algorithm that directly processes the event stream following an *event-by-event* processing approach. This algorithm is part of the perception modules of the work in [60]. Code

- *grvc_e_clustering*. An asynchronous clustering method to group events based on their spatial-temporal distribution. The clustering method is part of the perception modules of the work in [60]. Code

- *Eye-Bird Dataset Utils*. A repository with a set of useful scripts to handle the sensing information of the dataset in [43]. Code

1.5 Document structure

This Ph.D. Thesis describes the design, implementation, and evaluation of several event-based methods and schemes for multirotor and flapping-wing robot perception. Figure 1.7 shows a block diagram highlighting the relationship between the different chapters of this Ph.D. Thesis. A brief description of the document structure is reported as follows:

Chapter 2 describes a set of low-level processing algorithms for event-based vision. They intend to offer low-level perception primitives for the development of high-level event-based methods. Each algorithm leverages the asynchronous nature and microsecond resolution of event cameras by considering the spatial-temporal distribution of the event stream. Most of these algorithms are used as perception modules in the pipelines

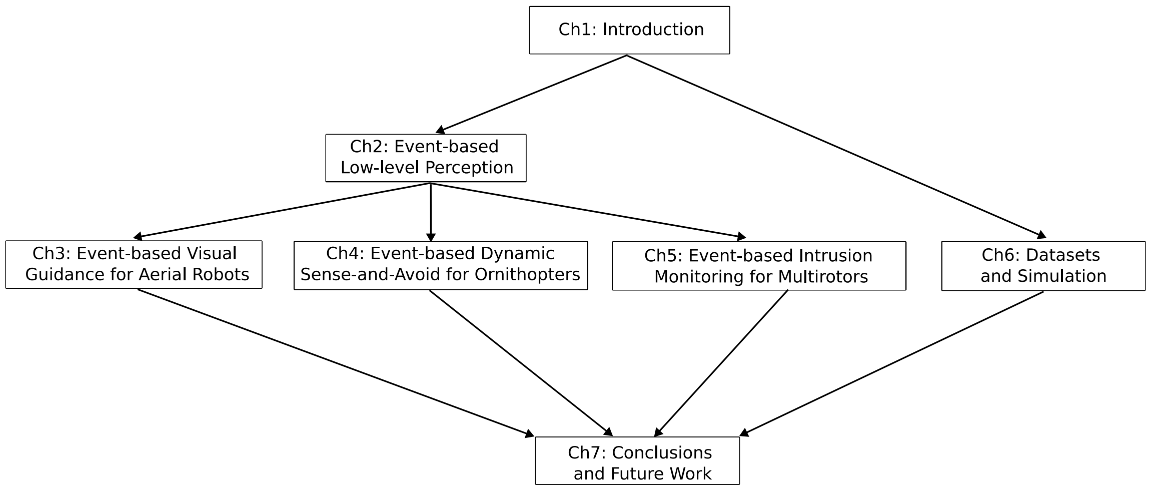

Ch1: Introduction
Ch2: Event-based Low-level Perception
Ch3: Event-based Visual Guidance for Aerial Robots
Ch4: Event-based Dynamic Sense-and-Avoid for Ornithopters
Ch5: Event-based Intrusion Monitoring for Multirotors
Ch6: Datasets and Simulation
Ch7: Conclusions and Future Work

Figure 1.7: Block diagram depicting the relationship between the chapters (Ch in the diagram) of this Ph.D. Thesis.

and schemes presented in the following Chapters. The presented methods are enclosed in **Contribution 1**, **Contribution 2**, **Contribution 3**, and **Contribution 7** of this Ph.D. Thesis.

Chapter 3 describes an event-based guidance approach for aerial robots designed to cope with the perception challenges that occur during the execution of fast guidance maneuvers. The proposed approach includes an event-based algorithm to detect and track line features defining the goal reference pattern. It includes as well a time-to-contact method to compute bioinspired landing trajectories. The method has been validated in two aerial platforms, a small quadrotor, and a large-scale flapping-wing robot. The experimental validation was performed in indoor and outdoor scenarios while varying the illumination conditions of the scene. This Chapter encloses **Contribution 1**, and its experimental validation is part of **Contribution 6** of this Ph.D. Thesis.

Chapter 4 introduces an event-based dynamic *sense-and-avoid* method designed and validated in an ornithopter. The proposed method focuses on detecting dynamic obstacles in the scene and reacting as fast as possible to prevent possible collisions with the robot's body. The perception module analyzes the event stream to extract event information from moving objects and estimate their direction of motion. An avoidance strategy together with the onboard controller guides the flapping-wing robot in a collision-free direction. The method was validated in a large-scale ornithopter designed and manufactured by the GRVC Robotics Laboratory. Several experiments were performed to validate the method in indoor and outdoor scenarios. This Chapter describes **Contribution 2**, and its experimental validation and software implementation are part of **Contribution 6** and **Contribution 7** of this Thesis.

Chapter 5 presents an event-based intrusion monitoring scheme for multirotors. It exploits the capabilities of event cameras to trigger pixel information from moving objects to detect possible intruders in the monitoring area. The scheme includes as well an auto-tuning approach to automatically adjust the parameters of the perception algorithms to improve the system performance. The proposed scheme offers a complete solution for autonomous intrusion monitoring using multirotor platforms equipped with event cameras. The system is validated in several experiments in diverse scenarios under different weather and illumination conditions. This Chapter corresponds to **Contribution 3**, and the collected datasets are part of **Contribution 7** of this Ph.D. Thesis.

Chapter 6 presents two different tools aiming to reduce the current gap between bioinspired aerial robotics and vision-based perception. In particular, both solutions have been designed to provide event-vision data, as event cameras offer a suitable solution for several of the perception challenges that arise during ornithopter flight. The first tool is a simulator that provides synthetic perception information during the execution of bioinspired landing trajectories. This tool corresponds to **Contribution 4** of this Ph.D. Thesis. The second tool is the first perception dataset recorded onboard a large-scale ornithopter. It includes sensing information from an event camera, a frame-based camera, two IMUs, and ground truth measurements of the robot pose. The perception dataset corresponds to **Contribution 5**, and its experimental validation and software tools are part of **Contribution 6** and **Contribution 7** of this Ph.D. Thesis.

Chapter 7 summarizes the conclusions of this Ph.D. Thesis highlighting the open challenges and future opportunities of using event-based perception for aerial robots.

Chapter 2

Event-based Low-level Perception

2.1 Introduction

Event cameras represent a new paradigm for traditional computer vision. These sensors provide single-pixel information with microsecond temporal resolution instead of the intensity images provided by frame-based cameras. Although event cameras offer many novel advantages over traditional vision sensors, the event stream cannot be directly processed by state-of-the-art computer vision algorithms. Rendering event pixels into frames is the simplest frame representation to use event data with frame-based perception methods. However, these frames do not provide the same brightness information as grayscale and colored images, which is used by many computer vision and Deep Learning (DL) algorithms. Additional processing is required to render *event-images* with enhanced information similar to the brightness information provided by traditional images [66].

Conversely, novel algorithms have been designed to exploit the asynchronous generation and format of events. Most of these algorithms process events in batches of accumulated events or by processing single events in an *event-by-event* manner. The latter leverages the nature of event cameras by analyzing asynchronous events individually allowing fast perception response and low latency processing. This Chapter presents a set of low-level *event-by-event* algorithms useful for event processing that have been developed in this Ph.D. Thesis. These algorithms are designed to cope with the event stream format. Besides they can be integrated with other *event-by-event* algorithms to develop high-level event-based perception methods [60] [3] [59].

The algorithms presented in this Chapter are included in the perception methods and systems of **Contribution 2**, **Contribution 3**, **Contribution 4**, and **Contribution 7** of this Ph.D. Thesis. Additionally, the research presented in this Chapter led to publications [60], [3], [62], [59], and [65].

This Chapter is organized as follows. Section 2.2 reports a summary of the main works related to the topics addressed in this Chapter. A set of low-level algorithms performing *event-by-event* processing is presented in Section 2.3. These algorithms are integrated with the majority of the perception methods developed in this Ph.D. Thesis. Section 2.4 describes the experimental validation of the proposed algorithms and includes a discussion of their possible use for a specific perception task. Finally, the conclusions and future work are presented in Section 2.5.

2.2 Related work

During the last decades, the neuromorphic and event-vision communities have designed and implemented a wide variety of methods to leverage the advantages of event cameras for computer vision, neuromorphic engineering, and robot applications. Some of the first works proposed frame representations of events that can be applied directly to state-of-the-art computer vision algorithms. These representations follow the concept of building 2D maps with the same sensor resolution. One of the simplest approaches is to build histograms of events $\mathbf{H} \in \mathbb{R}^2$ [67]. Each incoming event with coordinates $\mathbf{x}$ increases the value of the histogram at $\mathbf{H}(\mathbf{x})$. Thus, the histogram describes a map highlighting the areas with high spatial occurrence over time. The histogram is typically normalized to obtain a bounded representation of the event occurrence (e.g., ranging between $[0, 1]$ or $[-1, 1]$ considering the event polarity). This frame representation of events is known in the literature as *Image of Warped Events* (*IWEs*), *event frames*, and *event images*, among others. Many other 2D map representations have been proposed in the literature. Authors in [68] introduce the concept of Surface of Active Events (SAE), also known as Time Surfaces, as a map that keeps the temporal information of the last event occurring at location $\mathbf{x}$. The work in [69] proposes Time Surfaces with exponential time decay. The decay function considers the activity of events during a specific period of time, reducing the influence of old samples on the map. Inspired by the same concept, an exponential kernel decay approach is proposed in [70]. The exponential kernel computes the time decay of each new event in the neighborhood around it. The kernel spreads the influence of past events while providing information about the past activity in the event neighborhood.

Despite frame-based event representations are suitable inputs for traditional computer vision techniques, they do not leverage the asynchronous capabilities of event cameras. Conversely, *event-by-event* approaches process each event independently allowing asynchronous perception responses. Additionally, *event-by-event* processing methods avoid the latency produced by accumulating events to build meaningful *event images*. Some works [71] [72] have reported event processing rates similar to the event temporal resolution (i.e., 1 µs). Besides, *event-by-event* methods avoid computing redundant information of the entire frame by processing single events and their past information occurring in their spatial-temporal neighborhood [73] [74] [72]. Thus, they process only visual information triggered by the camera motion and the dynamic of the scene.

Feature extraction is one of the first event-vision topics reporting *event-by-event* processing methods. The work in [73] proposes an event implementation of the Harris corner detector [75] with similar accuracy to the frame-based version. The method reports outstanding detection accuracy with a high processing latency compared to the event resolution. Authors in [74] propose a fast event-based corner detector relying on the temporal information of an SAE. This method performs comparison operations in a circle around the input event to detect corners. Following the same principle, the approach in [72] compares arcs greater than 180° to enhance detection speed and accuracy. The method also proposes a tool to avoid processing redundant events which significantly reduces the number of events processed by the method. The work in [71] proposes an adaption of eHarris method in [73] including the event filter in [72] to increase the responsiveness of the corner detector. The work includes as well a detailed comparison between different event-based corner detectors highlighting the advantages of the proposed method. Recently, authors in [76] present an event-based corner detector with high event throughput and similar accuracy to the original Harris algorithm. Authors claim that the proposed descriptor overcomes the state-of-the-art of event-based corner detection algorithms providing additional metrics and evaluations compared to previous works.

Several additional low-level *event-by-event* methods have been proposed in the literature. The method in [51] presents a corner tracking method based on graph trees to discard unreliable corners. The method is evaluated with benchmarking event datasets reporting significant improvements in terms of accuracy and computational efficiency. A patch-feature tracking approach is presented in [77] which considers the feature tracking task as an optimization problem of matching a feature template with the current tracking state. Feature templates are defined by square patches of accumulated events occurring in the neighborhood of the current tracked feature. The method reports remarkable results providing fast and stable feature tracks compared to the work in [51]. The authors in [78] present an event-based cluster tracking algorithm inspired by the state-of-the-art mean-shift method in [79]. The mean-shift solution is formulated as a gradient descent problem of matching the current input event with the feature space. A Kalman Filter is used to perform multi-target clustering providing smooth tracking trajectories. The method demonstrates robustness to different cluster shapes and different camera speeds. The work in [68] presents an optical flow method relying on an SAE describing the temporal evolution of previous events. The surface is updated with each new event, and flow is computed by analyzing the previous temporal information around the event location in the SAE. Authors in [80] propose a block-matching approach for event-based optical flow. The flow is computed based on previous time information on slice windows similar to time surfaces. The method is extended in [81] by including an adaptive time reference to update the slice windows. The adaptative time reference is given by a grid of accumulated events that updates the slice windows based on the event occurrence in different areas of the image plane.

2.3 Low-level perception algorithms

This Chapter presents four low-level event-based algorithms following *event-by-event* computation. The set of algorithms intends to serve as a tool to build complex pipelines for event-based vision. Several of these algorithms are used in the next Chapters of this Ph.D. Thesis to build more complex methods for robot perception. As many computer vision and perception algorithms, the proposed methods include internal parameters to adjust the algorithm behavior to the user application and the scene configuration. Their correct adjustment leads to improvements in the algorithm's performance, as it is discussed in Section 2.4 and the following Chapters. Table 2.1 provides a summary of the parameters of each method.

Each event is defined by the tuple $e(\mathbf{x}, ts, p)$, where $\mathbf{x}$ represents the event coordinates (x, y), ts corresponds to the timestamp of the event, and $p \in \{+1, -1\}$ is the event polarity. The previous notation is used throughout this Ph.D. Thesis to refer to an event and its parameters. The following subsections describe each of the proposed algorithms.

2.3.1 Asynchronous feature tracking

Feature tracking is a relevant topic for several vision applications such as optical flow estimation, visual odometry, and vision-based Simultaneous Localization And Mapping (SLAM). This module focuses on tracking features obtained from the event stream following an *event-by-event* processing approach. Each feature is represented by $\mathbf{f} = \{\mathbf{x}, t_s\}$, where $\mathbf{x}$ is the feature coordinates $\mathbf{x} = (x, y)$, and t_s its timestamp (i.e., the timestamp of the event that triggered the feature). Inspired by the concept of Surface of Active Events, the proposed method defines Surfaces of Active event Features (SAeF), $\mathbf{S}_f \in \mathbb{R}^2$, as a map of the timestamps of previous features. The SAeF represents a spatial-temporal memory of previous features. Features on $\mathbf{S}_f$ are also buffered in buffer $\mathbf{B}^T$. Each new feature updates $\mathbf{S}_f$ and $\mathbf{B}^T$ by adding the new feature and removing the oldest sample such that they keep a maximum of n^T features. Thus, $\mathbf{S}_f$ describes a time-varying memory that adapts to the event generation, which

Algorithm	**Symbol**	**Description**
Feature Tracker	$\mathbf{f}$	Feature.
	$\mathbf{f}_c$	Feature candidate.
	$\hat{\mathbf{f}}_i$	Feature track i.
	$\hat{\mathbf{F}}$	List of features.
	$\mathbf{M}^T_{\mathbf{f}_c}$	Feature track candidate list of feature $\mathbf{f}_c$.
	$\mathbf{S}_f$	SAeF.
	A_T	Neighborhood area.
	ε^T	Tracker proximity threshold.
	$\mathbf{B}^T$	Event tracking buffer of size n^T .
	τ^T	Oldest timestamp in n^T.
Clustering	$\mathbf{c}$	Cluster.
	$\boldsymbol{\mu}_c$	Centroid of cluster c.
	$\hat{\boldsymbol{\mu}}$	Cluster moving average.
	$\mathbf{L}_c$	List of events associated to cluster c.
	$\mathbf{M}^C_e$	Cluster candidate list for event e.
	$\hat{\mathbf{C}}$	List of clusters.
	$\mathbf{B}^C$	Event clustering buffer of size n^C.
	τ^C	Oldest timestamp in $\mathbf{B}^C$.
	κ_c	Cluster proximity sampling number.
Attention Priority Map	$\boldsymbol{\Omega}$	Attention Priority Map.
	$\mathbf{B}^A$	Event buffer for APM of size n^A
	τ^A	Oldest timestamp in $\mathbf{B}^A$.
	l	Update window size.
	ν^A	Filtering threshold.
Time Filter	$\mathbf{S}_t$	Surface of events.
	τ_t	Time filtering threshold.
	β_{τ_t}	Complementary velocity gain.

Table 2.1: Notation used in *Feature Tracking*, *Clustering*, *APM*, and *Time Filter* methods.

mainly depends on the dynamics of the scene and camera motion. For instance, fast camera motions produce a $\mathbf{S}_f$ representation of features detected in a smaller time window than the representations obtained with slow camera motions.

The proposed method filters input features to avoid possible noisy samples. It discards event features with a number of neighbors in $\mathbf{S}_f$ lower than the threshold ε^T, typically set to 2. Valid filtered features are considered as feature candidates $\mathbf{f}_c$ for tracking. The set of feature tracks is represented by the list $\hat{\mathbf{F}} = [\hat{\mathbf{f}}_1, \ldots \hat{\mathbf{f}}_i]$, where $\hat{\mathbf{f}}_i$ is the $i-th$ feature tracked. The proposed tracker performs data association between the elements in $\hat{\mathbf{F}}$ and $\mathbf{f}_c$ producing either a new track or a tracking update. Candidate association is performed by analyzing the occurrence of each of the previous tracks $\hat{\mathbf{f}}_i$

in a neighborhood area A_T around the candidate $\mathbf{f}_c$. Previous tracks within A_T are appended to the list of associated tracks $\mathbf{M}^T_{\mathbf{f}_c}$. If $\mathbf{M}^T_{\mathbf{f}_c}$ has only one element, the track is updated with the values of $\mathbf{f}_c$. Conversely, if $\mathbf{M}^T_{\mathbf{f}_c}$ has more than one element, only the oldest track is updated. The method prioritizes older tracks as they provide a larger temporal consistency over time. Moreover, if $\mathbf{M}^T_{\mathbf{f}_c} = \emptyset$ a new feature track is created and appended to $\hat{\mathbf{F}}$.

Moreover, the proposed tracker discards tracks in the list $\hat{\mathbf{F}}$ in two cases. First, candidate tracks in $\mathbf{M}^T_{\mathbf{f}_c}$ that are not considered for tracking update are eliminated from $\hat{\mathbf{F}}$. Second, feature tracks $\hat{\mathbf{f}}_i$ with a timestamp lower than the dynamic reference time τ^T are canceled from the list of feature tracks. τ^T is a time-varying forgetting horizon obtained from the timestamp of the oldest feature in $\mathbf{B}^T$. Thus, only tracks updated within the time window defined by τ^T are kept in the list $\hat{\mathbf{F}}$. The proposed approach adopts a time-varying reference instead of a fixed value as the event generation in a specific time interval is affected by changes in the camera velocity and scene dynamics [82]. Finally, Algorithm 1 summarizes the functioning of the proposed event-based feature tracker.

Algorithm 1: Asynchronous event-based feature tracking

```
Input: $\mathbf{f}(\mathbf{x}, ts)$
Output: $\hat{\mathbf{F}}$
if isCandidate($\mathbf{f}, \mathbf{f}_c, \mathbf{S}, \varepsilon^T$) then
    $\mathbf{M}^T_{\mathbf{f}_c} \leftarrow$ SearchMatch($\hat{\mathbf{F}}$,$A_T$,$\mathbf{f}_c$)        ▷ Find matches in $\hat{\mathbf{F}}$.
    if $\mathbf{M}^T_{\mathbf{f}_c} = \varnothing$ then
        $\hat{\mathbf{F}} \leftarrow$ Append($\mathbf{f}_c$)        ▷ Add a new feature to $\hat{\mathbf{F}}$.
    end
    else
        $\hat{\mathbf{F}} \leftarrow$ Update($\mathbf{f}_c$)        ▷ Update oldest track by $\mathbf{f}_c$.
    end
end
$\hat{\mathbf{F}} \leftarrow$DiscardTracks($\mathbf{M}^T_{\mathbf{f}_c}$, $\tau^T$)        ▷ Remove old features.
return $\hat{\mathbf{F}}$        ▷ Return list of tracks.
```

2.3.2 Asynchronous event-based clustering

The events' format together with their asynchronous generation set a new paradigm for traditional computer vision. This Section describes a clustering approach for event data. It aims at clustering events triggered by objects in the image plane with a similar spatial-temporal distribution. Thus, it considers the spatial occurrence of events and their temporal resolution. The proposed method performs *event-by-event* processing to exploit the asynchronous nature of event cameras. Its spatial-temporal proximity criterion groups events without prior information about the scene geometry. Unlike methods using the cluster centroid as a reference for data association [78], the proposed method clusters events with spatial-temporal continuity by evaluating the proximity of new events to random samples associated with the cluster. This approach benefits *event-by-event* clustering as events are mostly triggered at the object contour which is often distant from its centroid.

Algorithm 2 describes the proposed event-based clustering method. The list $\hat{\mathbf{C}} = [c_1, \dots c_i]$ corresponds to the list of active clusters, where c_i is the $i - th$ cluster in the list. Each cluster $c_k \in \hat{\mathbf{C}}$ is defined by its centroid μ_{c_k}, its weighted moving average $\hat{\boldsymbol{\mu}}_{c_k}$, and its list of associated events $\mathbf{L}_{c_k}$. The proximity of an event e to a cluster c_k is evaluated using the distance from e to $\hat{\boldsymbol{\mu}}_{c_k}$, and to κ_{c_k} random samples in $\mathbf{L}_{c_k}$. The Manhattan distance is used due to its efficiency for high-dimensional data problems. An event is close to cluster c_k if any of these distances is below a threshold r (typically $r = 10$). A new event e assigned to cluster c_k updates $\hat{\boldsymbol{\mu}}_{c_k}$ as follows:

$$\hat{\boldsymbol{\mu}}_{c_k} = \frac{\alpha \mathbf{x} + (1 - \alpha)\hat{\boldsymbol{\mu}}_{c_k}}{2}, \tag{2.1}$$

where $\mathbf{x} = (x, y)$ corresponds to the event coordinates and $\alpha \in [0, 1]$ is the weight parameter. α is typically set > 0.5 to assign priority to new events. Each new event creates a list $\mathbf{M}_e^C$ of cluster candidates, if an event is close to a cluster $c_k \in \hat{\mathbf{C}}$, it is appended to $\mathbf{M}_e^C$. An event e is assigned to c_k in the two different cases. First, if $|\mathbf{M}_e^C|= 1$, e is appended to the cluster, where $|\cdot|$ refers to the cardinality of the list. Second, if $\mathbf{M}_e^C$ contains more than one element (i.e., $|\mathbf{M}_e^C|> 1$), the clusters in $\mathbf{M}_e^C$ are merged and e is added to the merged cluster. Otherwise, if $\mathbf{M}_e^C = \emptyset$, a new cluster is created with the event. Each new event-cluster assignation updates $\hat{\boldsymbol{\mu}}_{c_k}$ and $\mathbf{L}_{c_k}$ of cluster c_k. Additionally, following the idea of an adaptable forgetting horizon, the clustering algorithm includes a mechanism to delete old samples. A buffer $\mathbf{B}^C$ keeps the n^C most recent events. $\mathbf{B}^C$ is updated with each incoming event by adding the new event, and removing the oldest event in $\mathbf{B}^C$ if its size is greater than n^C. The forgetting horizon τ^C corresponds to the timestamp of the oldest event in $\mathbf{B}^C$. Events on the lists $\mathbf{L}_1 \dots \mathbf{L}_i$ are periodically deleted if their timestamp is lower than τ^C. Thus, each cluster keeps an event representation that follows the event generation over time. Finally, if all events are removed from cluster c_k (i.e., $\mathbf{L}_{c_k} = \emptyset$) the cluster is removed from the list of active clusters $\hat{\mathbf{C}}$.

Algorithm 2: Asynchronous event-based clustering

```
Input: $e, r, \kappa_c$
Output:
if $|\mathrm{M}_e^C| = 1$ then
    $\hat{\mathbf{C}} \leftarrow$ AddTo($e$, $\mathbf{M}_e^C$)                ▷ Add event to cluster.
end
else if $|\mathrm{M}_e^C| > 1$ then
    $\hat{\mathbf{C}} \leftarrow$ MergeAndAdd($e$, $\mathbf{M}_e^C$)          ▷ Merge clusters & add event.
end
else
    $\hat{\mathbf{C}} \leftarrow$ CreateNew($e$)                          ▷ Create new cluster.
end
for $c \in \hat{\mathbf{C}}$ do
    $\mathbf{L}_c \leftarrow$ DiscardSamples($c$, $\tau^C$)     ▷ Discard old events in the cluster.
    if $|\mathbf{L}_c| = 0$ then
        Remove($c$, $\hat{\mathbf{C}}$)                          ▷ Remove the empty cluster.
    end
end
return
```

2.3.3 Attention priority map (APM)

Event cameras trigger events due to the changes in illumination in the scene. These illumination variations are mainly caused by the camera motion and the movement of objects in the camera's Field of View (FoV). The latter is also known as the *scene dynamics*. Previous works have tackled the problem of distinguishing events triggered from the background (i.e., due to the camera motion) from those produced by dynamic objects [83] [84] [85]. Despite those approaches report remarkable results, they require either data from additional sensors or solving optimization processes that limit their real-time processing capabilities. This Section describes an algorithm to assign priority to events with high spatial-temporal occurrence using only event information. Following the event processing approach of methods in Section 2.3.1 and Section 2.3.2, the algorithm performs *event-by-event* computation to leverage the asynchronous capabilities of event cameras.

Inspired by neuroscience, the proposed approach uses the term *Attention Priority Map* (*APM*) to refer to a map $\mathbf{\Omega} \in \mathbb{R}^2$ that highlights regions triggering more events within a time window. Thus, $\mathbf{\Omega}$ describes the zones with high event occurrence in the image plane. Assuming that moving objects trigger significantly more events than the static background, the zones in $\mathbf{\Omega}$ with high occurrence correspond to moving objects. The APM ($\mathbf{\Omega}$) has the same resolution as the event camera, and it is updated asynchronously. Each new event e updates the APM by increasing the event occurrence in a window of length l around the event coordinate $\mathbf{x} = (x, y)$. The APM is updated as follows:

$$\mathbf{\Omega}_{ij} = \mathbf{\Omega}_{ij} + l - D(\mathbf{x}, \mathbf{y}) + 1, \tag{2.2}$$

where $D(\cdot)$ is the Manhattan distance, $\mathbf{y} = (i, j)$, $i \in [x - \frac{l-1}{2}, x + \frac{l-1}{2}]$, and $j \in [y - \frac{l-1}{2}, y + \frac{l-1}{2}]$. Conversely, the APM discards events with a timestamp older than τ^A. This time reference is obtained from the last event saved in buffer $\mathbf{B}^A$ of size n^A. Events are discarded from the APM as follows:

$$\mathbf{\Omega}_{ij} = \mathbf{\Omega}_{ij} + D(\mathbf{x}, \mathbf{y}) - l - 1 \tag{2.3}$$

The map $\mathbf{\Omega}$ is normalized within the range $[0, 1]$ for simplicity. A coordinate (x, y) is considered to draw attention when $\mathbf{\Omega}(\mathbf{x})$ is greater or equal than $\nu^A \in [0, 1]$. The threshold ν^A defines the regions in $\mathbf{\Omega}$ considered with high attention. The value of ν^A

is selected depending on the scene dynamics and the camera motion. For instance, when the camera motion is considerably lower than the object motion, small values of ν^A are enough to differentiate events triggered by dynamic objects. Conversely, when the camera velocity is close to the object velocity higher values of ν^A are required to differentiate events produced by moving objects.

2.3.4 Time filter

Different from frame-based vision sensors, event cameras provide pixel information with microsecond resolution. Several works have exploited the time resolution of events to perform event contrast maximization [86], high dynamic range video reconstruction [87], video deblurring [88], and high-speed frame interpolation [89], among others. The high time resolution of events can be used to differentiate events triggered by dynamic objects from those triggered by the camera motion. This Section introduces the *Time Filter* algorithm, a fast method to distinguish events produced by moving objects based on their temporal information. The method assumes a considerable velocity difference between the camera and the moving object.

The *Time Filter* saves the temporal information of previous events in the time surface $\mathbf{S}_t \in \mathbb{R}^2$. This time surface maps the event coordinates $\mathbf{x}$ with the timestamp ts of the last occurring event at $\mathbf{S}_t(\mathbf{x})$. Under the assumption that dynamic objects move faster than the camera, the proposed method differentiates events triggered from moving objects using the temporal difference Δt between the input event e_k and the previous timestamp at $\mathbf{S}_t(e_k)$. If the difference $\Delta t = ts_k - \mathbf{S}_t(\mathbf{x}_k)$ is lower than τ_t, the event is considered to belong to a moving object. Afterwards, e_k updates the time surface by $\mathbf{S}_t(\mathbf{x}_k) = ts_k$ for the future evaluations.

The threshold τ_t works as a *Time Filter* and its value directly impacts the performance of the method. Small values of τ_t impose a strong filtering action while large values of τ_t allow noise samples. In the simplest case, where the camera moves with a constant velocity, a fixed value of τ_t is enough to filter events belonging to the background. However, when the camera motion varies the selection of τ_t requires a different approach. Assuming that the camera velocity is known, τ_t can be represented as a function of the linear speed v_c and the angular speed ω_c of the camera:

$$\tau_t = \beta_{\tau_t}\tau_{\mathrm{v}_c} + (1-\alpha_c)\tau_{\omega_c}, \tag{2.4}$$

where τ_{v_c} and τ_{ω_c} represent the time contribution due to each speed and $\beta_{\tau_t} \in [0,1]$. τ_{v} and τ_ω are defined as follows:

$$\begin{bmatrix}\tau_{v_c}\\ \tau_{\omega_c}\end{bmatrix} = \begin{bmatrix}\tau_H\\ \tau_H\end{bmatrix} - (\tau_H - \tau_L)\begin{bmatrix}(\|\mathrm{v}_c\| - \|\mathrm{v}_L\|)(\|\mathrm{v}_H\| - \|\mathrm{v}_L\|)^{-1}\\ (\|\omega_c\| - \|\omega_L\|)(\|\omega_H\| - \|\omega_L\|)^{-1}\end{bmatrix}, \tag{2.5}$$

where $[\|\mathrm{v}_L\|, \|\mathrm{v}_H\|]$ and $[\|\omega_L\|, \|\omega_H\|]$ are the typical speed ranges of the camera, and the operational range $[\tau_L, \tau_H]$ can be found empirically. The camera speed ranges vary depending on the application and may be set experimentally.

2.4 Experiments

This Section describes the experimental validation of the *Tracking* and *Clustering* methods along with a discussion of the advantages and disadvantages of the *APM* and *Time Filter* algorithms for the problem of moving object detection. The following validation does not intend to compare the proposed algorithms with the methods in [51], [90], and [91]. Some of them are not publicly available for comparison [51], or they describe different processing approaches [90] [91] to ours. Instead, this validation aims at evaluating the *event-by-event* capabilities of the proposed algorithms for their integration into more complex event-based processing schemes for aerial robot perception.

2.4.1 Tracking

This subsection validates the tracking capabilities of the method proposed in Section 2.3.1. Ground truth tracking references were obtained from the Kanade-Lucas-Tomasi algorithm [92]. The validation was divided into two experiments. In the first set of experiments, reference features were directly extracted from the event stream to avoid tracking failures due to False Negative detections from feature detection algorithms. Thus, input events corresponded to the set of features to track using the proposed method. In the second set of experiments, a corner detector algorithm was used to extract the set of features to track (i.e., corners) with the proposed method. These experiments were performed with data recorded in complex scenarios where a feature detection algorithm is needed to identify features in the event stream. Both experiments focused on validating the lifetime and Mean Absolute Error (MAE) metrics of the tracking method. In this validation, tracking lifetime was computed as t_T/t_F, where t_T corresponds to the time that the tracker successfully followed the reference feature, and t_T is the time that the feature occurred during the experiment. Besides, the MAE was calculated as the mean difference between the location of the GT samples and the tracking features. The visual sensing data used in the experiments were provided by DAVIS cameras integrating both a Dynamic Vision Sensor (DVS) and an Active Pixel Sensor (APS).

The proposed algorithm and the method in [92] receive as input feature samples, which are usually provided by a feature detector (e.g., a corner detector). Thus, if the detector fails due to misdetection issues, it might affect the tracking performance. To avoid this issue, the first set of experiments focused on evaluating the proposed method when it received input features extracted directly from the event stream. The experiment required a special setup, where feature samples were provided by a set of laser beams emitting over a black canvas. The beams triggered events and produced white pixels in intensity images. A DAVIS346 provided events and grayscale frames during each experiment. The laser light generates brightness changes on the scene, triggering events while producing grayscale images with white pixels representing the laser beam. Figure 2.1 shows a grayscale frame with the laser lights displayed on the canvas, and its equivalent *event image* created by accumulating events in time windows of 250 ms.

Ground truth features were extracted from frames by normalizing the image (e.g., black and white frame), and defining white pixels as features. Multiple nearby white pixels were clustered such that there is only one feature per laser light. Conversely, event features are extracted from the event stream using only positive samples.

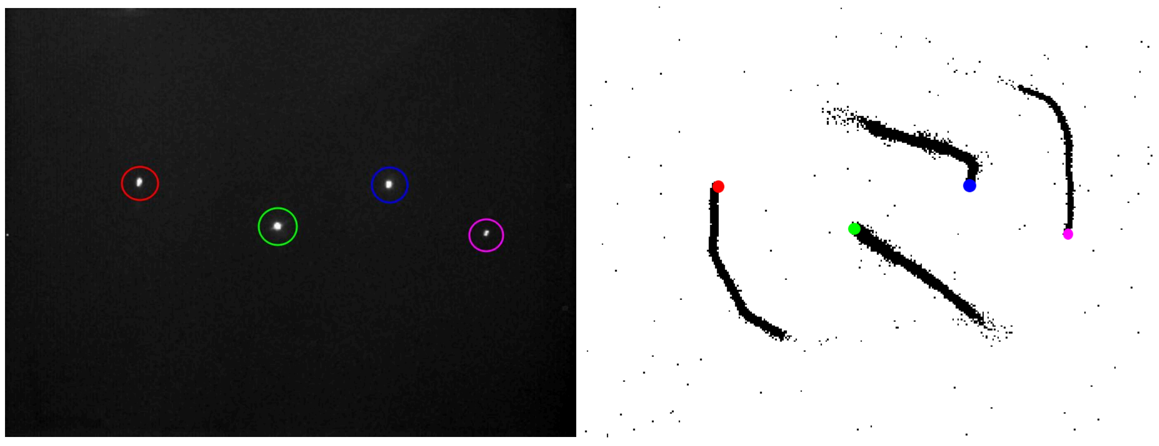


Figure 2.1: (a) Grayscale frame and (b) *event image* showing the laser beams emitted over a black canvas. The *event image* was rendered by accumulating events for 250 ms. The colors on the images (red, green, blue, and magenta) indicate the correspondence between the laser beams in the frame and their equivalent in the *event image*.

Negative polarity events were discarded. The MAE and lifetime were computed by comparing the ground truth tracking references extracted at t_I, with their equivalent event-based tracking references with the closest timestamp to t_I. The time reference t_I corresponds to the timestamp of the grayscale frame provided by the APS.

Three types of experiments were performed by emitting a different number of laser beams on the canvas; one, two, and four. Each experiment was repeated 10 times with a duration of $\sim$ 120 s where the laser lights moved in different directions within the canvas. Figure 2.2 shows an example of a tracking experiment where a single laser beam moved in the scenario. Table 2.2 shows the results obtained at each experiment. The proposed event-based algorithm reported an average feature lifetime of 95.21 %. Tracking errors occurred when the distance between two or more reference features was < 5 px. The proposed method performs data association by evaluating the occurrence of new features in a neighborhood window A_T centered at each tracker location. When this spatial condition was satisfied by several tracks, only the oldest was updated. Afterward, the newest track was either updated after the feature overlapping or deleted by τ^T. Moreover, the tracking MAE was 1.86 px among all experiments similar to the error reported by the method in [72]. Finally,

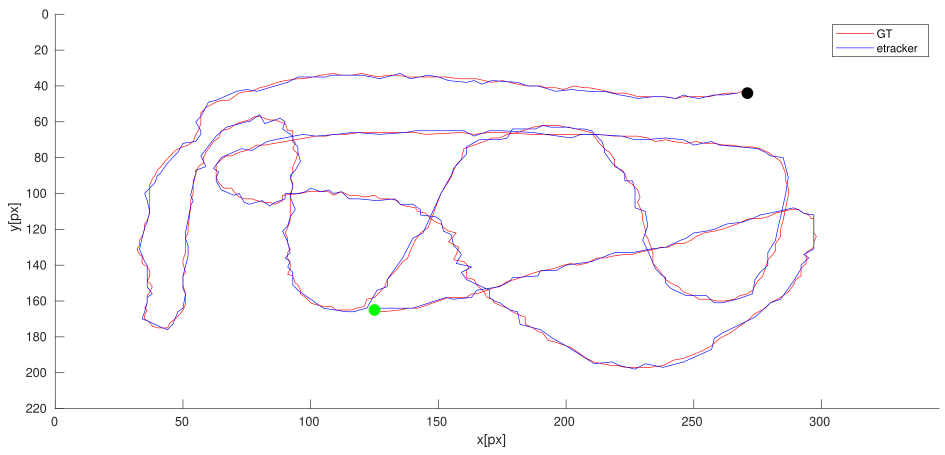


Figure 2.2: Validation results of an experiment where a laser beam moves (red) in different directions on the canvas while being successfully tracked by the proposed algorithm (blue). The black dot represents the initial location of the laser beam, while the green dot corresponds to the location after 7 s.

the proposed tracker reported an average event processing time of 1.1 µs, close to the temporal resolution of the DAVIS346 event camera (i.e., 1 µs).

No. of Laser beams	Lifetime	MAE (px)
1	99 %	1.87
2	94 %	1.57
4	90 %	2.43

Table 2.2: Experimental results of tracking laser beams. Each row describes an experiment with a fixed number of laser moving in the camera field of view.

In the second set of experiments, the proposed algorithm was validated with some sequences of the datasets in [64] and [93]. Ground truth features were extracted using the state-the-of-art Harris corner detector [75]. Similarly, event corner features were obtained using *eHarris corner detector with remarkable accuracy and False Positive Rate. *eHarris was implemented using the event-based version of the Harris algorithm in [73] combined with the filter in [72]. The synthetic sequences of the dataset in [64] describe perception data collected during the execution of bioinspired landing trajectories. These sequences are relevant for the development of perception algorithms for bioinspired aerial robots. Moreover, the dataset in [93] describes real and synthetic event data recorded in challenging conditions and complex scenarios. The dataset includes grayscale frames with motion blur due to the strong motion described by the camera. The experiments performed with this dataset used only the first 8 s of each set of sequences to avoid ground truth corner detection and tracking errors due to motion blur on the intensity frames. A total of six experiments were performed using three sequences from each dataset; *shapes_translation*, *shapes_rotation*, and *boxes_translation* from [93], and *warehouse_1*, *warehouse_2*, and *refinery_3* from [64].

Table 2.3 shows the Mean Absolute Error (MAE) and lifetime results obtained with the samples of each dataset. The proposed method reported remarkable results with the sequences of the synthetic dataset (i.e., rows 2 to 4 of Table 2.3). The most challenging synthetic dataset was *refinery_3*, where the robot described faster motions in a more complex environment. Conversely, the method's performance considerably decreased with two sequences of the dataset [93] (i.e., rows 6 and 7 of Table 2.3). The *shapes_rotation* dataset includes sequences with strong variations in the camera orientation. The rotational motion hindered the performance of the algorithm. However, it reported acceptable results using only a spatial-temporal approach for data association. Furthermore, the *boxes_translation* dataset includes sequences with several nearby corners in the image plane, which hindered the algorithm's performance due to the size of the window A_T.

Dataset	Lifetime	MAE (px)
warehouse_1	94 %	1.92
warehouse_2	95 %	1.98
refinery_3	88 %	2.25
shapes_translation	90 %	1.87
shapes_rotation	80 %	2.6
boxes_translation	61 %	2.02

Table 2.3: Validation of the proposed tracking algorithm in different sequences of two publicly available event-vision datasets.

2.4.2 Clustering

This subsection aims at validating the clustering capabilities of the algorithm presented in Section 2.3.2. Clustering algorithms aim at grouping input samples such that elements of the same cluster are more similar to each other than samples belonging to other groups (i.e., clusters). The proposed clustering algorithm gathers events with a similar spatial-temporal distribution. Thus, it considers the temporal resolution of each event to add or delete it from a cluster considering the dynamics of the event generation. Additionally, the proposed method performs *event-by-event* processing which exploits the asynchronous nature of events. This evaluation focuses on validating the capabilities of the proposed algorithm. It does not aim to compare the proposed algorithm with state-of-the-art clustering methods as they are not designed to consider the spatial-temporal resolution of events and their asynchronous generation. Despite, algorithms such as K-means [94], DBSCAN [95], and Gaussian Mixture Model Clustering [96] might be adapted to cluster event data, their implementation, adaptation, and validation to process single events is out of the scope of this Ph.D. Thesis.

The algorithm validation was divided into two parts. The first evaluation consisted of validating the method's capability to adapt to the event generation. The second validation focused on evaluating the noise rejection capability of the algorithm, which was estimated by measuring the number of wrong-grouped events per cluster. In both validations, the proposed algorithm clustered event data describing the contour of different objects in a static scene. Thus, each cluster should include events triggered by a single object. This evaluation assumes that objects do not overlap in the image plane, which is the typical study case for centroid-based clustering algorithms [97].

The first validation aims at evaluating the ability of the clustering algorithm to adapt to the event generation. Despite the main goal of the proposed algorithm is to gather events with similar spatial-temporal information, clustering asynchronous events over time leads to large clusters with redundant information. This affects the clustering process as the trace of events generated by each object (see Figure 2.5-a) may overlap with events triggered by other objects in the scene producing wrong event-cluster associations. Besides, clusters with a high number of elements increase the computational load by keeping in memory a large amount of redundant information. This first validation focuses on evaluating the algorithm's capability to adapt to the event generation such that each cluster includes enough samples to describe an object in the scene. The adaptive forgetting horizon of the proposed algorithm τ^C allows discarding event samples to keep a reduced but valid representation of the object shape. A wrong selection of τ^C might lead to clustering issues. For instance, setting a small value of τ^C causes a fast event discarding when the events in the clusters do not satisfy the forgetting condition. Conversely, a large value of τ^C leads to a redundant representation of the object causing wrong event-cluster associations. In the context of this evaluation, a redundant representation corresponds to clusters with several events describing the same edges and corners of the object contour. Conversely, a non-redundant cluster includes event samples that describe the contour of the associated object without enlarging its shape with redundant information.

Two metrics were considered in these experiments; the number of valid clusters and the number of redundant events in each cluster. These metrics were obtained by using as reference an image mask I_M describing an enlarged version of objects' shapes in the image plane. The number of redundant events in the cluster was defined by the number of clustered events lying outside the mask I_M. The *shapes* sequences of the dataset [93] were used for this validation, where a DAVIS camera moved in a real scenario with different objects in its FoV. The reference masks were obtained from the grayscale frames provided by the APS device of the DAVIS camera. Each grayscale frame was processed to find the contour of each object in the scene. Afterward, contours were filled with white pixels, and the image was binarized. The binary images represented the mask I_M, where the objects' shapes were defined by groups of white pixels in the image. Additionally, the object shapes were enlarged to increase their area in I_M. In these experiments, enlarging the object shapes by 1 px reported a good trade-off between considering enough events to describe the object and avoiding clusters with redundant event information. Figure 2.3 shows an example of a grayscale image and its equivalent mask I_M.

The parameter n^C (i.e., the size of the buffer $\mathbf{B}^C$) intrinsically sets the value of τ^C as it corresponds to the timestamp of the oldest event in the buffer. In this validation,

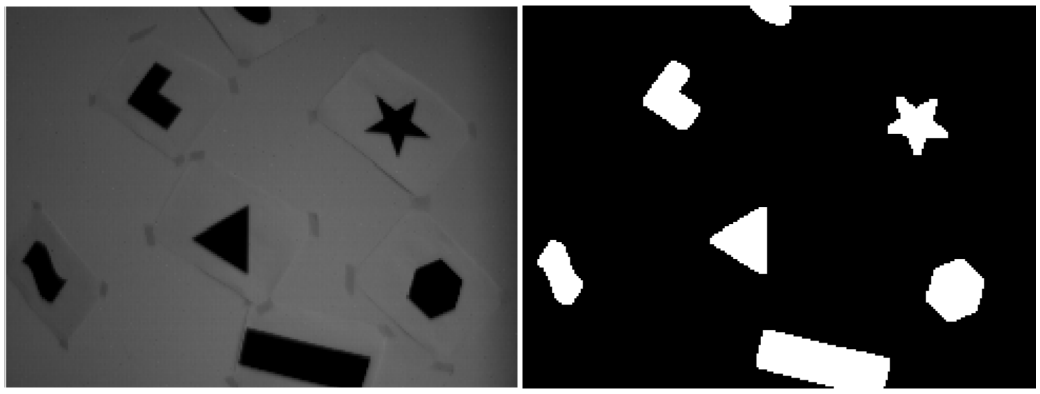


Figure 2.3: (a) A grayscale frame and (b) its equivalent reference Mask I_M from the samples of the *shapes_6DoF* dataset.

the size of the buffer was varied in the range between 1000 and 20000 to determine the value that provides the best results. Values of n^C lower than 1000 were not considered for two reasons. First, they led to clusters with an incomplete representation of the object shape (e.g., missing edges and line segments) Second, by setting $n^C < 1000$ many clusters were fastly deleted by the τ^C forgetting horizon producing unnecessary clustering errors with the dataset in [93]. Moreover, only clusters containing more than 15 events and with an area bigger than 5×5 px were considered for comparison. The proposed algorithm might create new clusters from events not associated with previous groups (i.e., clusters). This led either to clusters describing new objects in the camera FoV, or clusters of noisy events. The latter were quickly discarded by the forgetting horizon condition. The timestamps of both grayscale frames and events were used to synchronize the validation mask I_M and the output of the clustering algorithm.

Figure 2.4 shows the validation results of varying n^C in the specified range. The y-axes of the plot correspond to the average ratio of noise samples (left) and the mean number of valid clusters obtained in each experiment (right). The ratio of noise samples was defined by η_n/η_C, where η_n is the number of wrong-grouped events, and η_C is the number of events in the cluster. The Figure confirms the previous intuitions.

Large values of n^C led to a large ratio of noise samples while decreasing the number of valid clusters. The best results were obtained with buffer sizes of 1000 and 1500 events while setting $n^C > 5000$ led to a reduction in the number of valid clusters. In this case, several events triggered by different objects were gathered in a single cluster. Figures 2.5-b and d show the clustering results using two values of n^C, 1500 and 15000. The event images (i.e., Figures 2.5-a and c) correspond to the events accumulated during the last 25 ms. Figure 2.5-d shows that using a large value of n^C produced wrong event-cluster associations and cluster merging. Moreover, setting n^C to 1500 (see Figure 2.5-b) provided a valid trade-off between obtaining a good cluster representation with a small number of redundant events.

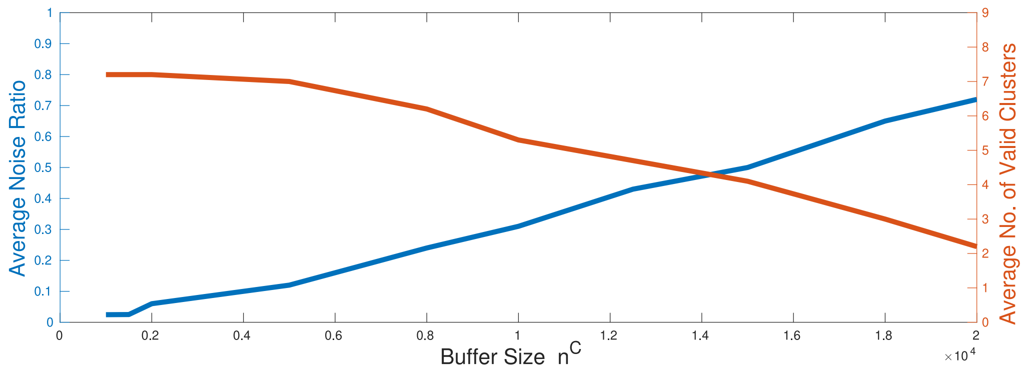


Figure 2.4: Experimental results obtained by varying the value of n^C in the range [1000, 20000] using the sequences of the *shapes_6DoF* dataset.

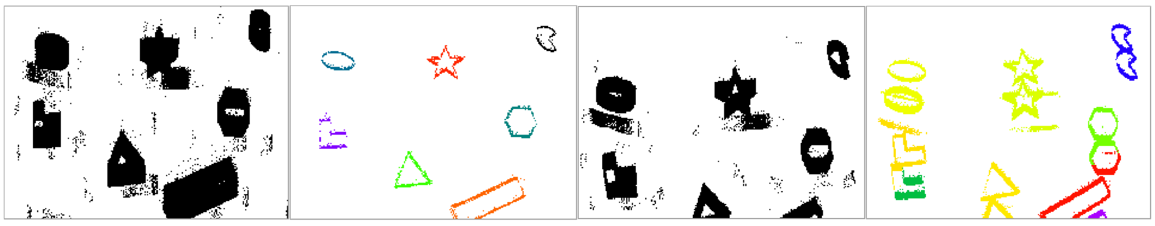


Figure 2.5: (a,c) *Event images* of accumulated events within a time window of 25 ms, and (b) their cluster representation using the proposed clustering algorithm with $n^C = 1500$, and (d) $n^C = 15000$.

The second validation aims at determining the noise rejection capabilities of the algorithm using additional datasets. Following the previous experimental results (e.g., setting $n^C = 1500$), the clustering algorithm was validated in the *shapes_translation* and *shapes_rotation* datasets. Table 2.4 depicts the mean ratio of wrong classified samples obtained with each dataset. The proposed algorithm reported an average noise ratio of 1.73 %, which was mainly due to noisy events or events triggered by other small objects in the scene. The worst results were obtained with the *shapes_rotation* dataset. In this dataset, the camera performed strong rotational motions, which increased the number of wrong-cluster samples. Figure 2.6 shows several events grouped in different clusters and their representation after being rendered over the mask I_M. Event samples out of the reference mask correspond to noise samples.

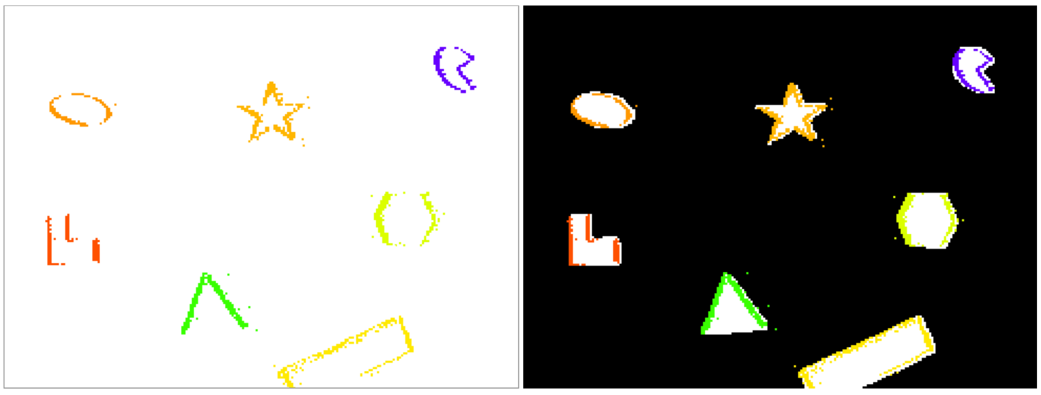

(a) (b)

Figure 2.6: (a) Clusters of events obtained by using the proposed method, and (b) their equivalent representation after being rendered over the mask I_M. The color of each event sample (i.e., colored pixels) indicates the cluster identifier.

2.4.3 APM and time filter

This Section provides a discussion about the advantages and disadvantages of using the *APM* and *Time Filter* methods (see Sections 2.3.3 and 2.3.4) for the problem of detecting moving objects with event data. These algorithms intend to provide an initial approach to distinguish events triggered by moving objects from those generated

Dataset	Average Noise Ratio	Average number of events per cluster
shapes_translation	0.9%	197
shapes_rotation	2.5 %	191
shapes_6DoF	1.8 %	196

Table 2.4: Average clustering noise ratio obtained by validating the proposed algorithm with the *shapes* sequence in [93]. The second column represents the average number of events per cluster among all the experiments.

from the scene background, assuming that dynamic objects move at a higher speed than the camera. Under this assumption, fast-moving objects trigger more events than the static background of the scene. Thus, events generated by moving objects may be detected by analyzing only their spatial-temporal information.

Despite both algorithms follow a similar approach, their design and implementation are considerably different. The *Time Filter* discards events with low spatial occurrence by comparing the time difference Δt with the threshold τ_t. Δt corresponds to the difference between the timestamp of the event and the previous time reference at the global time surface $\mathbf{S}_t$. The timestamp comparison requires only a few computational operations and provides low-latency responses. The method relies on the camera velocity estimations to dynamically adapt τ_t. The camera velocity can be retrieved from either additional sensors (e.g., Inertial Measurement Units (IMUs) and navigation devices) or specialized perception methods [98] [99]. The *Time Filter* is suitable for perception systems with low-latency requirements and very limited computational capacity due to its simplicity and adaptability to the changes in the camera motion. However, its dependence on camera velocity estimations constrains its use in platforms with reduced hardware or limited computational resources to run additional perception methods to estimate the camera velocity. Conversely, the *APM* evaluates the spatial-temporal information of events on the map Ω, which describes the regions in the image plane with a high event occurrence within a variable time window τ^A. The forgetting horizon (i.e., τ^A) allows adapting Ω to the event generation, which mainly varies due to the camera motion and the scene dynamics. Ω is updated with each incoming event. The update operation modifies the values of Ω in an $l \times l$ region around the event location. Despite the number of computational operations to update Ω is limited by l, a large number of input events considerably increases the number of operations to update the APM. Empirical results (see Chapter 5) show that the method performs well in applications where the camera triggers less than 0.1 Mevent/s using l=10. However, adaptively sampling event approaches such as the method proposed in Section 5.3.2 and the framework in [100] may be used to reduce the number of events to process while keeping the method's performance (see Chapter 5). The *APM* is suitable in applications where no additional sensing information is available to distinguish events triggered by dynamic objects moving faster than the camera. However, it requires additional pre-sampling approaches when the number of events to process is higher than 0.1 Mevent/s.

The previous considerations indicate that the method selection to distinguish events triggered by dynamic objects depends on the operational requirements (e.g., the number of triggered events per second) and the availability of additional sensing information. Chapter 4 and Chapter 5 describe two different perception methods related to the problem of detecting moving objects in different robotic applications. Chapter 4 describes a *sense-and-avoid* perception system that uses the *Time Filter* to detect dynamic objects while running on board a large-scale flapping-wing robot. Moreover, Chapter 5 presents a surveillance system that integrates the *APM* algorithm in a perception pipeline to detect intruders in unstructured scenarios. Finally, it is worth mentioning that the proposed algorithms perform under the assumption that dynamic objects move with a higher velocity than the camera. However, more complex approaches (e.g., [101]) are required to provide a general solution to fully address the problem of detecting moving objects with event cameras.

2.5 Conclusions

Event cameras represent a new paradigm for computer vision and robotic perception. The event representation limits the direct use of event information with traditional computer vision methods. Although several approaches have been proposed to convert the event stream into frames, these representations do not fully exploit the asynchronous nature of event cameras. Conversely, *event-by-event* approaches allow individual event processing offering asynchronous algorithm responses. This Chapter presents a set of *event-by-event* algorithms for low-level event-based perception. The algorithms have been carefully designed to exploit the spatial-temporal information of event data while providing fast responses.

This Chapter presents four *event-by-event* processing methods for low-level perception. The *Tracking* and *Clustering* algorithms provide event-based solutions for feature tracking and event clustering. Both algorithms have been validated with publicly available datasets showing remarkable results. They are integrated in more complex perception pipelines in Chapter 4 and Chapter 5. Moreover, the *APM* and *Time Filter* methods propose two tools to distinguish events triggered by moving objects from those triggered by the scene background. Both methods assume that dynamic objects move with a higher velocity than the camera. Thus, these methods do not intend to provide a general solution to segment events triggered by dynamic objects. A motion segmentation algorithm requires more sophisticated methods [101] [102], which are more computationally expensive than the algorithms described in this Chapter. Instead, the proposed methods were designed for being integrated with perception systems that require distinguishing events triggered by moving obstacles while satisfying some velocity conditions [103] [104] [60] [59].

The future work focuses on exploring additional low-level event perception algorithms such as novel visual descriptors or *event-by-event* optical flow algorithms. Although the number of research works on event-vision has considerable growth during the last decade, the literature reports few event-based low-level methods compared to the number of high-level perception algorithms for SLAM [105] [106], pattern recognition [107] [108] [109], and video reconstruction [87] [88], among others. Moreover, future work includes the development of an *event-by-event* fast and robust method to segment events triggered by moving objects. Despite some works have reported valid solutions for moving object event segmentation [101] [102], few methods follow an *event-by-event* processing approach [84].

Chapter 3

Event-based Visual Guidance for Aerial Robots

3.1 Introduction

The interest in performing fast and agile maneuvers with aerial platforms has increased during the last decades [110]. However, these types of maneuvers entail relevant challenges for the onboard perception sensors and systems. First, perception methods should provide fast responses to feed the onboard controllers that guide the robot along the maneuver. Second, the perception sensors should be robust to the fast motions of the robot to avoid issues such as motion blur in frame-based cameras. Third, the onboard perception hardware should satisfy the payload limitations of the aerial platforms. The challenges mentioned above suggest using novel sensors and perception methods for the guidance of fast and agile aerial robots.

Vision-based perception together with visual servoing provide a suitable solution for the guidance of aerial robots. Visual servoing approaches have been adopted in different guidance solutions [111] [112]. In particular, the Image-Based Visual Servoing (IBVS) method uses mainly visual information in the image plane to guide the robot toward a goal configuration without requiring additional pose information. Although IBVS methods have reported outstanding results for aerial robot guidance [113] [114], their adaptability for fast maneuvering is prone to the intrinsic limitations of frame-based cameras (e.g., low dynamic range and motion blur). Event cameras overcome those limitations by providing a high dynamic range, low power consumption, and robustness to motion blur, making them a suitable solution for visual servoing. Nevertheless, the literature lacks approaches exploring the use of event cameras for visual servoing that precedes the research described in this Ph.D. Thesis.

This Chapter presents an event-based perception scheme for aerial robot visual guidance. The proposed method is initially validated in a quadrotor platform, and afterward it is extended to run on board a large-scale ornithopter and validated in diverse experimental conditions. The design of the perception modules considers the hardware and software limitations to run onboard platforms with limited computational capacity. Further, it has a biological inspiration using event-based vision, which mimics biological retinas by providing asynchronous data information due to changes in illumination in the scene. Besides, the proposed scheme includes a Time-To-Contact (TTC) trajectory approach that aims at matching the retinal separation between the current and the goal visual references. The proposed approach uses visual servoing and low-level controllers to guide the robot to the goal configuration. Although some works focus on the design of novel controllers for robot guidance, the development of these techniques is out of the scope of our objectives as we mainly focus on the event-based perception method for robot guidance. Additionally, to the best of our knowledge, **this research describes the first visual servoing approach that integrates event data as visual input**. This approach is known in the literature as Event-Based Visual Servoing (EBVS) [115].

This Chapter describes **Contribution 1**, and its experimental validation is part of **Contribution 6** of this Ph.D. Thesis. Besides, the research conducted in this Chapter motivated the development and publication of the works in [57] and [58].

This Chapter is organized as follows. Section 3.2 briefly summarizes the main works in the topics addressed in the Chapter. The general scheme of the proposed guidance method is presented in Section 3.3 including event-based perception algorithms together with a visual servoing approach for robot guidance. Section 3.4 describes the event-based line detector algorithm. The event-based line tracking algorithm is presented in Section 3.5. The EBVS approach including a bioinspired time-to-contact trajectory algorithm is described in Section 3.6. The experimental results of the validations performed in a quadrotor and in an ornithopter robot are presented in Section 3.7. Finally, Section 3.8 describes the conclusions of the Chapter.

3.2 Related work

Visual guidance methods for robotics integrate perception algorithms and control methods to perform the guidance task. Several methods extract visual information from the scene to feed a visual servoing approach that guides the robot toward its final configuration. Visual Servoing strategies are divided into Pose-Based Visual Servoing (PBVS) and Image-Based Visual Servoing (IBVS). PBVS uses visual information to compute the robot's 3D pose and exert the guidance action. This method has been mainly used in applications using robot manipulators [116] [117]. Moreover, most of the visual servoing approaches for mobile robots rely on IBSV strategies as they do not require computing the pose information for robot control. In the context of aerial robotics, several works have proposed different IBVS strategies. The work in [111] proposes an IBVS method to control the 3D translational motion and yaw rotation of a quadrotor that works in both image and Cartesian spaces. Simulation experiments in both perturbed and nominal conditions were performed to validate the proposed approach. The method in [112] describes a guidance approach based on IBVS for a multirotor with a manipulator. It adopts a passive-based controller for velocity control to guide the robot toward the goal using visual information of the scenario. An IBVS approach for grasping tasks with micro aerial vehicles is proposed in [113]. The method considers planar systems for the grasping maneuvers as they predominantly occurred in the longitudinal plane. The method is validated through simulation and experiments with a real quadrotor. Despite the previous IBVS works report successful results for visual guidance, the intrinsic limitations of frame-based cameras such as the reduced dynamic range, fixed frame rate, and sensitivity to motion blur limit the application of IBVS methods for fast guidance maneuvering. Event cameras offer several advantages to deal with the aforementioned issues. Several event-based perception methods for robotics have been recently reported in the literature [16], which suggests event cameras as a suitable perception alternative for the guidance of fast and agile aerial robots.

In the last few years, several works have proposed event-based methods for robot perception. However, few of these works have tackled the problem of visual guidance in robotics. The work in [118] uses a Convolutional Neural Network (CNN) in the context of a predator/prey scenario. A CNN receives grayscale frames and *event images* to detect the robot pray. The CNN outputs the steer commands to guide the predator robot toward the prey keeping a minimum distance between them. The work in [115] proposes an Eye-In-Hand visual servoing approach for a robotic manipulator. The method uses an event camera mounted on the end effector of the robot to detect corner features of a reference pattern in the scenario. Event-based features feed an EBVS approach that guides the robot toward a suitable configuration to perform grasping tasks. The work in [119] presents an autonomous Micro Aerial Vehicle (MAV) landing approach based on event optical flow from a single event camera. Optical flow information is used as a reference to guide the robot toward the desired landing location while decreasing the descendent speed. The work includes a comparison between some frame-based methods and their event-based solution highlighting that the proposed solution provides higher accuracy. Authors in [120] propose a robot pursuit application using two event cameras. A CNN detects the moving object (a leader robot) and uses its information as a reference to guide the follower robot in the pursuit task. Recently, the work in [121] uses *event images* to estimate the position and yaw orientation of a quadrotor. The proposed scheme intends to stabilize the quadrotor using visual information while the platform experiences rotor failures during flight. To the best of our knowledge, the research conducted in this Chapter describes the first event-based guidance approach validated on an ornithopter robot.

Moreover, detecting and tracking robust visual features is crucial for the success of the visual guidance task. Lines represent simpler and robust visual features compared to other features such as corners and edges. Previous works have performed line detection from event data. The work in [122] tracks lines of a square pattern to estimate the pose of a flying drone. The method in [123] proposes an event-based implementation of the Line Segment Detector (LSD) [124]. Line segments are obtained by clustering events describing lines in the image plane. Authors in [125] detect lines by using least-squares on optical flow measurements. The line segments are computed by estimating the end-point of each detected line. The work in [126] clusters events to estimate planes containing lines in the 3D spatial-temporal space (i.e., $x - y - t$). The method reports remarkable results although it is prone to errors by detecting lines describing rotational motions. Despite previous approaches provide valid solutions for line detection, most of them were validated in simple scenarios where the camera performed either small motions or none. Only the work in [122] performs line detection with a moving event camera mounted on a quadrotor. In the context of visual guidance, this Chapter proposes an event-based line detection and a tracking method to provide fast visual references for robot guidance. These algorithms are specifically designed to cope with the hardware issues occurring in aerial platforms with very limited payload (e.g., small multirotors and ornithopters). Besides, the methods perform *event-by-event* processing differently from the majority of the event-based algorithms previously described. The robustness of the methods is evaluated in challenging scenarios with a variety of illumination conditions. Initially the proposed method is validated in a multirotor platform performing landing trajectories. Afterward, the method is adapted to execute on a large-scale ornithopter and validated in several guidance experiments while the robot performs forward flight.

3.3 General scheme

This section briefly describes the proposed guidance scheme based on event vision. It assumes a robot mounting an event camera that collects perception information of the environment. The method guides the robot toward a goal pose defined by a reference pattern, which is assumed always within the camera's Field of View (FoV) during the guidance maneuver. The reference pattern is defined by a group of straight lines in the image. Line segments are geometric features commonly used for robot perception [127] [128]. They define several shapes and contours of the scene structure and offer a richer geometric structure than features such as corners. The pattern is defined offline, e.g., knowing its coordinates at the goal location, or selected online while the robot navigates. Figure 3.1 shows a general diagram of the guidance approach.

The proposed scheme detects and tracks the lines defining the reference pattern using event information. The event camera triggers events due to the changes in illumination in the scene. In this case, events are mainly caused by the camera motion while approaching the goal. The event perception algorithm includes two main modules: a line detector and a line tracker (see Figure 3.1). The first detects lines from *event images* using the line representation on the Hough Space [129], while the second performs line tracking using a Kalman Filter (KF) combining an *event-by-event* processing method and an *event image* approach.

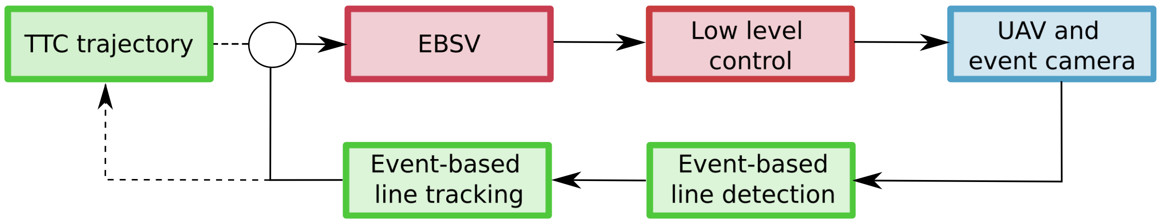


Figure 3.1: General scheme of the proposed event-based guidance approach. Green blocks represent the perception modules, while red blocks the control modules. The TTC trajectory module is optional, and its integration depends on the robot platform.

A visual servoing approach computes the velocity reference for robot guidance. It is based on an Event-Based Visual Servoing (EBVS in Figure 3.1) that is inspired by the well-known IBVS. The velocity reference feeds a low-level controller that operates over the aerial robot actuators. The controller selection varies depending on the robot platform, in the experiments of this Chapter: a quadcopter and a flapping-wing robot.

Finally, line information is optionally used to provide time-to-contact trajectories. The *TTC trajectory* module provides the line references to guide the robot towards the goal within a specific time using *tau-theory*. This module is optional, as the controlled platform needs to be capable of following the trajectory references within the specified time interval. Otherwise, a fixed-line reference (e.g., the initial configuration of the pattern) replaces the module, and the EBVS computes the velocity commands to guide the platform toward the goal.

3.4 Line detection

Line detection is performed using the Hough transform [129]. Under this representation, each straight line with parameters (m, b) is described by the point (θ, ρ) in the Hough space. The detector receives as input binary *event images* (denoted as I_e in this Chapter) which are processed for line extraction. The method samples pixels in I_e to the Hough space by $r = x \cos\theta + y \sin\theta$, where (x, y) represents the pixel coordinates and θ is ranged in $[0, 2\pi]$. Thus, each sampled pixel leads to multiple line hypotheses in the Hough space. To reduce computational costs only pixels describing an event in I_e are considered for sampling. In the work [130], line hypotheses with high occurrence in the Hough Space correspond to lines, however, many of these hypotheses could represent the same line depending on the Hough space resolution. The proposed method clusters high-occurrence hypotheses to avoid multiple line representations while providing a better approximation to the line parameters.

The set of clustered lines defines the set of detected lines $L_C = [l_1, \ldots, l_m]$, each one with a centroid (ρ, θ). This set is used as input for line tracking in the next Section. It is worth mentioning that the line detector is compatible with intensity images, as it receives as input standard frames. However, in this work, the method is fed only with *event images* to provide a full event-based method for line detection and tracking.

3.4.1 Event coordinates undistortion

Image undistortion focuses on removing the deformations on the image produced by the lens distortion. Undistorting algorithms map the coordinates of distorted pixels to their undistorted equivalent in the image. This procedure is particularly relevant to detect straight lines as many of them are deformed by lens distortion (i.e., radial and tangential). The proposed approach undistorts events coordinates (i.e., $\mathbf{x}$) by using the distortion coefficients $D = [k_1, k_2, k_3, k_4, k_5, k_6, p_1, p_2]$, and the camera matrix K_c:

$$K_c = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.1}$$

where f_x and f_y represent the camera focal length in pixels, and c_x and c_y are the optical center. Contrary to the frame-based undistortion algorithm in [131], the proposed method undistorts single events for *event-by-event* processing. The process of removing rectilinear lens distortion from event coordinates is described by the following equations:

$$\bar{x}' = [\bar{x} - 2p_1\bar{x}\bar{y} - p_2(r^2 + 2\bar{x}^2)]\alpha, \tag{3.2}$$

$$\bar{y}' = [\bar{y} - p_1(r^2 + 2\bar{y}^2) - 2p_2\bar{x}\bar{y}]\alpha, \tag{3.3}$$

$$r^2 = \bar{x}^2 + \bar{y}^2, \tag{3.4}$$

$$\alpha = \frac{1 + k_4r^2 + k_5r^5 + k_6r^6}{1 + k_1r^2 + k_2r^5 + k_3r^6}, \tag{3.5}$$

where $(\bar{x}, \bar{y})$ are the calibrated coordinates of the event, and $(\bar{x}', \bar{y}')$ are the calibrated and undistorted event coordinates. Afterward, the undistorted event coordinates (x', y') are mapped to the image plane $(\bar{x}', \bar{y}')$ by using the camera matrix. In summary, the process of undistorting event coordinates is described by the following steps:

1. Transform the event coordinates $\mathbf{x}(x, y)$ to the calibrated coordinates $(\bar{x}, \bar{y})$ using K_c^{-1}, the inverse of the camera matrix.
2. Undistort the calibrated event coordinates by using Eqs. (3.2) and (3.3).
3. Map the calibrated and undistorted event coordinates to the image using K_c.

It is worth mentioning that this approach does not perform an intensity interpolation mapping (e.g., bilinear voting) to reduce the computational cost of the undistortion process. Despite this generates some empty spaces in undistorted *event images*, it does not considerably affect the line detector based on the Hough transform. Furthermore, the undistortion process is performed at initialization only by mapping all possible event coordinates to their undistorted equivalent $\mathbf{x} \mapsto \mathbf{x}'$. The mapping is locally saved and queried each time an incoming event is received from the event stream.

3.5 Line tracking

The proposed line tracker uses event information as input to track lines represented in the Hough space. The set of tracking lines is represented by the list $L_T = [l_1, \ldots, l_n]$ with n the number of lines defining the reference pattern. The line tracker fuses tracking information from two modules: an *event-by-event* line tracker and an *event image* line tracker. For each tracked line, a KF combines the information from both modules. The *event-by-event* tracking estimations are used for KF *Prediction* while those from the *event image* correspond to the KF *Update*. To avoid statistical inconsistency, events are subsampled such that each tracker is fed with different events. The gains γ_{ee} and γ_{ei} (s.t. $\gamma_{ee} + \gamma_{ei} = 1$) define the amount of input events assigned to the *event-by-event* and the *event image* tracking modules respectively.

3.5.1 Event-by-event line tracker

The *event-by-event* line tracking approach is inspired by the work in [122], it includes additional mechanisms to enhance tracking robustness while keeping a similar computational burden. Each tracking line l is defined by the tuple (θ_l, ρ_l, P_l), where (θ_l, ρ_l) defines the line in the Hough Space, and P_l is a set of prototype events with their coordinates distributed along l in the image plane ($P_l = [\mathbf{x}_1, \ldots, \mathbf{x}_{N_P}]$). Further, the coordinates of events in P_l are projected to the Hough space and buffered in B_l keeping current and last prototypes in the Hesse normal form. P_l and B_l are bounded to N_P and N_B to prevent line over-representation.

The line tracker is initialized by the set of lines L_0 describing the reference pattern. Thus at initialization, the set of tracking lines is $L_T = L_0$. Then, the set of prototypes P_l is initialized with input events. Events are assigned to a line $l_i \in L_T$ based on their distance in the Hough space. Each coordinate $\mathbf{x}_e$ of event e is represented in the Hough space by (θ_{l_i}, ρ_e), where $\rho_e = x_e \cos\theta_{l_i} + y_e \sin\theta_{l_i}$, and θ_{l_i} is the angle parameter of line l_i. If the distance between $(\theta_{l_i}, \rho_{l_i})$ and (θ_{l_i}, ρ_e) is lower than η_H, e is assigned to the list of prototypes P_{l_i}, and (θ_{l_i}, ρ_e) to B_{l_i}. Besides, when $|P_{l_i}| > 1$, with $|\cdot|$ the cardinality of the set, e is appended to P_{l_i} if the distance between $\mathbf{x}_e$ and the nearest element in P_{l_i} is greater than the distance threshold η_C. This additional condition ensures a better distribution of prototypes by separating them by a distance η_C along the line. The previous initialization procedure iterates until P_{l_i} of each line l_i contains N_p prototype events. After prototype initialization, the line parameters $(\theta_{l_i}, \rho_{l_i})$ are updated by the centroid of the prototypes in B_{l_i}.

Algorithm 3 summarizes the functioning of the *event-by-event* line tracker after the line initialization process. Each new event e_j is processed by the *event-by-event* line tracker by evaluating the consistency of e_j with every line $l_i \in L_T$. The evaluation follows the procedure to assign prototype samples to P_{l_i}. This consistency evaluation is equivalent to a data association process by assigning new events to the elements in L_T. If event e_j is associated with more than one line, it is discarded to avoid line misalignment. This multiple association case typically occurs with events triggered at line intersections. If e_j is associated with only one $l_i \in L_T$, it is assigned to l_i. Afterward, if the distance between $\mathbf{x}_{e_j}$ and the nearest prototype P_{l_i} is higher than η_C, the event is discarded as being considered a spurious sample. Otherwise, P_{l_i} and B_{l_i} are updated with e_j. That is, the event e_j replaces the prototype sample, and its equivalent in the Hough space updates B_{l_i}. Figure 3.2 shows an example of three lines described by several prototypes and their equivalent representation in the Hough space.

The previous step updates as well the line parameters $(\theta_{l_i}, \rho_{l_i})$ with the centroid of the prototypes in B_{l_i}. This operation corresponds to the *Prediction* stage of the KF. The centroid update requires few operations, adding the contribution of the new event to the current centroid, and dividing the result by the new size of the buffer B_{l_i}. A similar operation updates $(\theta_{l_i}, \rho_{l_i})$ when removing elements of the buffer. Element removal occurs either when the buffer size is equal to the bound N_B, or when the timestamp of old samples is lower than τ^L. The variable forgetting horizon τ^L follows the working principle of τ^F and τ^B (see Chapter 2). τ^L corresponds to the timestamp of the oldest event in the global buffer B^L, which is updated with the timestamp of each analyzed event and it is bounded by n^L.

Algorithm 3: Asynchronous event-by-event line tracking

```
Input: $e_j$
Output: $l_k$
$candidates \leftarrow 0$
for $l_i \in L_T$ do
    if $consistentWithLine(e_j, l_i)$ then
        $candidates \leftarrow candidates + 1$
        $k \leftarrow i$                                   ▷ Index of the last consistent line $l_i$.
    end
end
if $candidates = 1$ then
    $P_{l_k} = getPrototypeList(l_k)$
    if $validUpdatePrototype(e_j, P_{l_k})$ then
        updateSamples($e_j$, $P_{l_k}$, $B_{l_k}$)
        $l_k \leftarrow$ updateLineComponents($B_{l_k}$)     ▷ Update $(\theta_{l_k}, \rho_{l_k})$.
    end
end
else
    $l_k \leftarrow \emptyset$
end
$\tau^L \leftarrow$ updateGlobalBuffer($e_j$)
return $l_k$
```

3.5.2 Event-image line tracker

This module tracks lines in the Hough space using the reference provided by the line detector of Section 3.4. The algorithm receives as input the set of lines candidates L_C. Each input line $l_k \in L_C$ is associated with the elements of L_T. Line association consists of evaluating the occurrence of line l_k in a rectangle area A_u around each line $l_j \in L_T$ in the Hough space. A_u is defined by $l_\theta \times l_\rho$. If l_k is associated with only one line, it directly updates the line associated with it. If l_k is associated with more than one element of L_T, it updates the line closer to l_k. Otherwise, l_k is discarded. The proposed line tracker corresponds to the *Update* stage of the KF by tracking the set of lines L_C provided by the detector described in Section 3.4.

The *event-image* line tracker can perform either synchronously or asynchronously depending on the *event-image* I_e generation. As it is described in Section 3.4, I_e is

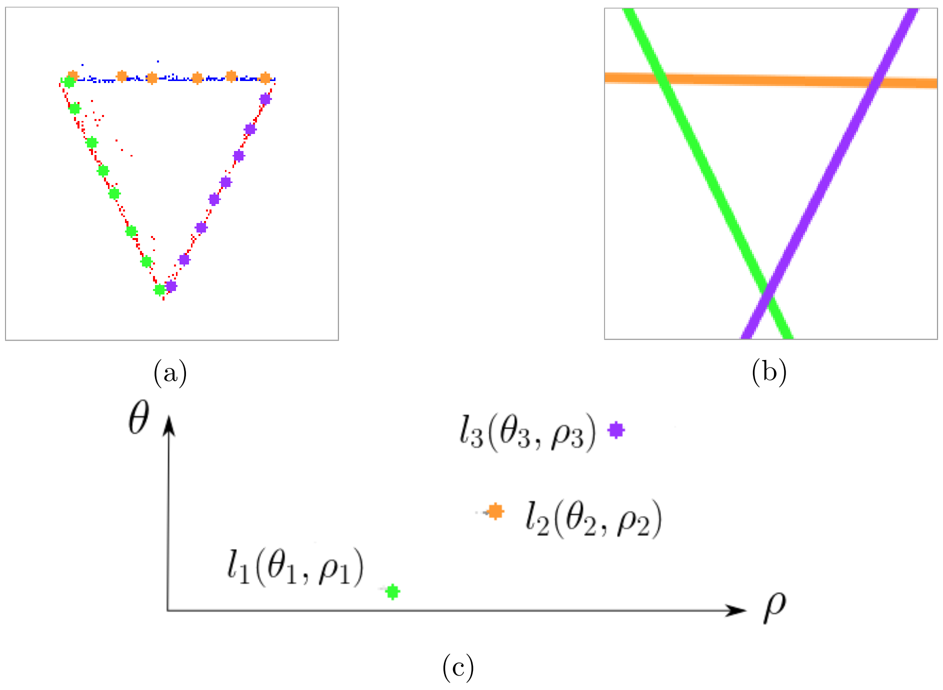


Figure 3.2: The *event-by-event* line tracker in one example: (a) processed input events (red and blue) and prototype coordinates of each line P_{l_i} (big green, orange, and purple points); (b) tracked lines in the image plane; (c) event Hough space representation showing the tracked lines (green, orange, and purple).

produced by accumulating events either by time or by a fixed number of events. A constant time window leads to a synchronous I_e generation while building I_e with a fixed number of events produces asynchronous *event-images*. Moreover, it is worth mentioning that this tracker is compatible with lines detected from intensity images as it performs line tracking from frames instead of single events.

3.6 Visual servoing

The proposed visual servoing approach focuses on reducing the error between visual features given by the desired line configurations L_D and the current line tracking estimations L_T. Thus, it follows an image-based visual servoing approach instead of a position-based visual servoing scheme that reduces the position error in the 3D space.

In general, the control law that computes the error velocity using visual features is given by:

$$\nu(t) = -K_p J_v^* e_v(t), \tag{3.6}$$

where K_p is a positive definite diagonal matrix, $e_v(t)$ is the error between the desired and current visual features, and J_v^* is the pseudoinverse of the Jacobian J_v. This Jacobian describes the variation of the visual features with respect to the camera velocity, and it is also known as the interaction matrix. The previous notation is general and applies to different types of visual features. Among the most common visual features there are feature points $p(x, y)$, lines $l(\theta, \rho)$, and ellipses $s(x, y, \mu_1, \mu_2, \mu_3)$. The structure of the Jacobian in Eq. 3.6 varies depending on the type of feature. For instance, the Jacobian for line features is described as follows:

$$J_v^l = \begin{bmatrix} \lambda_\theta s(\theta) & \lambda_\theta c(\theta) & \rho\lambda_\theta & -\rho s(\theta) & -\rho c(\theta) & -1 \\ \lambda_\rho s(\theta) & \lambda_\rho c(\theta) & \rho\lambda_\rho & -c(\theta)(1+\rho^2) & s(\theta)(1+\rho^2) & 0 \end{bmatrix}, \tag{3.7}$$

where $s(\cdot)$ and $c(\cdot)$ stand for *sine* and *cosine* respectively, and λ_θ and λ_ρ are defined by:

$$\lambda_\theta = \frac{c^{-1}(\theta) - Bs(\theta)}{D}, \tag{3.8}$$

$$\lambda_\rho = \frac{\rho(As(\theta) + Bc(\theta)) + C}{D}, \tag{3.9}$$

where $AX + BY + CZ + D = 0$ defines the plane that contains the line $l(\theta, \rho)$. Moreover, the equivalent Jacobian for point features is given by Eq. 3.10:

$$J_v^p = \begin{bmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xy}{f} & -\frac{f^2+x^2}{f} & y \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & \frac{f^2+y^2}{f} & -\frac{xy}{f} & -x \end{bmatrix}, \tag{3.10}$$

where f represents the focal length and Z is the depth to the point. Feature points $p(x, y)$ can be extracted directly from the image or computed by the intersection of lines. The latter is preferred in this work as lines provide robust features than points in the image. The Jacobians of Eqs. 3.10 and 3.7 are used in the experiments of Section 3.7 for visual guidance of aerial robots.

The interaction matrix J_v of Eq. 3.6 is computed by concatenating row-wise Jacobians J_{v_i} of each individual feature. In the case of point and line features, each visual feature provides a Jacobian J_{v_i}. At least three features are required to obtain full rank J_v. However, additional features can be used to enhance accuracy and avoid singularities.

Both J_v^l and J_v^p Jacobians depend on the depth Z, which can be estimated using visual information or obtained from external sensors. For instance, in [132], the depth is computed by deteriorating the optical flow of point features in the image. Conversely, Time of Flight (ToF) cameras provide an approximation of the depth of each pixel in the image. A simpler hardware solution is the use of 1D range sensors when the scene surface is assumed as flat and parallel to the camera. This last option could be feasible for the landing of Vertical Take-Off and Landing (V-TOL) platforms on flat surfaces.

3.6.1 Time-to-contact velocity guidance

IBVS and EBVS approaches for robot guidance hold for platforms capable of exerting the required velocities such that the visual error tends to zero. This is feasible for quadrotor platforms moving in different directions, and flapping-wing robots under very specific conditions (i.e., a slow guidance maneuver describing mainly a forward motion). Despite these visual servoing approaches represent feasible guidance solutions, they do not intrinsically include a trajectory guidance strategy.

This section proposes a guidance strategy based on *tau-theory*. A detailed explanation of *tau-theory* concepts can be found in Chapter 6 including its main assumptions and mathematical expressions. This section describes only the most relevant aspects of this theory. It postulates that animals use a set of simple actions and the time τ to guide the majority of their intended movements. τ represents the time that an observer would require to make contact with a surface, and corresponds to a first-order approximation of the TTC for a given gap. The proposed velocity guidance approach uses the tracked lines L_T to compute the robot velocity commands such that the set L_T follows the desired TTC trajectories in the image plane. The difference between the desired line configuration L_D and the initial tracked lines L_0 defines the set of gaps to close using *tau-theory.* The adopted *intrinsic-Tau* guidance closes the main gap within a defined time (see Eq. 6.2), and zero velocity and acceleration. Assuming the camera is calibrated (i.e., K_c is available) and the reference pattern geometry at the goal is known, the desired line features L_D are computed by projecting the reference pattern in the image plane. This step is performed once before starting the guidance maneuver.

The *intrinsic-Tau* guidance approach for a single gap is described as follows. For a given gap $\chi(t)$, $\tau(t)$ provides a first order approximation of the time-to-contact computed as $\tau(t) = \chi(t)/\dot{\chi}(t)$ (see Eq. 6.1), where $\dot{\chi}(t)$ represents the gap closing rate at time t. The gap $\chi(t)$ is defined always negative and it closes when $\dot{\chi}(t)$ is positive.

The previous gap definition is valid for each line feature at time t. Hence, a gap distance $\chi(t)$, its closure velocity rate $\dot{\chi}(t)$, and its closure acceleration $\ddot{\chi}(t)$ are defined for each feature. The proposed approach focuses on maneuvers starting at a still initial position (i.e., $\chi(0) \neq 0, \dot{\chi}(0) = \ddot{\chi}(0) = 0$), accelerating, and decelerating to the goal at time T_g (i.e. $\chi(T_g) = \dot{\chi}(T_g) = \ddot{\chi}(T_g) = 0$). These are typical conditions for landing and perching maneuvers starting from a hovering state. Following the approach in [133], the *intrinsic-tau* guidance is computed as follows:

$$\begin{aligned}
\chi(t) &= \frac{\chi}{T_g^{2/k_\tau}}\left(T_g^2 - t^2\right)^{\frac{1}{k_\tau}},\\
\dot{\chi}(t) &= -2t\frac{\chi(0)}{k_\tau T_g^{2/k_\tau}}\left(T_g^2 - t^2\right)^{\frac{1}{k_\tau}},\\
\ddot{\chi}(t) &= -2t\frac{\chi(0)}{k_\tau T_g^{2/k_\tau}}\left[\frac{2-k_\tau}{k_\tau}t^2 - T_g^2\right]\left(T_g^2 - t^2\right)^{\frac{1-2k_\tau}{k_\tau}},
\end{aligned} \tag{3.11}$$

where k_τ defines the trajectory kinematics, and it is typically in the range $[0, 0.5)$ to ensure gap closure at time T_g (see Section 6.3.1). Moreover, all gaps (i.e., line features) close together following the concept of *tau-coupling* [134]. Under this assumption, one gap is defined as the main gap (e.g., the gap with the greater initial distance), and the rest are set as the coupled gaps. The constant κ_τ defines the relationship between the closing rates of the main χ gap and the coupled gaps ϑ_i as $\tau_\chi = \kappa_\tau \tau_{\vartheta_i}$. The evolution of the main gap is described by Eq. 3.11, while the trajectory of each coupled gap is obtained from Eq. 6.4. Chapter 6 (see Section 6.3.1) includes further details about the of *tau-coupling* strategy.

3.7 Experiments

The experimental validation is divided into three parts. First, a preliminary evaluation of the line tracker is presented. Different validation experiments were performed to evaluate the tracking error, robustness, and fast response of the proposed event-based line tracker in diverse challenging conditions. Second, the time-to-contact velocity guidance approach is validated by performing landing maneuvers with a quadrotor. The experiments include maneuvers performed at different velocities and heights, and changing the lighting conditions in the scene showing the advantages of using event data instead of intensity frames. Finally, the proposed guidance approach is validated in a large-scale ornithopter in outdoor and indoor scenarios.

The experiments were performed in the installations of the University of Seville and the testbed of the GRVC Robotics Laboratory. The testbed is equipped with 24 *OptiTrack Prime^x*13 cameras that provided millimeter accuracy for robot pose estimation. The motion capture system was used only to retrieve the pose ground truth for the experimental validation. The testbed encloses an area of $15 \times 21 \times 8$ m designed to evaluate ornithopters and other aerial robotic platforms. The outdoor scenario corresponds to the installations of the School of Engineering of the University of Seville.

3.7.1 Line tracking

First, the accuracy and robustness of the line tracking method were evaluated. The evaluation also validated the execution rates of the algorithm to confirm its fast response for aerial robot guidance. The maximum size of P_l (i.e., N_P) was set to 6, and n^L to 50 for the validation experiments. Besides, γ_{ee} and γ_{ei} were set to 0.6 and 0.4 respectively such that a higher number of events are used in the *event image*. In the following experiments, the *event image* I_e was built by accumulating $5K$ events as a trade-off between robustness and computational effort.

The accuracy of the line tracking method was evaluated by measuring the error between the tracked lines and the ground truth. Two different approaches were used to compute the mean tracking error by considering diverse line representations; pixels describing a line in the image plane and intersection points that define the lines to follow. Each approach used a different sensor to retrieve the ground truth line information; a standard camera providing grayscale frames and a motion capture system providing 3D pose information of the lines. At each validation, the reference lines to track were defined by a triangular pattern in the scenario. A DAVIS346 camera moved in front of the reference pattern in different directions while keeping the reference pattern in the camera FoV. The output from DAVIS (i.e., events from the Dynamic Vision Sensor (DVS) and frames from the Active Pixel Sensor (APS)) together with the pose estimations from the motion capture system were recorded on the same computer to guarantee temporal consistency.

In the first approach, the ground truth was obtained from the grayscale frames provided by the APS sensor of a DAVIS346. A Canny edge detector [135] was used to extract lines from the intensity frames. The output images were filtered keeping only line segments that correspond to the lines to follow. The filtered frame with the ground truth information (i.e., pixels defining the lines) was denoted as I_{gt}. The tracking error was computed as the mean minimum distance between the pixels defining a ground truth line in I_{gt} and the prototypes P_l of the equivalent line $l \in L_T$. At each validation experiment, the tracking method was initialized with the ground truth lines in I_{gt}, and fed only with events during the rest of the experiment. The mean tracking error after evaluating more than 20000 line tracks was lower than 2.23 px, which demonstrates the remarkable capabilities of the proposed method.

In the second approach, the ground truth was obtained from a motion capture system. A set of markers were located at the line intersections of the reference pattern to retrieve their 3D pose. In each experiment, the set of 3D points was projected into the image plane to obtain the ground truth references. The tracking error was defined by the distance between the projected reference points and their equivalent line intersection between the elements of L_T. A mean tracking error of 2.85 px was reported after comparing more than 30000 ground truth measurements with their equivalent tracking estimations. The results depict the remarkable capabilities of the tracking algorithm and suggest the potential of using line intersection points as feature references for visual servoing.

The second set of experiments focused on evaluating the robustness of the method in highly cluttered environments. In these experiments, objects in the scene partially occluded or overlapped the tracking pattern on the camera FoV. Three experiments were performed where different objects intersected the reference pattern: *flying objects*, *office background*, and the *testbed safety net*. For each experiment, the lifetime of the tracking lines was measured. A lifetime with a 100 % ratio means that the line was successfully tracked during the entire experiment. Table 3.1 shows the duration of the experiment and the lifetime of the lines tracked. In all experiments, the lines were robustly tracked by the proposed method. Figure 3.3 shows an example of each experiment. The first row shows the APS frames with different objects intersecting the reference pattern. The second row shows *event images* where the lines are successfully tracked using the proposed approach. The last column represents the most challenging experiment where the net triggered many events, which hindered the tracking performance.

Table 3.1: Line tracking lifetime in different cluttered environments.

	Duration (s)	**Lifetime (s)**	%
Flying Objects	15.5	14.82	95.6
Office background	35.06	32.02	91.3
Safety Net	18.48	14.58	78.9

Finally, the execution rates of the line tracking method were evaluated while running on a Khadas VIM3, a lightweight micro-computer suitable for aerial robot platforms. Three line detection and tracking approaches were evaluated in this

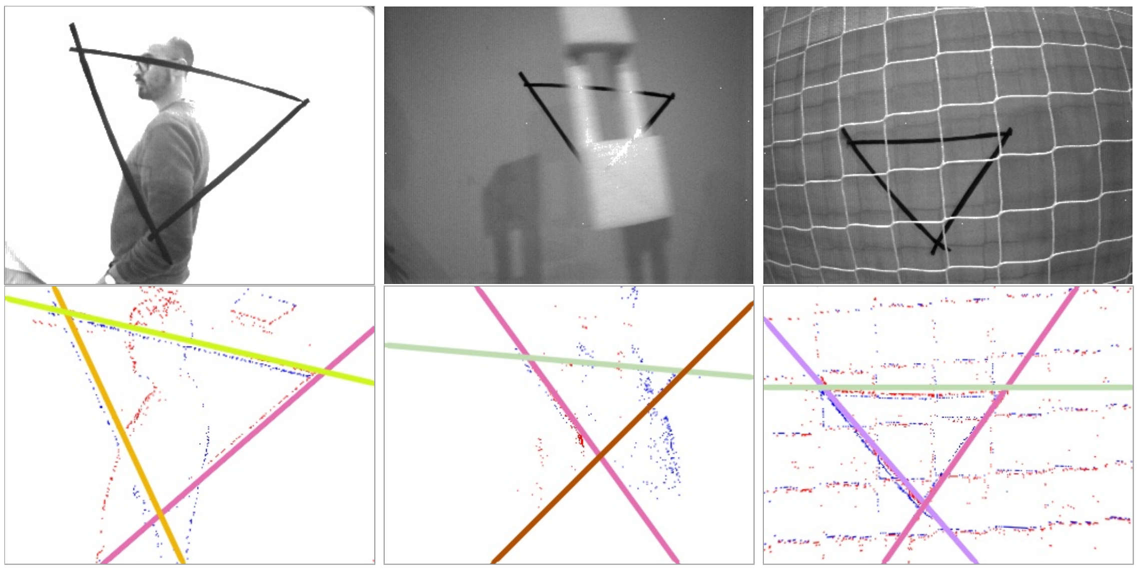


Figure 3.3: Undistorted *event images* along with the tracked lines in situations where different objects intersected the reference pattern. The intensity frames (without lens undistortion) on the first row provide a clear representation of the experiment.

validation. *Method 1* extracts lines from intensity frames using the classical Hough transform implementation of OpenCV [136] together with the frame-based line tracker of Section 3.5.2. In *Method 2*, lines are detected and tracked from *event images* using the algorithms of Sections 3.4 and 3.5.2. *Method 3* corresponds to the complete tracking method (see Section 3.5) fed with lines detected by the line detector of Section 3.4. The three methods were validated in the same experiment where four lines represented the reference pattern. Table 3.2 depicts the execution rates of the described perception methods including the computation of the time-to-contact trajectories.

Table 3.2: Excution rates of the three evaluated methods.

	Method 1	***Method 2***	***Method 3*** *(Adopted method)*
Line extraction	38 Hz.	68 Hz.	348 Hz.
Line extraction + TTC guidance	37 Hz.	67 Hz.	339 Hz.

The proposed line detection and tracking method reported an average execution rate of 348Hz and 339Hz including the TTC module, which were significantly higher than the rates obtained with *Method 1* and *Method 2*. It is worth mentioning that the proposed methods reported similar execution rates in the guidance experiments of the following sections. Finally, an empirical evaluation of the scheme scalability was performed concluding that the computational cost of running both methods (i.e. line detection and tracking, and TTC trajectory computation) grows linearly with the number of lines. Despite tracking several lines (>9) reduces the performance of the proposed method, it is an atypical case as the reference pattern is easily defined with 4 lines.

3.7.2 Multirotor landing

The proposed time-to-contact guidance scheme was validated using a quadrotor platform. The experiments consisted of performing different descendent maneuvers under diverse experimental conditions; changing the duration of the maneuver and modifying the illumination conditions of the scene. The quadrotor mounted a DJI Flamewheel F450 frame and a PixRacer autopilot. A DAVIS346 was mounted pointing downwards and a lightweight Khadas VIM3 board was used to execute the proposed scheme. Besides, a range sensor pointing downwards was used to estimate the flying altitude of the quadrotor. Figure 3.4 depicts the robot platform in one experiment. The method was implemented in C++ and ran together with the UAL abstraction layer [137] for the control of Unmanned Aerial Vehicles (UAVs). The guidance velocity commands were handled by the UAL layer that communicates with PX4 velocity-based low-level controller. The guidance experiments were performed in the GRVC testbed. The ground truth pose of the robot was obtained from a motion capture system and was used only for validation purposes.

A total of 180 experiments with descent maneuvers were performed. The experiments included different initial and goal poses, duration T_g, and lighting conditions. The parameter T_g was used to ensure that the vehicle velocity coped with the low-level controller safety bounds, $2\,\mathrm{m\,s^{-1}}$ in the selected platform. The camera retreat effect

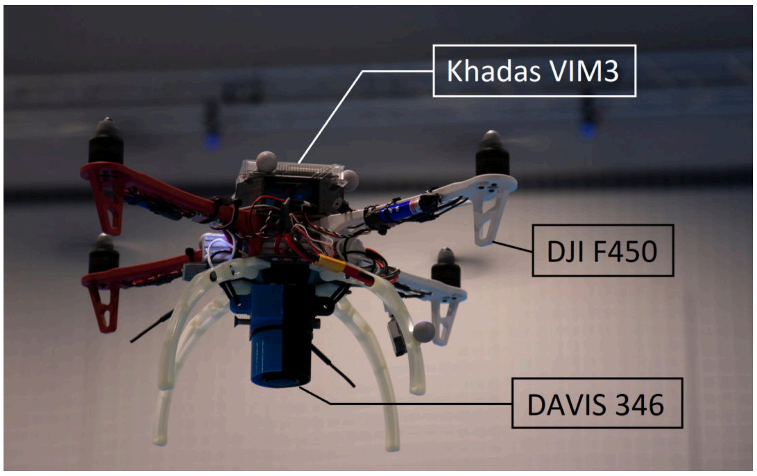
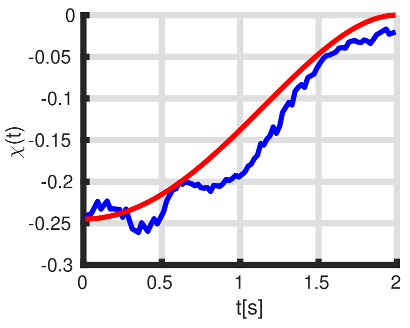


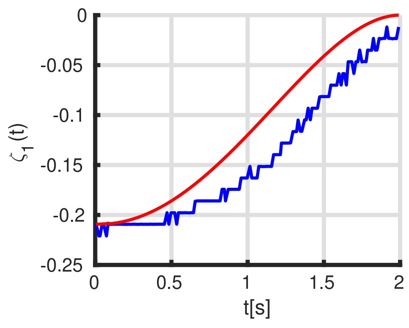
Figure 3.4: Aerial platform used in the TTC guidance experiments. The quadrotor is equipped with a DAVIS346 event camera and a Khadas VIM3 for event processing.

[132] of the IBVS was avoided by limiting the rotations between the initial and goal robot poses up to $\pi/4$. Furthermore, the methods enclosed by the guidance scheme used the same set of parameters in all experiments. The gains of the visual servoing controller were set to 0.2 in all the diagonal elements of K_p. Besides, κ was set to 1.0 and k_t to 0.5 to ensure that all coupled gaps ϑ_i close at the same time that the main gap χ with zero velocity and acceleration.

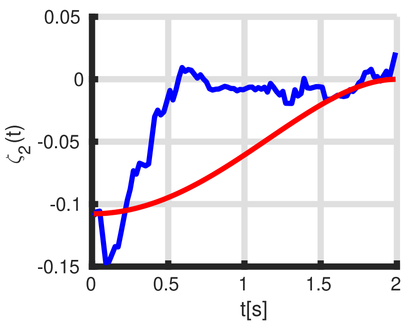
The gap evolution along a descending maneuver is shown in Figure 3.5, The robot traveled a distance of 1.5 m within a time T_g= 2 s. The maneuver required starting and finishing with zero velocity, thus the robot reached a maximum velocity close to the safety bounds (i.e., $2\,\mathrm{m\,s^{-1}}$). The reference pattern included four lines, a total of 8 gaps to close. For simplicity only four gaps are shown in the Figure: a) the main ρ gap $\chi(t)$, b) an example of the θ gap $\vartheta_1(t)$, and c) and d) the ρ coupled gaps that reported the best $\vartheta_2(t)$ and the worst $\vartheta_3(t)$ gap reduction at T_g. The vertical axis of the Figures represents the gap distance assuming a unitary focal length. The predicted gap trajectory obtained with the *intrinsic-Tau* guidance approach is depicted in red, while the evolution of the gap is shown in blue. Closing θ gaps only led to rotations in the yaw angle of the robot. These trajectories can be easily followed with low error as it is shown in Figure 3.5b). Conversely, ρ gap closure reported lower accuracy as it required spatial translations in the X, Y, and Z axes. In general, the main gap (see Figure 3.5a) was closed more accurately than the coupled gaps. Although not all ρ gaps might be closed at time T_g, the gaps were consistently reduced within the time constraints. The mean gap closing error in θ and ρ at T_g were 0.5° and 0.32 px respectively. The 3D trajectory followed by the quadrotor together with the distance between the robot and the goal position is shown in Figure 3.6a. The robot followed a smooth trajectory during the experiment. The quadrotor pose error after T_g was 10.4 cm by analyzing the data from the motion capture system. Similar results were obtained in all the maneuvers performed.



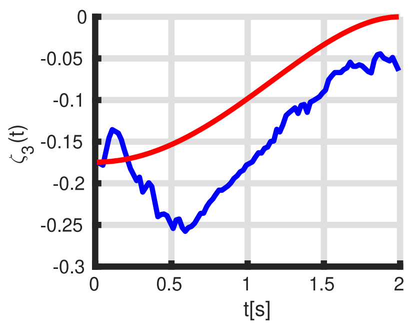
(a) Main gap.



(b) Example θ gap.



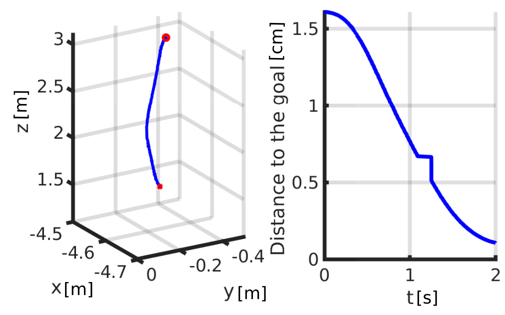
(c) Best coupled ρ gap.



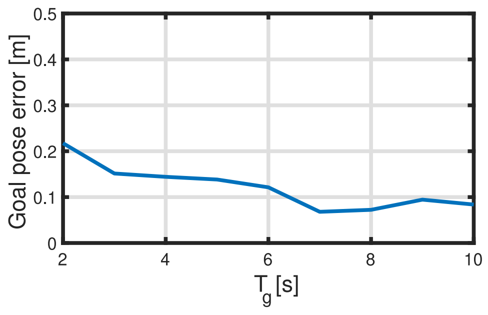
(d) Worse coupled ρ gap.

Figure 3.5: Evolution of four gaps along the maneuver. (a) main ρ gap, (b) a θ gap, (c) and (d) the best and worst ρ coupled gaps.

Additional experiments were performed by changing the duration of the maneuver T_g in the range [2,10]s. A total of 20 maneuvers were performed for each value of T_g. The same set of parameters was used for all maneuvers including the gains of the matrix K_p. Figure 3.6b shows the average robot error obtained by performing



(a)

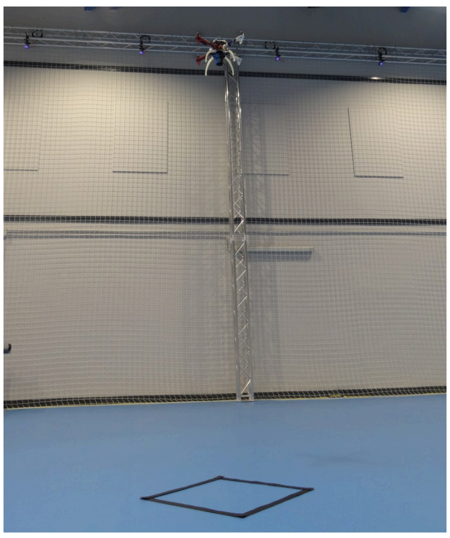


(b)

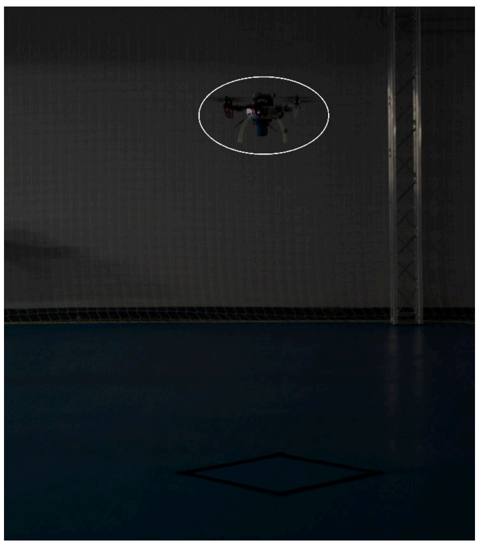
Figure 3.6: (a-left) The trajectory followed by the quadrotor during the experiment. (a-right) The robot-goal distance along the maneuver. (b) Average robot spatial error in reaching the goal pose within different values of T_g.

the same trajectory while varying T_g. The error decreased with large values of T_g. In contrast, small values of T_g reported higher errors due to the higher velocity required to exert the trajectory within T_g. The limitations in the responsiveness to the velocity commands by the controller (PX4 in this case) were more notable with low values of T_g. This error might be reduced by keeping running the visual servoing method after the finishing trajectory (i.e. $t > T_g$) such that the error is reduced during hovering.

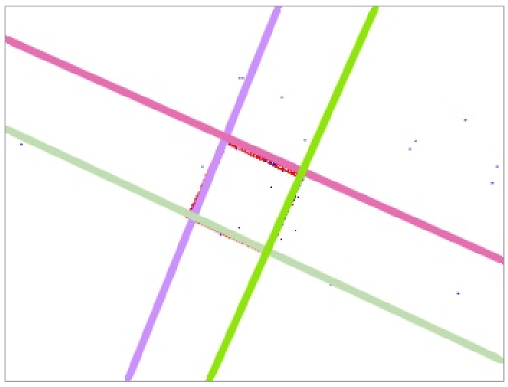
Finally, additional experiments were performed for a variety of lighting conditions. The experiments aimed at validating the performance of the proposed scheme in a wide range of illumination conditions ([30, 800] lx). The proposed method was capable of operating under these conditions due to the high dynamic range of the DAVIS346 event camera. Figure 3.7 shows two examples of the maneuvers performed in diverse lighting conditions. Figures 3.7c and 3.7d show the initial and goal line configurations for the same trajectories in both lighting conditions. The RGB image on Figure 3.7b was modified to increase the visibility of the quadrotor in the scene. The variations in the illumination conditions did not report a considerable impact on the experimental results, even in pitch-dark scenes where a high number of noise events were triggered by the event camera.



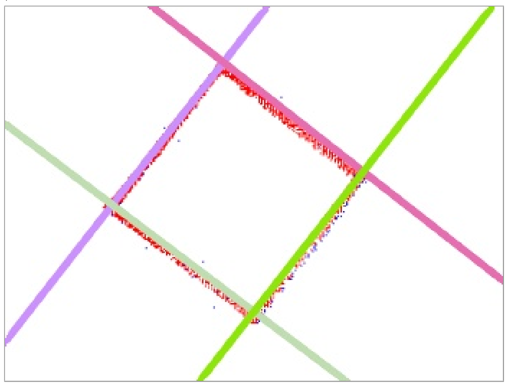
(a) Experiment with regular light.



(b) Experiment with pitch dark conditions.



(c) Line features at the initial pose.



(d) Lines features at the goal configuration.

Figure 3.7: (a,b) examples of the experiments performed by modifying the illumination conditions of the scene. (c,d) tracked lines describing the reference pattern at the start and goal configurations.

3.7.3 Ornithopter guidance

In these experiments, the proposed scheme for visual guidance was extended to run in a flapping-wing robot. The experimental validation of Section 3.7.2 demonstrates the remarkable capabilities of the proposed method for visual guidance. In this section, the method is adapted to run on a large-scale ornithopter and validated in experiments where the robot performs guidance maneuvers while performing forward flight. The validation was performed in the *E-Flap* robot [34], an ornithopter developed and manufactured by the GRVC Robotics Laboratory. The robot has a total length of 95 cm, a maximal wing span of 1.5 m, an empty weight of 510 g, and a maximum payload of 520 g (reducing maneuverability and flight time). Differently to the *E-Flap* version in [34], the robot was modified to mount a DAVIS346 event camera, and a Khadas VIM3. The sensors and onboard computer were carefully installed to keep the weight balance of the platform. The additional hardware represented an additional payload of $\sim$180 g, which is within the limits of the maximum payload of the robot.

The limitations of the EBVS approach together with the underactuated nature of *E-Flap* were considered for the validation experiments. First, the robot flight and hardware specifications influence the duration of the guidance maneuver. *E-Flap* requires a flight speed in the range [3.5, 4]$\mathrm{m\,s^{-1}}$ to maintain a constant altitude [34]. Moreover, the camera resolution and its angle of view set a maximum distance of 10 m to detect and track the reference pattern. Following these considerations, the guidance maneuvers described trajectories of $\sim$2 s in the performed experiments. Second, similar to IBVS approaches, EBVS methods require the reference pattern in the camera FoV for correct functioning, which is particularly challenging with large-scale ornithopters due to the changes in the camera translation and orientation produced by the flapping strokes. Considering the ornithopter kinematics, the tail actuation over the horizontal deflection δ_e and lateral deflection δ_r were limited in the ranges $[-30, 10]^\circ$ and $[-20, 20]^\circ$ to avoid large motions which may lead to losing the pattern in the camera FoV.

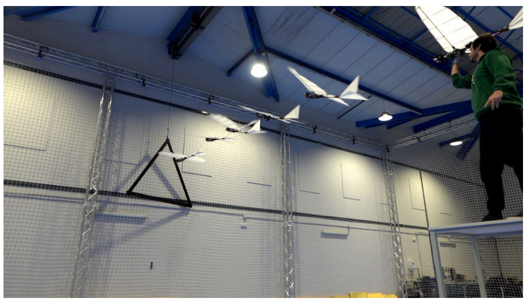
Furthermore, the ornithopter tail deflections were controlled by the method proposed in [58], which was inspired by the high gain feedback control theory [138]. The visual servoing approach aimed at minimizing the error $\nu(t)$ from Eq. 3.6. Different from the approach used in the multirotor's experiments, the interaction matrix was estimated by stacking Jacobians from point features (see Eq. 3.10). Point features were extracted from line intersections, and their mean depth was estimated as the distance between the camera and the plane containing all features [58]. It is worth mentioning that in these experiments the TTC guidance module was not included for visual guidance. Although the perception algorithm provided line references at high rates, the kinematic constraints of the platform and the limitations of the adopted controller could not guarantee that the robot followed the line trajectories provided by the TTC within T_g=2 s. Besides, the proposed TTC method assumed specific initial conditions and robot trajectories finishing with zero velocity and acceleration at T_g, which is particularly challenging with large-scale ornithopters. Further analysis of the robot's kinematics and dynamics is required to design a suitable controller to exert the TTC trajectories, which is out of the scope of this Ph.D. Thesis.

The proposed method was validated in experiments performed in two different scenarios: an indoor testbed and an outdoor scenario. The testbed scenario included a motion capture system to obtain the ground truth position of the robot which was used only to validate the experimental results. Conversely, the outdoor scene was selected to evaluate the scheme's robustness to the uncertainties presented in outdoor spaces. A total of 20 flights were performed covering different gliding and flapping maneuvers. Four types of experiments were evaluated: *Gliding* (indoor); *Flapping descending* (indoor); *Flapping straight* (indoor); and *Flapping outdoors.* Figure 3.8 shows some visual examples of experiments *Flapping descending*, *Flapping straight*, and *Flapping outdoors.*

The first part of the experiments focused on evaluating the robustness of the line tracking method running onboard *E-Flap.* The module of the linear acceleration measured by the onboard Inertial Measurement Unit (IMU) during a flight is depicted in Figure 3.9-top-left, which provides an idea of the vibrations experienced during the ornithopter flight. Figure 3.9top-right and bottom show the line tracking evolution during a guidance experiment. The proposed event-based method provided smooth line tracking despite the vibrations experienced during the flight. Additionally, the robustness of the line tracking method in outdoor scenarios was validated. Figure 3.10 shows examples of the tracked lines in both indoor and outdoor scenarios where the lighting conditions changed significantly. In all experiments, the proposed method provided remarkable robustness to the variations in the illumination conditions of the environment.

The validation results of the guidance method are summarized in Table 3.3. Two different metrics are described in the table; the Root Mean Square Error (RMSE)

(a) *Flapping descending.*


(b) *Flapping straight.*

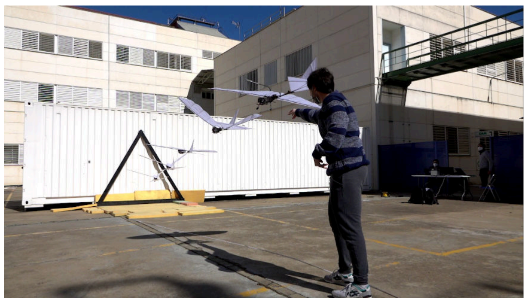

(c) *Flapping outdoors.*

Figure 3.8: Sequences of the different validation experiments performed with the flapping-wing robot: (a) *Flapping descending*, (b) *Flapping straight*, and (c) *Flapping outdoors*.

and the Normalized Root mean Square Error (NRMSE), which was normalized by $max(\mathbf{v}) - min(\mathbf{v})$. The latter provided an estimation of the control performance by comparing the final value of $\mathbf{v_x}$ and $\mathbf{v_y}$ with respect to their variation during the experiment. The experimental results reported an RMSEy of 0.64, an RMSEx of 0.31, an average NRMSEy of 8%, and an average NRMSEx of 21% in the performed experiments. The obtained error in each velocity component was small, however, the variation range was larger in $\mathbf{v_y}$ than in $\mathbf{v_x}$. This was due to the dynamics of the ornithopter and the bounded tail deflections (i.e., δ_e and δ_r), which made the lateral-directional dynamics of the robot less maneuverable than its longitudinal dynamics. Furthermore, the flapping guidance experiments provided better results than performing gliding guidance when executing descending maneuvers. In this case, the robot was more responsive to controlling the altitude generating thrust by flapping its wings. Moreover, despite the *Flapping straight* experiments were the most

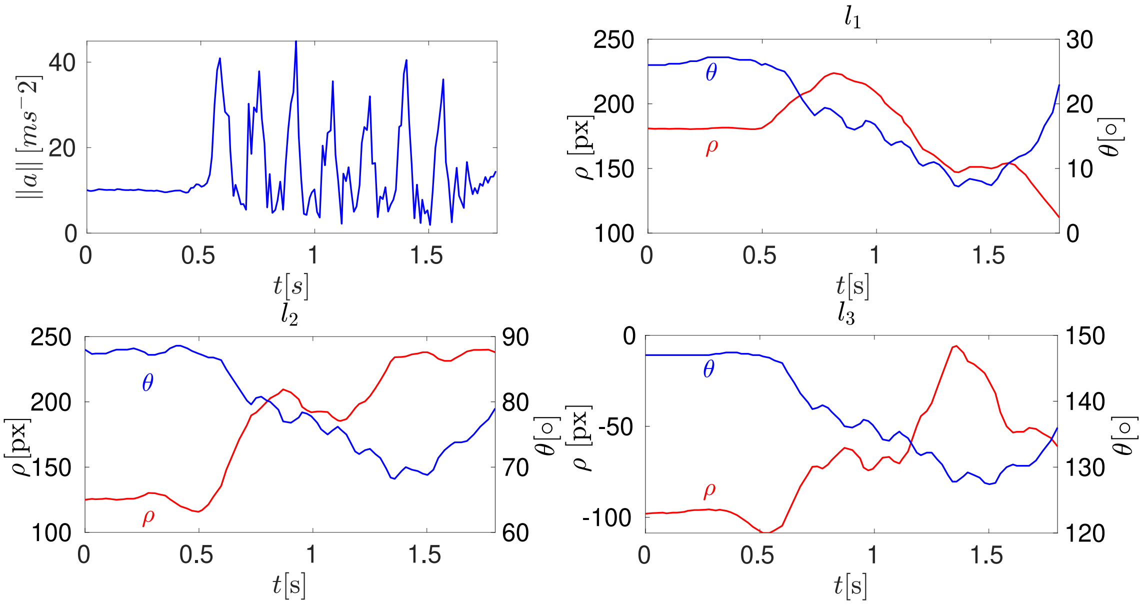
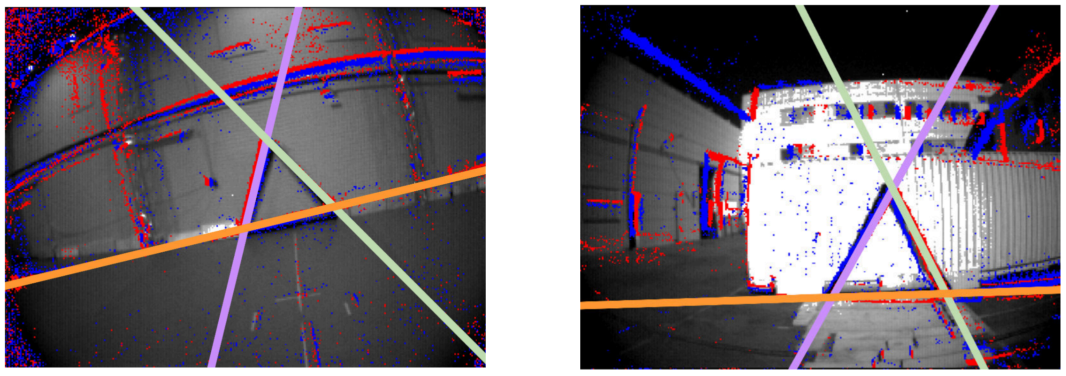


Figure 3.9: Top-left) the module of the linear acceleration registered by the onboard IMU during a guidance experiment. Top-right and bottom) evolution of the line components (θ, ρ) of three tracked lines along the experiment. The flapping-wing robot was approximately launched at time 0.7 s.



(a)

(b)

Figure 3.10: Examples of lines tracked by the proposed algorithm in different scenarios: (a) lines tracked in an indoor experiment, and (b) lines tracked during an outdoor validation test. The grayscale frames were used only for visualization purposes.

challenging, the reported average goal position error was 0.40 m, which is consistent with guidance methods based on external perception systems [39].

	Gliding	*Flapping descending*	*Flapping straight*	*FlappingOutdoors*
Error (m)	0.465	0.221	0.409	–
NRMSEx	0.2482	0.1745	0.3810	0.0320
RMSEx	0.3356	0.4192	0.4291	0.0227
NRMSEy	0.1384	0.0300	0.1209	0.0331
RMSEy	1.1665	0.2648	0.8214	0.3225

Table 3.3: Average performance of the proposed method in the different experiments.

Finally, Figure 3.11-top depicts the camera velocity error $\nu(t)$ in the same guidance experiment of Figure 3.9. The purple area represents the launching stage while the yellow area corresponds to the guidance maneuver. The camera velocity errors tended to zero evidencing that the robot approached the goal while following the line tracks. The computation of $\nu(t)$ depends on the feature error and the estimated distance to the pattern. Besides, Figure 3.11-top shows the smoothed input errors $\mathbf{v}$. During the initial flight stage, $\mathbf{v_y}$ was large compared to the magnitude of $\mathbf{v_x}$. Hence, the initial longitudinal deviation was larger and required a more aggressive control action to converge. Additionally, Figure 3.11-bottom shows the position of the flapping-wing robot obtained by the motion capture system during the same guidance experiment. The robot position is represented with continuous lines while the goal position with dashed lines. The robot performed a smooth maneuver, and it reached the goal position when the longitudinal position (i.e., Y) reached the reference. The final position error was 0.207 m.

3.8 Conclusions

This Chapter presents a guidance method for aerial robots based on event vision. It leverages the asynchronous nature of event data by fusing *event-by-event* and *event-image* processing. Events are used for line detection and tracking providing line references at high rates (i.e.,>300 Hz). The guidance system includes a time-to-contact trajectory approach based on *Tau* theory, which allows gap closing trajectories within

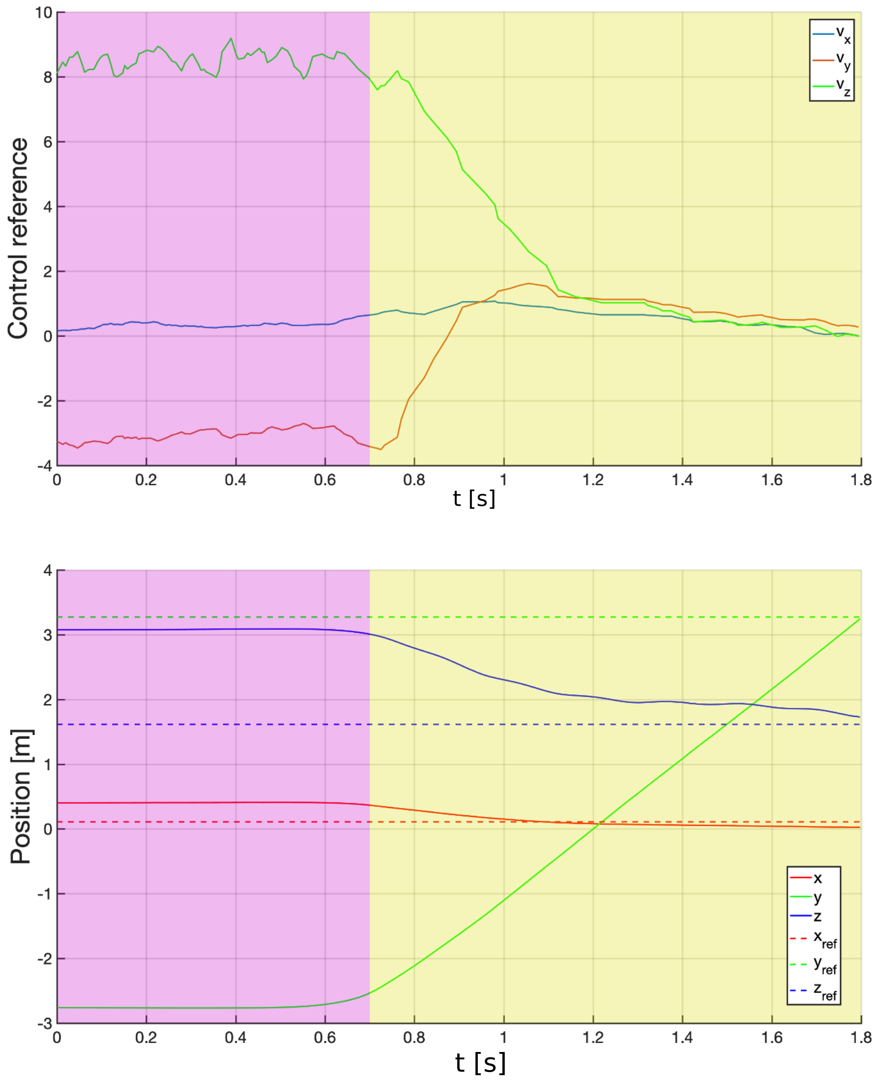


Figure 3.11: Evolution of the velocity error $\nu(t)$ (top), and ornithopter position (bottom) during the guidance maneuver described in Figure 3.9. The purple and yellow colors correspond to the launching and flapping stages of the experiment. The robot pose was obtained from the motion capture system.

a specific time T_g. Besides, an EBVS controller guides the aerial robot toward its goal configuration. The proposed approach has been validated in two different aerial robots. Initially, the method was validated in a quadrotor and later in one of the flapping-wing robots of the ERC GRIFFIN Project. The experimental results show the capabilities of the proposed scheme for the problem of visual guidance. Furthermore, the validation results of the line tracking approach demonstrate its low tracking error and fast response.

The proposed solution demonstrated its suitability for visual guidance on aerial platforms. However, the current implementation uses simple low-level controllers to guide the robot toward the goal. Future work should include better control approaches to perform fast maneuvers with both experimental platforms. Thus, exploiting the fast response of the event-based perception algorithms together with specialized controllers for fast and agile maneuvering. Besides novel control approaches and models are necessary for flapping-wing robots such as the *E-Flap* ornithopter. Although the experimental results using ornithopters are promising, additional work is needed to design and implement more suitable controllers for the problem of visual guidance with large-scale flapping-wing robots.

Chapter 4

Event-based Dynamic Sense-and-Avoid for Ornithopters

4.1 Introduction

Although the interest in flapping-wing robots has led to the development of several ornithopters [44] [34] [56], few perception systems have been developed to perform autonomous navigation with these platforms. Obstacle avoidance is a relevant task for robot navigation as the platforms must be able to evade static and dynamic obstacles to avoid possible collisions. However, the development of *sense-and-avoid* methods for ornithopter robots is mainly limited by their restricted payload to mount perception sensors and processing boards to fastly detect obstacles and compute evasive trajectories. Besides, low latency perception methods are required in large-scale ornithopters as these platforms typically report flight speeds $\geq$ [3.5 - 16]$\mathrm{m\,s^{-1}}$ [34] [44]. Unlike traditional obstacle avoidance schemes that assume static obstacles in the scene, this Chapter addresses the problem of avoiding unexpected dynamic obstacles with ornithopter robots. The perception scheme is based on event cameras, which trigger pixel information due to the brightness variations produced by the camera motion and moving objects in the scene. This feature is particularly useful to detect dynamic obstacles in the camera's Field of View (FoV).

This Chapter proposes a dynamic *sense-and-avoid* scheme for ornithopter robots using event-based vision. It only processes perception information from an event camera to quickly detect dynamic obstacles, and modify the trajectory of the robot to avoid them. The scheme exploits the asynchronous nature of event cameras for obstacle avoidance by performing *event-by-event* processing. Moving obstacles are detected using spatial-temporal information from events triggered by objects that move with a higher velocity than the camera. The moving obstacle direction in the image plane is estimated through optical flow, and a reactive avoidance maneuver strategy rapidly evaluates and prevents collision risk situations. The scheme is validated in the *E-Flap* ornithopter [34] in indoor and outdoor experiments. **To the best of our knowledge, this is the first event-based *sense-and-avoid* scheme designed and validated in an ornithopter**.

This Chapter describes **Contribution 2**, and its experimental validation and software implementation are part of **Contribution 6** and **Contribution 7** of this Ph.D. Thesis. Besides, the research conducted in this Chapter led to publication [59].

This Chapter is organized as follows. Section 4.2 summarizes the main works related to the topics addressed in the Chapter. A general description of the dynamic *sense-and-avoid* method for flapping-wing robots is presented in Section 4.3. Section 4.4 describes the event-based dynamic object motion estimation pipeline. The proposed reactive collision evaluation strategy is described in Section 4.5. Section 4.6 presents the adopted tail controller used to perform evasive maneuvers. The experimental results are presented in Section 4.7. Finally, Section 4.8 describes the chapter conclusions.

4.2 Related work

Reactive obstacle avoidance methods aim at performing evasive actions as fast as possible without relying on a globally consistent map of the scene. These *sense-and-avoid* approaches are categorized into *map-based* and *map-less* obstacle avoidance [139]. *Map-based* approaches rely in local maps to compute obstacle-free trajectories [140]. The local maps are computationally cheap to build compared to those used in traditional methods [141]. Conversely, *map-less* approaches detect nearby obstacles and directly perform the avoidance action [142]. These methods provide faster responses by reducing the number of perception algorithms needed for obstacle avoidance, which make them suitable for platforms with limited processing capacity. For instance, the work in [143] describes a low latency *map-less* obstacle avoidance method to avoid flying obstacles using saliency-based reinforcement learning. Besides, authors in [85] study the influence of the perception latency in a high-speed *sense-and-avoid* method validated on a quadrotor.

During the last decades, the scientific and technological advances in ornithopter robots have led to the development of a few systems that integrate onboard perception sensors to provide information for navigation, landing, and perching. In general, vision-based sensors have been preferred to extract sensing information of the scene. The work in [144] estimates optical flow onboard a Micro Aerial Flapping-wing robot. It subsamples the input images and uses a motion detection algorithm to compute the optical flow. Gliding experiments show remarkable system capabilities while flapping tests depict the method limitations in flapping conditions. Object appearance variations and optical flow are used in [145] to perform obstacle avoidance with a monocular camera. The obstacle detection and avoidance methods run in a base station due to the payload limitations of the platform. The work in [41] presents a stereo-vision obstacle avoidance strategy for small-scale ornithopters. The method proposes the droplet strategy for obstacle avoidance, which consists of defining a drop-shape obstacle-free area in front of the robot that guarantees a safe path to exert the avoidance maneuver. Moreover, **Contribution 5** of this Ph.D. Thesis provides a dataset recorded onboard a bird-scale flapping-wing robot. The dataset includes measurements from a frame-based camera, an event camera, two Inertial Measurement Units (IMUs), and robot pose ground truth measurements from either a TotalStation or a motion capture system. The work in [146] presents a mechanical system for image stabilization onboard an ornithopter robot. It stabilizes the pitch and roll oscillations during flapping using a gimbal with a payload of 100 g. Despite the system provides a solution to deal with the mechanical vibrations arising during flight, the additional hardware weight often limits the autonomy and maneuverability of the robot.

The advantages of event cameras compared to frame-based sensors have increased the research interest of the robotic community on these sensors [16]. Event cameras are particularly useful to detect moving objects as they provide pixel information of dynamic objects without requiring further processing. Some previous works have presented event-based pipelines for obstacle avoidance. An initial approach is presented in [147] to estimate the time-to-contact between the robot and static obstacles in the scene. The method computes flow fields from events and estimates their Focus Of Expansion (FOE). Time-to-contact is estimated using the optical flow occurring at the FOE location. Experimental evaluation is performed onboard a ground robot moving with a maximum velocity of $1\,\mathrm{m\,s^{-1}}$ in a static indoor scenario. However, this work assumes static scenes without moving obstacles where almost all triggered events are generated by the camera motion. Conversely, the method in [142] presents a dynamic obstacle avoidance method using stereo event cameras on a quadrotor. The object trajectory is estimated and propagated in time to predict possible collisions. Authors in [120] describe a dodging system for quadrotors. The proposed pipeline consists of three Deep Learning (DL) modules for *event image* deblurring, odometry estimation, and moving object segmentation. The work in [103] presents a dynamic obstacle avoidance approach for quadrotors using a monocular and a stereo pair of event cameras. The method performs rotational motion compensation to distinguish events triggered by moving objects from those generated from the background. A potential field approach computes the evasive maneuver which is exerted by the quadrotor. The work in [104] proposes a similar approach including a depth camera to compensate for the camera's translational motion. Motion compensated events are used to detect moving obstacles using iterative Gaussian fitting. A 3D trajectory estimator keeps track of the obstacle during flight. However, the work misses an avoidance approach to evade the detected obstacles. Finally, the authors in [148] propose a DL framework for static and dynamic obstacle avoidance. The framework includes a network to compute optical flow from frames and events, and a second network that uses optical flow and depth measurements to compute time-to-impact maps. The proposed network integrates measurements from a Laser imaging Detection and Ranging sensor (LiDARs) to detect static obstacles and event cameras for dynamic obstacles. The experimental validation is performed only in a simulated scenario with a robot agent performing avoidance maneuvers similar to [120]. The majority of previous works propose event-based obstacle avoidance pipelines for mobile robots, however, none of them have approached the problem of detecting and evading dynamic obstacles on a flapping-wing robot.

Table 4.1 summarizes a brief comparison between the proposed approach and the methods in [142, 120, 103] for dynamic obstacle avoidance with quadrotors. First, methods [142, 120, 103] require platforms with enough payload to mount the necessary sensors and computers to run their obstacle avoidance solutions. The methods in [120] and [103] require a powerful embedded board, a specialized flight computer, and an autopilot board to run their proposed solutions. Further, the three methods integrate two event cameras which increases the onboard hardware weight, and the computational resources to process all the event information on the onboard computer. Despite, the work in [103] includes a monocular solution for obstacle avoidance, it requires an additional board to run vision-based state estimation. Conversely, the proposed scheme integrates only an event camera and runs on a lightweight board while providing a fast perception response of 250 Hz. Second, none of these methods is designed to run on a platform moving at medium-high velocities. All previous methods are experimentally validated in quadrotors while hovering (or moving with speeds up to $1.5\,\mathrm{m\,s^{-1}}$ in [103]), where the majority of events are caused by the motion of the obstacle which simplifies its detection. Conversely, the proposed scheme is designed to run in the *E-Flap* ornithopter which requires a minimum speed of $3.5\,\mathrm{m\,s^{-1}}$ to flight [34], and triggers additional events due to its flapping motion [42]. Finally, the works in [120] and [103] perform event motion compensation to segment moving objects. Both motion compensation approaches use *event images* obtained by accumulating the incoming events. This creates delays between event generation and processing, even when the *event image* processing times are lower than the event accumulation times (e.g., [103]). Conversely, the proposed scheme performs *event-by-event* processing to exploit the asynchronous nature of event cameras, and enable shorter obstacle detection times which is relevant for ornithopters flying at medium-high velocities.

	Flight speeds	**Board Weight**	**No. of cameras**	**CPU/GPU cores**	**Processing type**	**Accumulation Type**	**Processing Time**
[142]	$\sim 0\,\mathrm{m\,s^{-1}}$	∼49 g	2	4/0	*event-by-event*	Async	–.
[120]	$\sim 0\,\mathrm{m\,s^{-1}}$	∼150 g	2	6/1	*event-images*	30 ms	12 ms/*e-images*
[103]	$[0,1.5]\,\mathrm{m\,s^{-1}}$	∼150 g	1-2	6/1	*event-images*	10 ms	3.56 ms/*e-images*
Ours	$[2,4]\,\mathrm{m\,s^{-1}}$	28.5 g	1	6/0	*event-by-event*	Async.	4 ms/package

Table 4.1: Comparison of our approach with dynamic obstacle avoidance methods using only event-based perception for quadrotors. *e-images* is the abbreviation of *event images*.

4.3 General scheme

The strict specifications of flapping-wing robots entail several considerations for the design and implementation of *sense-and-avoid* schemes for these types of platforms. First, ornithopters report a low payload capacity which hinders the installation of multiple sensors, powerful processing boards, and large batteries. Second, simple perception algorithms are required to run on lightweight boards with limited computing resources without generating significant delays to perform *sense-and-avoid*. Third, flapping-wing robots describe complex dynamics and kinematics. They are non-holonomic robots that use only a few actuators to control their 6 Degree of Freedom (DoF) pose. For instance, ornithopters flap their wings to produce lift and thrust to gain altitude and produce forward motion. The aforementioned considerations define a set of requirements to design and implement fast perception methods for obstacle detection and avoidance in ornithopter robots.

The proposed scheme integrates event cameras to detect dynamic objects along with an obstacle risk evaluation approach to avoid them. Figure 4.1 depicts the general diagram of the scheme. The *Dynamic Obstacle Motion Estimation* module detects dynamic obstacles and estimates their direction of motion in the image plane. It analyzes the spatial-temporal information of the event stream to filter events triggered by the background of the scene (i.e., events produced by the camera motion). Despite some frame-based methods for motion segmentation have been proposed in the literature [149], event cameras provide asynchronous per pixel information which could be directly processed to enable fast perception responses for obstacle *sense-and-avoid*.

Moreover, event cameras offer high-dynamic range and robustness to motion blur, which are particularly beneficial for the perception on board ornithopters due to the strong vibrations and changes of illumination caused by the flapping strokes. Finally, the low power consumption of event cameras motivates their use in power-constrained platforms such as flapping-wing robots.

The *Reactive collision evaluation strategy* module determines possible collisions with the dynamic obstacle based on the robot and the obstacle geometries. In case of detecting a possible collision, the *Tail Control* module changes the flight course by actuating on the vertical and horizontal tail deflections. This last module satisfies both the robustness and simplicity requirements for the avoidance task.

Finally, the event processing module (i.e.,*Dynamic Obstacle Motion Estimation*) uses ASAP [150] to avoid processing overflow. ASAP synchronizes event packaging such that events are processed as soon as possible while closing the control loop at 250 Hz on the onboard resource-constrained hardware.

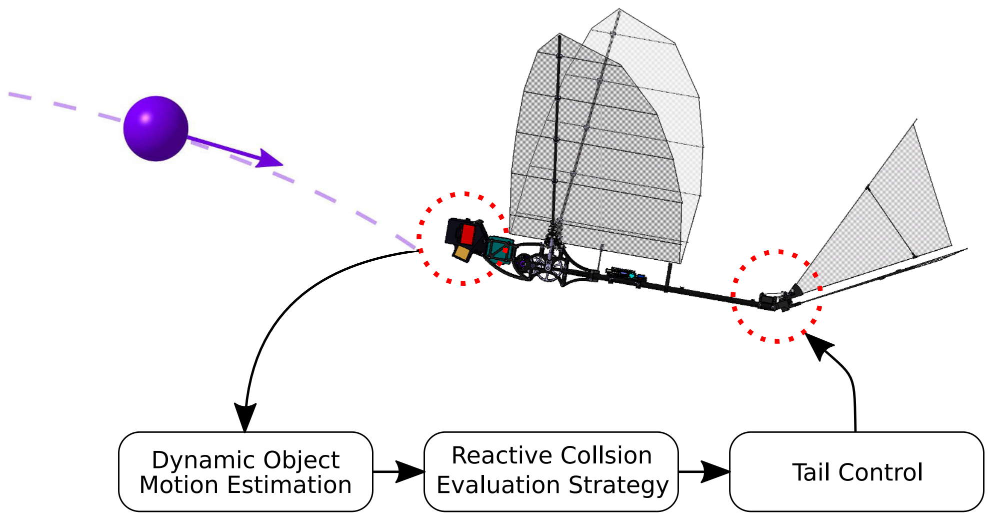

Dynamic Object Motion Estimation
Reactive Collsion Evaluation Strategy
Tail Control

Figure 4.1: General diagram of the *sense-and-avoid* scheme for ornithopter flight.

4.4 Event-based dynamic obstacle motion estimation

This module analyzes the event stream to detect dynamic objects and estimate their direction of motion in the image plane. The proposed method performs *event-by-event* processing to leverage the asynchronous property of event cameras. As it was mentioned in previous Chapters, each event is defined by: $e = (\mathbf{x}, ts, p)$, where $\mathbf{x}$ corresponds to the pixel coordinates (x, y), ts represents the timestamp of the event, and the polarity $p \in \{+1, -1\}$. This module is divided into four submodules each performing *event-by-event* processing. Figure 4.2 shows the block diagram of the module. The event stream is processed by the *Time Filter* submodule that discards events triggered by the scene background using as reference the timestamp of current and previous events produced at the same location. The *Corner Detector* submodule extracts corner features from events triggered by moving objects. The *Optical Flow* submodule estimates the direction of motion of events triggered by moving objects. The *Clustering* submodule clusters flow events and estimates the mean optical flow of the object. Algorithm 4 briefly describes the main step of the proposed pipeline.

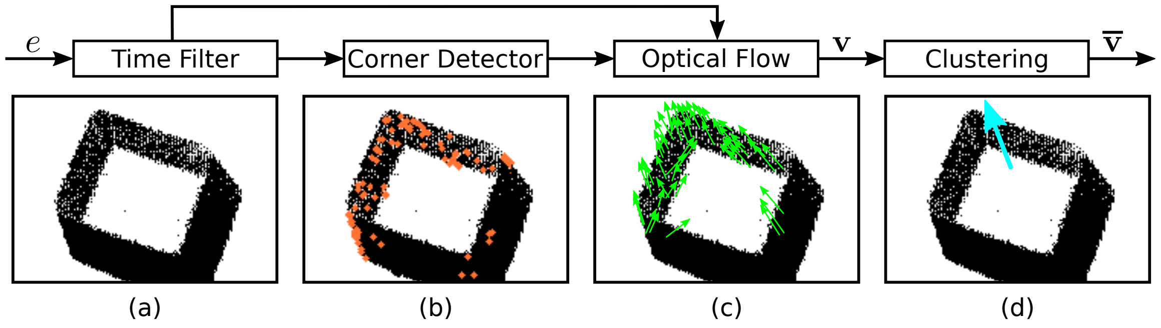


Figure 4.2: General block diagram of the *Dynamic Object Motion Estimation* pipeline: (a) the *Time Filter* filters events generated by the background of the scene; (b) the *Corner Detector* extracts relevant features of the object; (c) the *Optical Flow* submodule computes the direction of motion of events in the image plane; and (d) the *Clustering* submodule estimates the mean flow of the dynamic object. The *event images* of each submodule are examples of the results obtained by processing events triggered within a time window of 10 ms.

Algorithm 4: Event-based moving obstacle detection

```
Input: $e_k(\mathbf{x}_k, ts_k, p)$
Output: $(\overline{v_x}, \overline{v_y})$
if $ts_k - \mathbf{S}_t(\mathbf{x}_k) < \tau_t$ then                    ▷ Time Filter evaluation
    updateCurrentSlice($e_k$)
    if isCorner($e_k$) then                                   ▷ Corner Detector
        $(v_x, v_y) \leftarrow$ computeOpticalFlow()
        $id \leftarrow$ updateClusters($e_k, v_x, v_y$)
        $(\overline{v_x}, \overline{v_y}) \leftarrow$ getClusterFlow($id$)
    end
    updateClusters($ts_k$)                                    ▷ Compute obstacle flow
end
$S(e_k) = ts_k$                                             ▷ Update time reference at $\mathbf{x}_k$
$d, \tau_c \leftarrow$ sliceRotation($\mathbf{x}_k$)
```

The problem of distinguishing events triggered by moving objects has been previously addressed using optimization algorithms [84], motion compensation techniques [103] [104], and Deep Learning methods [148]. However, these methods either require powerful computers to run (e.g., including Graphics Processing Units (GPUs)) or accumulate events to correctly detect dynamic obstacles. Conversely, the proposed method performs *event-by-event* processing to provide low latency responses suitable for the *sense-and-avoid* task onboard an ornithopter such as *E-Flap.* The *Time Filter* submodule corresponds to the algorithm proposed in Section 2.3.4 of Chapter 2. These submodule filters events triggered by the background of the scene while keeping those generated by dynamic obstacles. The filter keeps the timestamp information of past events in the map $\mathbf{S}_t \in \mathbb{R}^2$. At each iteration, the filter compares the timestamp of the current event with coordinates $\mathbf{x}$ with the timestamp of the last event that occurred at $\mathbf{S}_t(\mathbf{x})$. Afterward, if Δts is lower than the time threshold τ_t, the event is considered as triggered by a dynamic object. The τ_t threshold is defined by Eq. 2.4 (see Section 2.3.4). In the current *Time Filter* implementation, τ_t is set by using the body frame velocity, which is either obtained from the onboard inertial navigation system VectorNav VN-200 or a motion capture system. Figure 4.3-right depicts a set of events considered as being triggered by a moving object (i.e., pixels in magenta).

The grayscale frame was captured by the Active Pixel Sensor (APS) sensor of the DAVIS346 and it is only used to provide a clearer visualization.

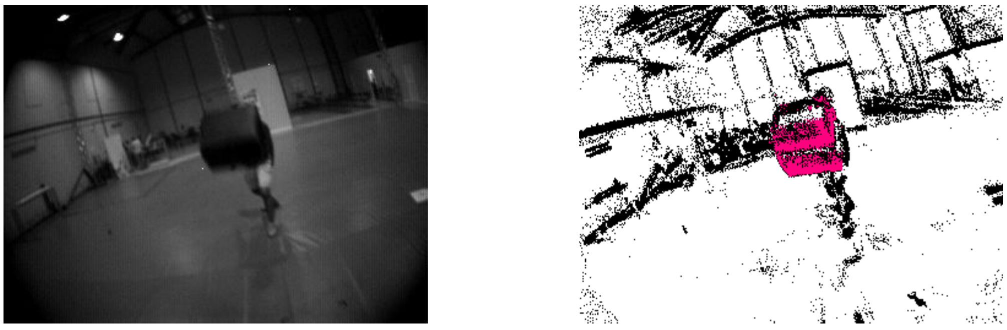

(a) (b)

Figure 4.3: An example of the events considered as generated by a dynamic object using the *Time Filter* submodule. (a) grayscale frame from the APS sensor of the DAVIS346; (b) *event image* rendered by accumulating events in time windows of 10 ms (black pixels). The set of events considered to belong to the moving object is shown in magenta.

Afterward, the *Optical Flow* submodule computes the direction of motion of events. This submodule aims at estimating the relative motion between moving objects and the camera by computing only the flow of events generated by dynamic objects in the scene. Figure 4.2-c shows an example of the flow computed by the *Optical Flow* submodule. The proposed approach adopts the event-based optical flow method ABMOF [81]. It performs block matching operations between event slices, which are 2D maps of events accumulated within the time window d. Event slices are similar to *event images* by being a 2D representation of the event stream. Unlike other approaches using *event images*, ABMOF updates event slices with each incoming event and performs optical flow estimations following an *event-by-event* processing approach. Besides, the accumulation time window d varies depending on the event generation through time. ABMOF has been adapted for the proposed pipeline to reduce computational processing for onboard computation on *E-Flap*. First, the method has been migrated to C++ and several implementation details of the original approach have been modified to reduce computational cost. For instance, event slices are updated only with events triggered by dynamic objects. This modification reduces the computation load by analyzing only <32% of the event stream in the experiments of Section 4.7. Moreover, optical flow is computed only from event corner features. This modification reduces the computational cost by >25 % by computing only flow from relevant features of the moving object while providing stable optical flow estimations as described in Section 4.7. Corner features are extracted using the *eFast event *Corner Detector* in [74], which reports fast responses with a low False Positive rate [71].

The *Clustering* method is the last submodule of the pipeline. It clusters event flow estimations and computes an approximation of the moving object's optical flow. This submodule adopts the clustering algorithm presented in Section 2.3.2 of Chapter 2. However, it has been modified to cluster flow from events. It inputs the tuple $f_e = (\mathbf{x}, ts, \mathbf{v})$, where $\mathbf{v} = (v_x, v_y)$ is the flow estimation of the event triggered at location $\mathbf{x}$ with timestamp ts. The clustering algorithm associates input events with previous clusters by evaluating the proximity between the new input event to a random event contained in each cluster. Each cluster is defined by its centroid $\overline{\mathbf{x}}$, its average optical flow $\overline{\mathbf{v}} = (\overline{v_x}, \overline{v_y})$, and $\mathbf{L}_c$ the list of previous tuples f_e assigned to the cluster. After event-cluster association, each new optical flow sample updates $\overline{\mathbf{v}}$ as follows:

$$\overline{\mathbf{v}} := \frac{\varrho}{\varrho + 1}\overline{\mathbf{v}} + \frac{1}{\varrho + 1}\mathbf{v}, \tag{4.1}$$

where ϱ is the number of tuples assigned to the cluster (i.e., the cardinality of $\mathbf{L}_c$). Moreover, the cluster updates as well when the influence of old samples is removed from $\overline{\mathbf{v}}$:

$$\overline{\mathbf{v}} := \frac{\varrho}{\varrho - 1}\overline{\mathbf{v}} - \frac{1}{\varrho - 1}\mathbf{v}_{\dagger}, \tag{4.2}$$

where $\mathbf{v}_{\dagger}$ represents the flow samples with timestamp lower than τ_c. The parameter τ_c defines the maximum lifetime of flow samples in each cluster. Moreover, only clusters with event samples covering an area in the image plane greater than the threshold η_f (5×5 px in the experiments of Section 4.7) are considered as dynamic objects. This condition is necessary to avoid triggering evasive maneuvers due to noisy samples in the scenario.

Finally, the proposed pipeline adopts ASAP to prevent processing overflow while preserving the responsiveness of the scheme. From a low-level software perspective, ASAP receives the event stream and dynamically adapts the packages of events before sending them to the dynamic object motion estimation pipeline.

4.5 Reactive collision evaluation strategy

The proposed *sense-and-avoid* scheme includes a reactive strategy to avoid impacts with moving obstacles. It estimates possible collisions with detected obstacles based on the geometry of the robot. The body of the ornithopter is approximated by a $2H \times 2W$ volume and the obstacle is represented by a sphere of radius R' (see Figure 4.4). R' is an enlarged volume enclosing the volume of the obstacle and a *Security Distance* b_u (i.e., $R' = R + b_u$) that considers the sensing uncertainties. Following the previous geometries, the minimum angles θ and ψ to evade an obstacle without modifying the flight trajectory of the robot are given by:

$$\psi^*(t) = \arctan \frac{W + 2R'}{Z(t) - 2R'}, \tag{4.3}$$

$$\theta^*(t) = \arctan \frac{H + 2R'}{Z(t) - 2R'}, \tag{4.4}$$

where $Z(t)$ is an approximation of the obstacle depth relative to the camera. For any set of angles (ψ, θ) such that $|\psi(t_c)| \leq |\psi^*(t_c)|$ and $|\theta(t_c)| \leq |\theta^*(t_c)|$, a collision between the ornithopter and the obstacle volumes occurs at $t \geq t_c$ if the relative velocity between them is not modified. In this case, an avoidance maneuver is activated by guiding the flapping-wing robot in an opposite direction from the obstacle trajectory. The reactive collision evaluation described by Eqs. 4.3 and 4.4 depends of the depth $Z(t)$, and the obstacle geometry R. The proposed module considers three different approaches to retrieve the depth information:

1. Obtaining $Z(t)$ from additional sensors such as a depth camera, a pair of stereo cameras, a laser, or using an external motion capture system.

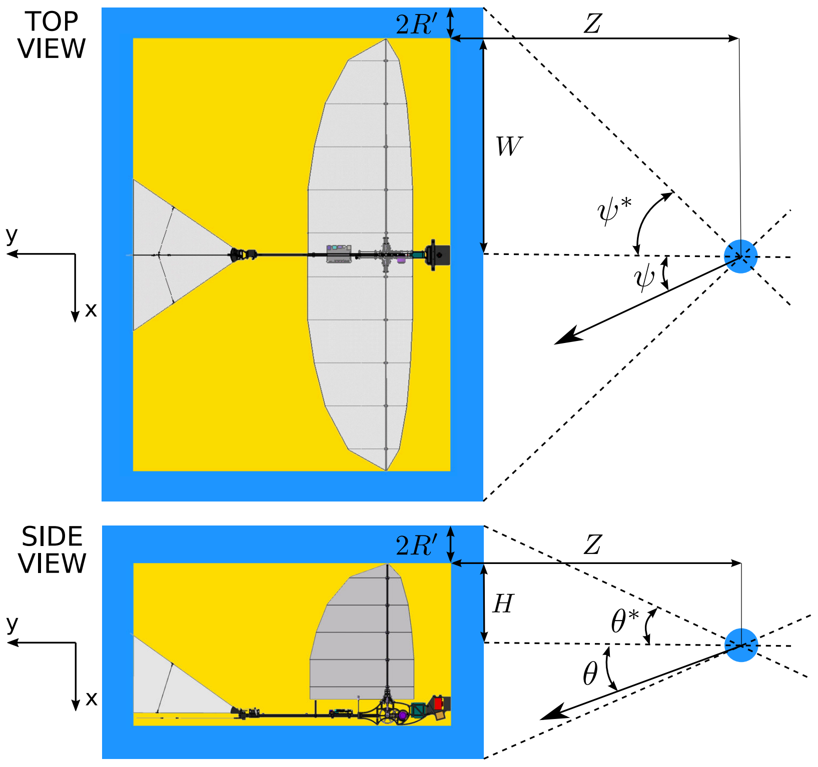


Figure 4.4: The ornithopter body is approximated by a $2H \times 2W$ volume and the obstacle is represented by a sphere of radius R'. The collision evaluation constraints are defined by ψ^* and θ^*, which are used to determine possible collisions with the ornithopter.

2. Computing $Z(t)$ from a vision-based method [151] [53] without hindering the on board obstacle detection computation.

3. Estimating $Z(t)$ using the geometric information of the obstacle R by $Z(t) = \lambda R/l(t)$, where λ is the camera focal length, and $l(t)$ is the largest side of the bounding box enclosing the clustered events. The bounding box of the cluster is obtained from the event distribution within it.

After retrieving $Z(t)$, the collision evaluation is directly computed from Eqs. 4.3 and 4.4. Moreover, If neither R nor $Z(t)$ are known, the collision cannot be predicted and any detected obstacle triggers an evasive maneuver. From the previous cases only the third is experimentally considered in the experimental validation of Section 4.7;

having prior information on the object geometry. The first case requires additional perception hardware to extract the obstacle depth information which increases the hardware weight and the onboard power consumption. Moreover, the current version of the *sense-and-avoid* scheme does not include a method to perform event-based monocular depth estimation. Most of these algorithms are computationally expensive to run together with the perception methods of the proposed scheme. Instead, it performs reactive avoidance maneuvers using dynamic obstacle information, which reduces processing requirements, permitting fast execution.

4.6 Tail control

Performing reactive obstacle avoidance with large-scale ornithopters requires a fast and robust response to exert rapid aggressive movements. Thus, the control method must be computationally simple to perform the evasive maneuver before colliding with the obstacle. Furthermore, the controller must be robust to reduce the possible uncertainties that may arise from the onboard motion estimation method.

The proposed tail controller sets an evasive maneuver in an opposite direction to the motion vector of the detected dynamic obstacle (i.e., $\overline{\mathbf{v}}$). The controller computes the lateral δ_r and longitudinal δ_e tail deflections to perform the evasive maneuver using as reference the obstacle optical flow $\overline{\mathbf{v}}$. The adopted controller is described as follows:

$$\mathbf{u_{react}} = -(\boldsymbol{\kappa_0} + \boldsymbol{\kappa_1}\|\overline{\mathbf{v}}\|) \circ \overline{\mathbf{v}}, \tag{4.5}$$

where $\mathbf{u_{react}} = [\delta_e^{react}, \delta_r^{react}]^T$ describes the control action, $\boldsymbol{\kappa}_0$ and $\boldsymbol{\kappa}_1$ are the control gains, and $\circ$ denotes the Hadamard product. The control gains $\boldsymbol{\kappa}_0$ and $\boldsymbol{\kappa}_1$ are experimentally tuned to achieve fast control response when an obstacle is detected. Hence, the proposed control approach assumes that the best avoidance maneuver is to fly in a direction opposite to the obstacle's motion. It describes a simple implementation with low computational requirements which is suitable to run onboard ornithopters. Besides, the controller has been designed to consider the tail deflection boundaries presented in [35] to avoid stall while performing the avoidance maneuver.

4.7 Experiments

The proposed *sense-and-avoid* scheme was validated using the *E-Flap* ornithopter. The main technical specifications of the robot along with a description of the onboard hardware have been presented in Section 3.7. Differently from the experiments of Chapter 3, the event camera integrated a low-distortion lens to avoid undistorting input events reducing the number of perception modules running on the robot. The event-based obstacle detector method, the evasive maneuver strategy, and the flapping-wing controller avoidance were implemented in C++ and embedded in a *ROS*-package. For all experiments, ASAP was configured to deliver event packages at 250 Hz to cope with the low computational restrictions. Besides, the motion direction of the obstacle was updated each 4 ms to reduce the computational load of communicating all sensing, perception, and control nodes running in parallel on the robot. Finally, the parameters of the ornithopter volume were given by W=0.75 m and H=0.273 m.

The proposed approach was validated in both indoor and outdoor scenarios. The indoor scenario corresponded to the GRVC Testbed equipped with a motion capture system to estimate the ornithopter pose. A soccer field of 48 × 54 m describes the outdoor scenario. In the presented experiments, the parameters of Eq. 2.5 were set to ω_L=1.3 rad s^{-1}, ω_H=3 rad s^{-1}, v_L=3 m s^{-1}, and v_H=6 m s^{-1} after studying the typical flight kinematics conditions of the ornithopter. Besides, the parameter α was set to 0.8 as the robot mainly described a forward motion in the *sense-and-avoid* experiments. Moreover, as it was mentioned in Chapter 2, the parameter τ_t defines the threshold to distinguish events triggered by a dynamic object from those generated from the background. A small value of τ_t leads to selective filtering while reducing the distance at which the obstacles are detected. Conversely, a large value of τ_t entails longer distance detection while allowing events triggered by static objects. The τ_t selection represents a trade-off between the maximum distance to detect an obstacle, and filtering events triggered by static objects. In the experiments described in this

Chapter, τ_H=25 ms and τ_L=15 ms were empirically selected to provide a balance between performing detection at distances of 6 m while filtering at least 82% of events produced by the scene background.

The validation of the proposed *sense-and-avoid* scheme was divided into two parts. First, an evaluation of the dynamic obstacle detection and motion estimation methods is presented in Section 4.7.1. Second, Section 4.7.2 describes the experimental validation of the *sense-and-avoid* scheme in experiments performed in indoor and outdoor scenarios.

4.7.1 Obstacle detection and motion estimation evaluation

This section evaluates the obstacle detection and motion estimation capabilities of the proposed *sense-and-avoid* scheme. The first set of experiments focused on evaluating the obstacle detection accuracy. The experimental setup consisted of launching obstacles into the FoV of the event camera while the flapping-wing robot performed forward flight. The testbed scenario was selected for the experiments as it includes a motion capture system to retrieve the pose information of the robot and the obstacle. Three types of objects were selected for the evaluation. Each object described a different shape and size. Figure 4.5 shows the selected objects, a Small Box of size 220 $\times$ 200 $\times$ 150 mm, a Stuffed Toy of size 400 $\times$ 450 $\times$ 400 mm, and a Fitball with a diameter of 750 mm. The obstacle detection was performed from distances (i.e., between the camera and the object) ranging from 0.5 m to 6 m. The minimum distance was set to 0.5 m as at closer distances the body of the obstacles filled a large zone of the image plane which produced invalid detections. Besides, at larger distances (i.e., >6 m) the representation of the object in the image plane was quite small to perform a reliable detection. A total of 45 experiments with each object were performed for this evaluation.

The detection performance evaluation consisted of comparing the obstacle detection results with the ground truth extracted from the grayscale frames by the APS sensor of DAVIS346 camera. A frame representation of the detected obstacle was provided by rendering its centroid into an *event image* together with the events accumulated during

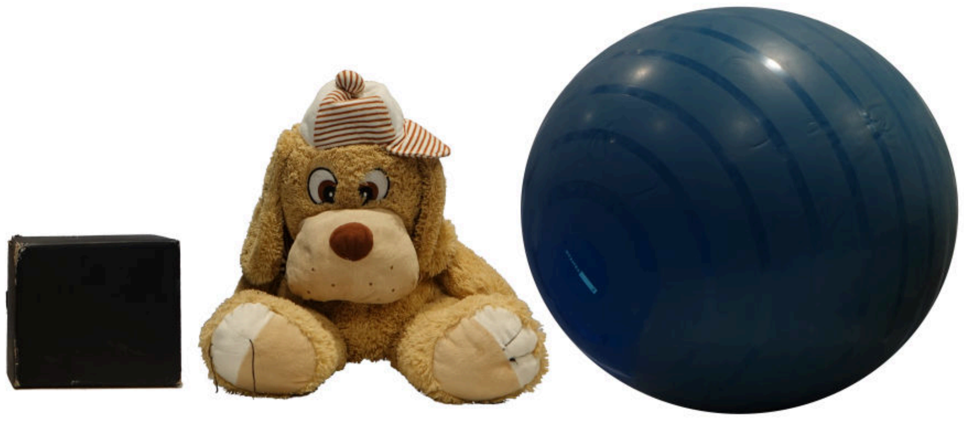


Figure 4.5: Objects of different shapes and sizes selected for the *sense-and-avoid* experiments: left) Small Box; center) Stuffed Toy; and right) FitBall.

the last 25 ms. The *event image* generation was synchronized with the frames obtained from the APS sensor such that both almost correspond to the same time window. The centroid of the obstacle in the grayscale frames was manually annotated for all experiments. The pixel distance between both centroids was used as an evaluation criterion. Each comparison produced one of the following results: (i) True Positives (TP), which occurred when the distance between both centroids was lower than 10 px, (ii) False Negatives (FN), which arose either when the detector did not detect the obstacle while it appeared in the grayscale frame, or when the distance between both centroids was larger than 10 px, (iii) False Positives (FP), which occurred when the detector reported a detection while there was no obstacle in the camera FoV, and (iv) True negatives (TN) corresponding to the cases where there was not an obstacle in the camera FoV and the detector did not report any detection. Figure 4.6 depicts the histogram of the distance between the dynamic obstacle and the ground of truth in the performed experiments. The majority of cases when the distance was greater than 15 px corresponded to False Positives. Setting the distance threshold to 10 px reported a suitable trade-off between validating the majority of samples as True Positives while avoiding False Positive samples.

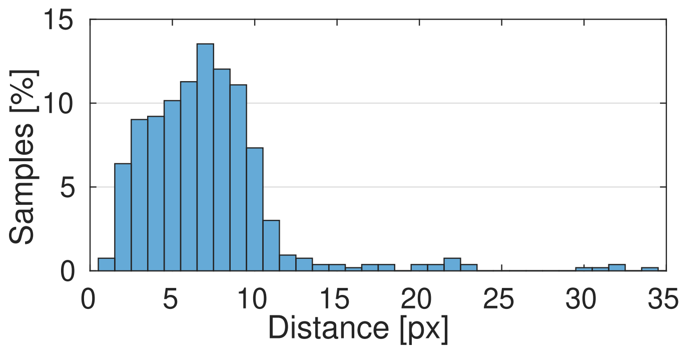


Figure 4.6: Distance in the image plane between the centroid of the detected obstacle and the ground truth for all performed experiments.

The aforementioned results were analyzed to compute the *Accuracy*, *Precision*, *False Positive Rate (FPR)*, and *True Positive Rate (TPR)* metrics to evaluate the obstacle detection performance. Table 4.2 summarizes the experimental results obtained with each obstacle. In general, the method reported an *Accuracy* of 91.7 %. The results evidenced that the distance affected the detection performance especially when small objects were launched at distances $> 4\,\mathrm{m}$. In these cases, the obstacle produced few events hindering its detection as the majority of the events in $\mathbf{S}_t(\mathbf{x})$ did not satisfy the τ_t condition. Moreover, the experiments performed at distances between $2\,\mathrm{m}$ to $4\,\mathrm{m}$ reported *Accuracy* results above 94.7 %. In this range, the size of the objects in the image was large enough to successfully detect the obstacles. Furthermore, the experimental results describe a low number of *False Positives* given the average *FPR* of 4.2 %, and an average *Precision* of 95.1 %. Finally, the average 88.5 % *TPR* result depicts that few detections were missed in the validation experiments.

Furthermore, the obstacle detection approach was evaluated in multi-obstacle experiments where three different objects were launched toward the camera FoV. The method performance decreased by reporting an average accuracy of 74.2%. This is due to the increment of False Positive samples when the obstacles overlapped in the image plane, which hampered the detection of each object. Although this experiment reported a significant degradation in the method performance, the results are still satisfactory due to the complexity of detecting several obstacles with a single event camera.

	1.5 – 2.0 m				**2.0 – 4.0 m**				**4.0 – 6.0 m**			
	Acc	*Pre*	*TPR*	*FPR*	*Acc*	*Pre*	*TPR*	*FPR*	*Acc*	*Pre*	*TPR*	*FPR*
Small Box	0.91	0.95	0.86	0.04	0.97	0.98	0.97	0.03	0.80	0.86	0.75	0.09
Stuffed Toy	0.97	0.95	0.97	0.04	0.93	0.98	0.86	0.02	0.92	0.97	0.85	0.02
Fitball	0.91	0.98	0.82	0.01	0.94	0.94	0.93	0.05	0.92	0.94	0.91	0.08

Table 4.2: Experimental results of the obstacle detection evaluation. The *Accuracy*, *Precision*, *TPR*, and *FPR* metrics were computed for each experiment.

Moreover, the motion estimation results were validated by comparing the ground truth with the mean optical flow of the detected obstacle. The ground truth was obtained by projecting the obstacle pose in the image plane and computing the difference between previous and current projections. The ground truth pose of the obstacles was obtained by the motion capture system providing pose measurements at 120 Hz. In these experiments, the obstacles were thrown from a distance of 8 m to obtain longer trajectories in the image plane. 30 experiments were performed varying the direction of motion of the obstacles. The direction error corresponded to the angle difference between the ground truth and the estimated motion direction. The experimental results are summarized in Figure 4.7. The quartiles of the error are shown in Figure 4.7-a, while 4.7-b depicts the mean error and standard deviation along each experiment. Considering all experiments the absolute mean error was 6.97°, the mean standard deviation was 3.89°, and the maximum instantaneous direction error was 19.54°. Furthermore, the experimental results reported a Root Mean Square Error of 11.2°, which was a reasonable error to guide the robot in a collision-free direction. It is worth mentioning that the *Security Distance* b_u was designed to consider this error by increasing the obstacle geometry to enhance safety in the avoidance maneuver.

4.7.2 *Sense-and-avoid* evaluation

This Section describes the experimental validation of the *sense-and-avoid* scheme. First, a brief study of the minimum angles to avoid a collision risk situation between the robot and each of the three obstacles is presented. Eqs. 4.4 and 4.3 define the relationship between the minimum angles to prevent a collision and the obstacle depth $Z(t)$. Figure 4.8 shows the evolution of the angles ψ^* and θ^* for obstacle depths ranging

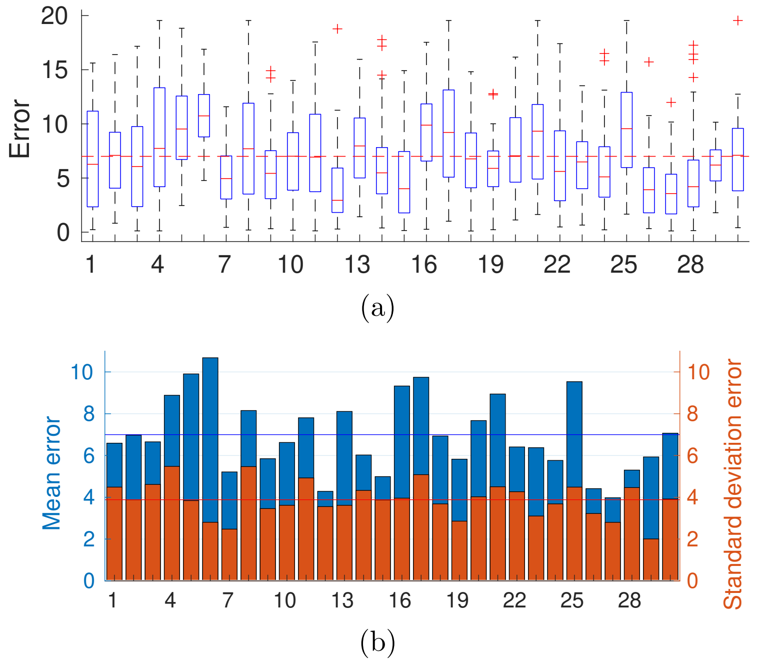


Figure 4.7: Summary of the motion estimation experiments. (a) box plot errors and (b) mean error and standard deviation. The directional error was defined as the mean angle difference between the estimated direction of motion and the ground truth.

between 1 m and 8 m. The curves were computed using the radius that enclosed each object; Small Box R =0.15 m, Stuffed Toy R =0.30 m, and Fitball R =0.39 m. The Figure shows that the closer the obstacle is to the robot, the more aggressive the evasive maneuver should be to evade the incoming obstacle.

The experimental setup consisted of launching the robot in a specific direction to perform a forward flight while an obstacle was thrown to collide with the ornithopter. The tests were performed in areas where the ornithopter could fly 10 m following a forward trajectory without any risk of colliding with static objects. The ornithopter was manually launched by an operator and described forward flights using the controller in [39]. Conversely, the obstacle was launched in different directions to intercept the robot using a launcher platform. The motorized launcher threw lightweight obstacles to enhance the repeatability of the experiments and set their initial velocity of the obstacle. The launcher was programmed to launch obstacles at velocities between 5 to 8 $\mathrm{m\,s^{-1}}$. The heavy object (i.e., Fitball) was launched by an operator with an

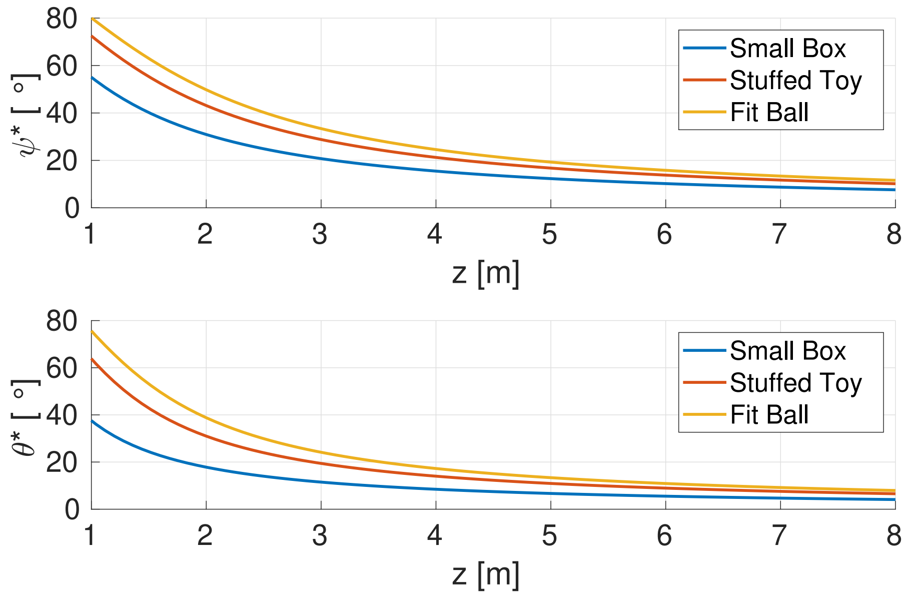


Figure 4.8: Evolution of ψ^* and θ^* by varying the distance between the obstacle and the robot (Z) for the three objects used in the experiments.

approximated speed of $5\,\mathrm{m\,s^{-1}}$. At each experiment, the proposed system evaluated possible collision risk situations and activate an evasive maneuver if necessary. Three *sense-and-avoid* evaluations were performed.

The first type of experiment consisted of analyzing the system performance when an *Intersection of the Safety Volumes (ISV)* occurred. An *ISV* arises when the artificial volume enclosing the obstacle and the volume of the robot intersect along the robot trajectory. To evaluate the *ISVs*, the robot was considered to follow a forward trajectory starting at its initial position and reaching a maximum velocity of $3.5\,\mathrm{m\,s^{-1}}$. The experiments were performed indoors where a motion capture system was used to estimate the robot's pose. Using the ornithopter and obstacle poses during the experiments the *ISVs* were validated. A total of 25 experiments were performed with each obstacle. Besides, two types of illumination conditions were considered in these experiments; regular indoor illumination (760 lx), and pitch dark conditions (<15 lx). The mean avoidance success rate with the three objects is shown in Table 4.3. The scheme reported an average success rate of 90.7%. The best results were obtained with the experiments conducted with the FitBall obstacle. Its larger size allowed earlier obstacle detections to prevent possible collision risk situations. The experimental results under pitch-dark conditions (dark in Table 4.3) were satisfactory. Their performance degradation was due to the additional noisy events which hampered obstacle detection.

Moreover, two possible situations may arise in case the obstacle detection method fails. First, a False Negative detection neglects the evaluation of a collision risk situation which may cause a collision between the obstacle and robot. Second, a False Positive obstacle detection may trigger an unnecessary evasive maneuver modifying the flight trajectory of the robot. This section analyzes the latter, by validating the scheme performance when ISV situations did not occur. 20 experiments were performed for this validation using only the Small Box, as it was the hardest obstacle to detect due to its size. Besides, its artificial volume was considerably small compared to the other obstacles reducing the chances to report an ISV. The system did not report any detection in 85% of the experiments. Thus, only in 15% of the experiments, the proposed scheme triggered an unnecessary evasive maneuver in flights with no ISV situations. This result was mainly due to the conservative selection of R' by enlarging the obstacle size to reduce impacts on the robot body. Moreover, the *False Positive* detection increased under pitch dark conditions due to the higher level of noisy events, similar to the results obtained in the ISV experiments.

	Validation	**Small Box**	**Stuffed Toy**	**FitBall**
Indoors	Optitrack	92%	92%	96%
Indoors Dark		84%	88%	92%
Outdoors	Visual	92%	92%	92%
Outdoors Dark		84%	84%	88%

Table 4.3: The success rate of the proposed dynamic *sense-and-avoid* scheme in the performed experiments.

Finally, the third set of experiments corresponded to the validation of the scheme in outdoor conditions. This validation aims at evaluating the robustness of the method in different scenarios including natural illumination conditions and scenes. The collision risk situations were evaluated visually in these experiments due to the lack of an external motion capture system. Besides, the mechanical launcher was not used in outdoor experiments due to the technical difficulties to install and calibrate it in the external scenario. 25 experiments were performed with each obstacle under light and dark pitch conditions. These results are reported as well in Table 4.3. The proposed method reported a success rate of 92.0% with daylight illumination conditions while reporting acceptable results (i.e., 85.3%) under dark lighting conditions. Figure 4.9 shows a sequence of an outdoor experiment where *E-Flap* evaded an obstacle under a collision risk situation.

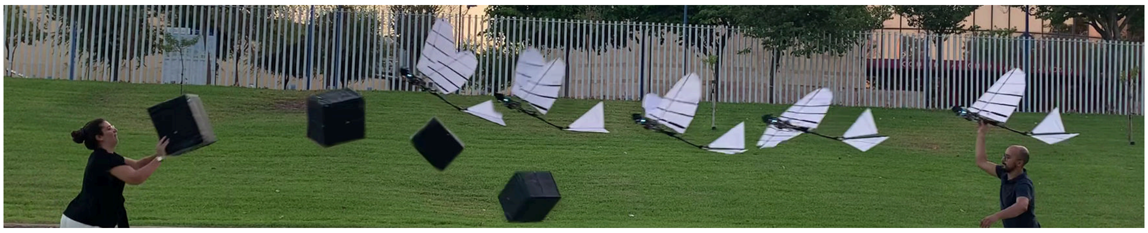


Figure 4.9: An example of a *sense-and-avoid* outdoor experiment, which corresponds to a sequence of images that includes the robot and the obstacle position during the experimental validation.

4.8 Conclusions

The development of *sense-and-avoid* schemes for large-scale flapping-wing robots presents complex challenges for traditional perception systems. The strict payload of ornithopters limits the installation of powerful computers, multiple sensors, and large batteries to feed the onboard electronics. Additionally, these platforms require perception methods capable to provide fast responses at the high velocities reached by large-scale ornithopters.

This Chapter presents a dynamic *sense-and-avoid* scheme carefully designed to run onboard a large-scale ornithopter. **To the best of our knowledge, it is the first event-based obstacle avoidance method for flapping-wing robots**. It relays on the intrinsic property of event cameras to react to the brightness variations produced by dynamic obstacles in the scene. The perception method performs *event-by-event* processing to leverage the microsecond resolution of event cameras. It allows fast onboard computation even in low-capacity hardware, providing high-rate estimations, 250 Hz in the presented experiments. The proposed scheme has been validated in indoor and outdoor scenarios with different illumination conditions. It reports an average success rate of 89.7% avoiding dynamic obstacles of different sizes and shapes.

The main limitation of the proposed approach is the detection of static obstacles in the scene. It has been designed to detect and avoid dynamic obstacles exploiting the advantages of event cameras. Future work focuses on integrating the proposed pipeline with a method to detect and evade static obstacles. The task may be addressed using either event data or frames from a standard camera. Moreover, future work focuses on exploring the use of the proposed scheme in other agile robots. The presented work aims at paving the way toward the development of obstacle avoidance techniques for large-scale ornithopter robots which have been barely studied in the literature.

Chapter 5

Event-based Intrusion Monitoring for Multirotors

5.1 Introduction

Multirotors are one of the most used robotic platforms in surveillance and monitoring applications. They offer several advantages compared to Unmanned Ground Vehicles (UGV) and other Unmanned Aerial Systems (UAS). First, multirotors can travel long distances and cover large areas in less time than ground robots. Second, they integrate several rotors (i.e., more than two lift-generating actuators) which enhance their maneuverability compared to fixed-wing robots. For instance, multirotors can hover which is a relevant feature in surveillance applications where the vehicles have to fly at low velocities to perform exhaustive monitoring. Third, multirotors are relatively cheaper than other platforms due to the high commercial demand for these platforms during the last decades. Finally, these platforms are easy to be deployed avoiding runways to take-off and landing.

The perception information collected onboard multirotors is mainly provided by frame-based cameras. Despite traditional frame-based cameras are the most adopted sensors for vision-based intrusion detection and large-area surveillance using UAS, these sensors experience significant issues in complex, and unstructured scenarios. Motion blur in highly dynamic or poorly-illuminated scenarios is a typical problem in frame-based vision. The blur effect hampers intrusion identification and detection by smearing the representation of intruders on the frame. This issue affects many UAS vision systems in applications such as fast autonomous navigation [152] or flapping-wing robot flight [42]. Furthermore, robustness to lighting conditions is a critical issue in outdoor tasks. It is often addressed by combining images from several onboard cameras (e.g., visual and infrared cameras), which increases the onboard weight, power consumption, and computational processing cost while reducing the UAS flight time. Conversely, the robustness of event cameras to the previous limitations has motivated their use for aerial robot perception. The integration of event-based vision on multirotor applications has been studied in different problems such as powerline tracking [153], autonomous landing [119], and obstacle avoidance [103].

This Chapter presents an event-based intrusion monitoring system for autonomous surveillance using multirotors. It contains two main perception modules. First, an event-based algorithm for Intrusion Monitoring (*IM*) that detects moving intruders under different illumination conditions and scene configurations. Second, an auto-tuning mechanism to adapt the parameters of the *IM* algorithm for different scene configurations. Both modules are integrated in an autonomous surveillance architecture to perform intrusion monitoring rounds using multirotors. The proposed system behaves similar to a robotic security guard exploiting the advantages of event cameras such as their high dynamic range to perform surveillance under different lighting conditions. Moreover, the research enclosed in this chapter intends to set an initial step toward the development of event-based algorithms for intrusion monitoring in flapping-wing robots.

This Chapter describes **Contribution 3**, and its software implementations are part of **Contribution 7** of this Ph.D. Thesis. Furthermore, the research conducted in this Chapter led to publications [60], [61], and [3].

This Chapter is organized as follows. Section 5.2 presents the main works in the topics addressed in the Chapter. Section 5.3 describes the intrusion detection algorithm which is the main perception method of the Chapter. Section 5.4 presents a scheme for autonomous intrusion monitoring enclosing an auto-tuning approach to adjust the parameters of the *IM* algorithm. The experimental validation of the

IM method and the auto-tuning scheme for surveillance is described in Section 5.5. Finally, Section 5.6 closes the Chapter with the conclusions of the presented work.

5.2 Related work

In the last decades, the use of multirotors has gained significant relevance in monitoring and surveillance applications [154]. These platforms offer several advantages such as long-distance coverage, simple assembly, fast deployment, and easy access to hazardous monitoring areas. Vision sensors are typically selected for these tasks as they capture visual information of the scene to detect anomalies in the area of interest. Several works have been proposed for monitoring and surveillance using multirotors. The work in [19] presents a system based on histogram equalization and RGB-Local Binary Pattern (RGB-LBP) operator for monitoring wide areas using small-scale Unmanned Aerial Vehicles (UAVs). The system detects changes among frames to use its output to perform a classification of the geometrical variations in the scene. The authors in [155] propose a multirotor-based surveillance system for parking lot monitoring and detection. The monitoring system uses Deep Learning (DL) to determine the number of vacant and occupied spots in a parking lot. In [156] a set of lightweight multirotors interact with a Wireless Sensor Network to improve border surveillance. The UAVs mount a camera to capture images and videos of the intruder in the scene to minimize the rate of false alerts on the network. Although these works offer suitable solutions for the monitoring task, they are prone to the typical limitations of frame-based cameras including motion blur, fixed frame rate, and reduced dynamic range, which hinder the capabilities of the monitoring system. Event cameras have attracted significant research interest in the robotics and computer vision communities in the last decade [16] providing a solution to the aforementioned limitations within a new vision-based perception paradigm.

Several works have integrated event cameras on multirotor platforms for different applications. The work in [157] presents a method to control the attitude of a dual-copter platform exploiting the microsecond resolution of event cameras. The method was extended in [158] where events are input directly to a neuromorphic chip and processed by a Spiking Neural Network (SNN) based controller. The work in [84] compensates the global motion of a Micro Aerial Vehicle (MAV) using the affine transformation model between two consecutive *event images* to separate events triggered by moving objects from those triggered by the background. An event-based method for powerline tracking running on board a multirotor is presented in [153]. It analyzes the spatial-temporal space of events to detect planes containing the powerlines. The method's performance is validated onboard a quadrotor flying in a scenario with powerline structures where the robot executes the powerline tracking task. Authors in [159] explore the use of event cameras for planetary robotics. They propose an event-based Visual Inertial Odometry (VIO) algorithm to cope with the low light and high vibration conditions that may occur on the Ingenuity helicopter [160] while exploring Mars. The method has been tested in a multirotor moving in a testbed arena emulating a Mars scenario.

The previous works process the event stream by accumulating events to generate *event images*. Thus, they do not fully exploit the asynchronous nature of event cameras. Conversely, *event-by-event* methods process single events leveraging the asynchronous feature of these sensors and reducing latency depending on their complexity and software implementation. Several *event-by-event* methods have been reported in the literature [73] [74] [71] [77]. However, those works correspond to low-level processing algorithms and their integration in applications for aerial robot perception in unstructured, realistic, and complex scenarios is still an under-researched area. Few works performing *event-by-event* processing onboard an aerial robot platform have been reported in the literature. The method in [122] estimates the 6 Degree of Freedom (DoF) pose of a quadrotor by tracking visual features from the event stream. Features are tracked asynchronously while the robot performs different flight maneuvers in an indoor structured scenario. Authors in [57] (see **Contribution 1**) present a line tracking method to provide visual references to guide a multirotor toward its goal configuration. Visual features are defined by lines in the scene which are asynchronously tracked from the event information. The method has been validated on a quadrotor platform equipped with an event camera that performs different landing trajectories. The work in [62] combines event data and frames for multirotor teleoperation. The operator is detected using frames with a classical state-of-the-art people detector and its gestures are determined by processing single events lying in a bounding box enclosing the operator's body. The detected gestures determine the commands to control the UAV motion in the working scenario. This Chapter presents an auto-tuning scheme for intrusion monitoring with multirotors. It includes an intrusion monitoring method that performs *event-by-event* processing to exploit the microsecond resolution of event cameras. The *IM* method is carefully designed to detect moving intruders in realistic scenarios. Besides, the scheme includes an auto-tuning mechanism to adjust the internal parameters of the *IM* module to improve its performance. Both methods are integrated into a complete surveillance system to perform autonomous monitoring missions with multirotor platforms. Finally, although developing an intrusion monitoring method for flapping-wing robots is out of the scope of this Ph.D. Thesis, this Chapter describes an initial study for the future design and implementation of an *IM* method for ornithopters.

5.3 Intrusion detection

This Section presents an event-based method for intrusion monitoring in complex and unstructured scenarios for multirotors. The background of the scenarios is assumed static, with only intruders moving in the monitoring area. That is the case in several applications such as night surveillance in factories or perimeter monitoring. The event camera onboard a moving multirotor triggers events from static objects and moving intruders which can be distinguished due to their different spatial-temporal properties. Our approach assumes that intruders originate nearby features in the event stream (e.g., produced by limbs in human intruders). Thus, intruders trigger groups of feature events with a globally consistent motion in the scenario.

The block diagram of the proposed method is shown in Figure 5.1. All modules perform *event-by-event* processing providing fully asynchronous intrusion monitoring. Our method adopts *event-by-event* processing to exploit the asynchronous nature of event cameras differently from *event image* methods that accumulate events in time windows or by collecting a fixed number of events. In the proposed approach, the events from a DAVIS346 camera onboard the multirotor platform are triggered asynchronously and processed by the robot's onboard computer. The event stream is encoded using the Address Event Representation (AER) providing pixel information with a resolution of 1 µs.

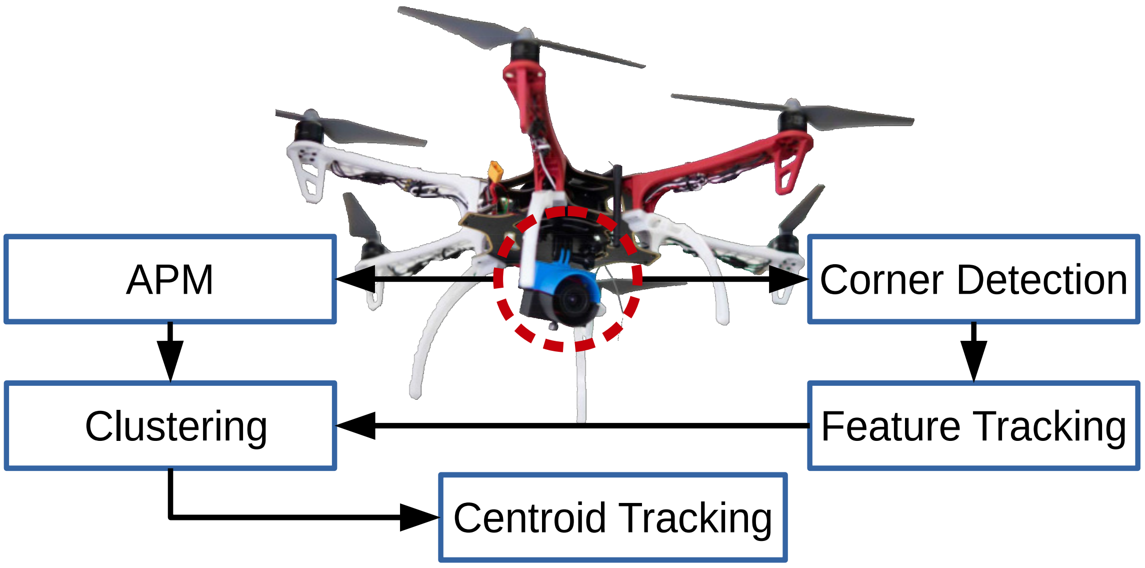


Figure 5.1: Block diagram of the intrusion monitoring method.

5.3.1 Intrusion monitoring algorithm

The diagram of Figure 5.1 depicts the four main modules of the Intrusion Monitoring *IM* algorithm. The first inputs the event stream and performs event-based corner detection. Among the corner detection methods available online, our method adopts *eFast [72], a modified version of the method in [74] with a remarkable trade-off between accuracy and computational efficiency. Next, the corners detected are separated by polarity and tracked to remove inconsistent, and noisy features. An example of the corner tracking output is shown in Figure 5.2-a. Although some asynchronous corner tracker methods have been proposed [51] [161], the presented algorithm adopts the feature tracker described in Section 2.3.1, adapted to our problem.

Moreover, the *Clustering* algorithm presented in Section 2.3.2 is adopted to create clusters of tracked features and nearby events with consistent motion. It includes

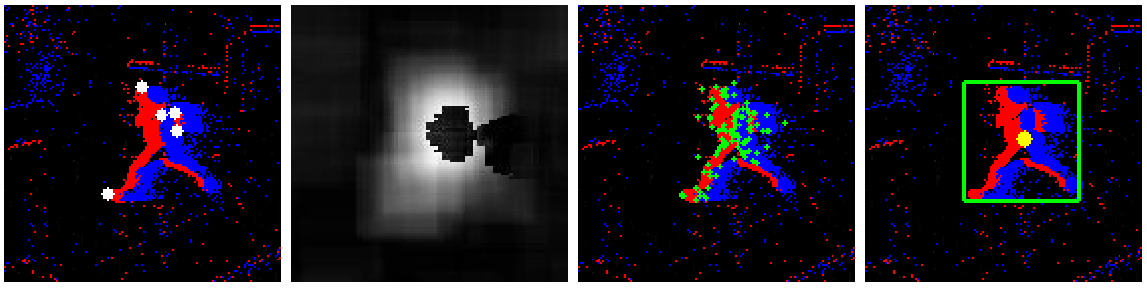


Figure 5.2: The output of each module of the *IM* method rendered in an *event image*: (a) corner tracking, (b) APM, (c) event clustering, and (d) intruder detection. These results were obtained by accumulating input events and the output of each module in a time window of 25 ms.

trade-off mechanisms to enable easy adaptation to different problems providing higher flexibility than other event-based clustering methods. Assuming a static event camera, the event clustering module might be sufficient for detecting intruders as events would be triggered only by the intruder's motion and the sensor's intrinsic noise. However, event cameras onboard multirotors are not static (even while hovering) and trigger many events produced by the static objects in the scenario. Our approach integrates the APM module proposed in Section 2.3.3 to capture the regions that trigger more events in the scene. Hence, the events triggered by moving objects are assigned with higher attention priority, see Figure 5.2-b, than those generated by static objects in the scenario. Besides, intruders create groups of event features with consistent motion in the scenario. The clustering module gathers events with high priority and event features from the *Tracking* module to enhance the robustness of the clustering method by using previous event features as a reference to gather new incoming events to the cluster. Figure 5.2-c. shows the events clustered by the *Clustering* module. Finally, the proposed algorithm tracks relevant clusters (i.e., the intruders) using an adapted version of the method of Section 2.3.1, which receives as input the centroid of the cluster. Figure 5.2-d depicts the intruder detected by highlighting it inside a green square. If a cluster contains less than λ samples or it is not updated in a consistent manner, it is considered noisy and discarded. Algorithm 5 summarizes the operation of the intrusion monitoring method. It inputs the event stream with events defined by the tuple $e(\mathbf{x}, ts, p)$, and outputs the centroid of the intruder detected (i.e., the centroid of the clusters in $\boldsymbol{C_T}$). The algorithm returns an empty list ($\boldsymbol{C_T} = \varnothing$) if no intrusion is detected and tracked.

Algorithm 5: Asynchronous event-based intrusion monitoring.

```
Input: $e(\mathbf{x}, ts, p)$
Output: $\boldsymbol{c_T}$
isCorner ← CornerDetection($e$)
if isCorner then
    $\boldsymbol{f_T}$ ←FeatureTracking($e$)              ▷ Asynchronous feature tracking.
end
if not APMFiltering(e) then
    $\boldsymbol{\mu}_c$ ←Clustering($e, \boldsymbol{f_T}$)             ▷ Asynchronous clustering.
    $\boldsymbol{C_T}$ ←CentroidTracking($\boldsymbol{\mu}_c$)          ▷ Cluster centroid tracking.
end
return $\boldsymbol{C_T}$                                ▷ Return tracked centroid.
```

5.3.2 Selective sampling of events

Event-by-event processing schemes exploit the asynchronous nature of event cameras. However, in some cases, they are highly computationally demanding since they process each event individually. This is a particular limitation in applications where additional events are triggered due to the camera motion or the noise produced in dark scenes. The *IM* module was carefully designed to enable its online and onboard execution even on hardware with low computational capacity. It includes a mechanism to reduce the computational load for real-time processing by selectively sampling the input events. The mechanism selects events using the parameter $\gamma \in [0, 1]$ as the percentage of input events being processed by the APM, and consequently by the clustering module. γ is set to adjust the computational cost of the method to run in real time on the computer mounted on the multirotor. However, it is worth mentioning that using the sampling mechanism might affect the performance of the intrusion monitoring method by reducing the number of events processed to detect the intruder. The advantages and disadvantages of using the selective sampling mechanism are discussed in Section 5.5.1.2.

5.3.3 Parameter dependencies

The methods described in Sections 2.3.1-2.3.3 include parameters that could require specific adjustments depending on the application. In the context of intrusion monitoring, some of these parameters depend on the structure of the scenario and the distance to the background, while other parameters represent a trade-off between computational cost and enhanced accuracy. Table 5.1 summarizes the set of parameters of the *Feature tracking*, *Clustering*, *APM*, and *IM* algorithms that directly influence the performance of the Intrusion Monitoring method. A set of empirical insights about setting each of the parameters is described below.

Parameter	**Module**	**Description**	**Typical range of values**
n^T	Feature tracking	Size of buffer $\mathbf{B}^T$	$[50, 200]$
κ_c	Clustering	Number of samples in the cluster for event-cluster proximity evaluation	$[10, 150]$
r	Clustering	Radius in event-cluster proximity evaluation	$[1, 50]$
n^C	Clustering	Size of buffer $\mathbf{B}^C$	$[20, 200]$
ω	*APM*	Sensitivity threshold to pay attention to a moving object	$[0, 1]$
l	*APM*	Size of the window used in the building of $\mathbf{\Omega}$	$[5, 40]$
n^A	*APM*	Size of buffer $\mathbf{B}^A$	$[50, 500]$
λ	*IM*	Minimum number of events per cluster to consider it an intrusion	$[0, 100]$

Table 5.1: Parameters of the *IM* method inherited from the algorithms of Section 2.

The buffer size n^T from the *Feature Tracking* algorithm should be chosen according to the complexity of the scenario and the camera motion. For instance, simple scenes (e.g., including few objects) and experiments describing slow camera motions require small buffers. Under these conditions, a large buffer increase the forgetting time horizon τ^T and the lifetime of outdated tracks leading to wrong matches between new features and old tracks. Conversely, cluttered scenes and experiments with high camera motions require large buffer values to avoid discarding tracks with a small forgetting horizon.

In the *Clustering* algorithm, the number of samples κ_c to evaluate event-cluster proximity with new events represents a trade-off between clustering accuracy and computational cost. Using many contour samples produces many proximity evaluations and increases the clustering accuracy. Conversely, using a small value of κ_c might lead to either discarding relevant event samples or appending noisy samples to the cluster. Thus, κ_c should be chosen considering the computational capabilities of the processing platform and the minimum required fault association error. Conversely, the buffer size n^C and the radius r in the event-cluster proximity evaluation depend on the scene and environment complexity. In dense scenes, small values of r are required to prevent wrong associations with events that belong to near clusters. In sparse scenes, large values of r are allowed as the majority of events are triggered by the intruder. Under this assumption, few κ_c samples are required to increase the algorithm robustness while reducing the number of proximity evaluations. Further, the size of $\mathbf{B}^C$ (n^C) should be chosen according to the scene complexity. Simple scenes containing only a few objects on a uniform background require small buffers as the majority of events are triggered by the intruder. Conversely, complex scenes require large buffers to accumulate enough events to describe the intruder. Additionally, the value of the threshold λ depends on the complexity of the scene and the mission. λ is defined as the minimum number of events per cluster to consider it an intrusion detection. Scenarios with short distances between the background and the camera require large values of λ as the shape of the intruder is described by a large number of events in the image plane. Otherwise, λ is set to medium-small values as the intruder tends to be represented by a lower number of events in the image plane. It is worth mentioning that the position of the camera in the scene is set to maximize the area covered by the camera Field of View (FoV) while avoiding collisions with the structures in the scene, e.g., walls, and objects, among others. Thus, it varies depending on the complexity of the scenario to analyze while performing the intrusion monitoring task.

Finally, three parameters of the *APM* depend on the complexity of the scene which impacts the spatial-temporal density of triggered events. These parameters are; (i) l which represents the size of the window to update $\mathbf{\Omega}$, (ii) ω, which determines the sensitivity threshold in $\mathbf{\Omega}$ to pay attention to moving objects, and (iii) n^A, the size of buffer $\mathbf{B}^A$. In general, simple scenes can be represented with a low-size buffer $\mathbf{B}^A$. Conversely, complex scenarios require larger buffers to describe the complexity of the scene. Further, using large buffers in sparse scenarios might result in a poor representation of the scene dynamics while setting unrealistic large areas of attention. A similar analysis can be done for parameters ω and l.

The manual tuning of the aforementioned parameters is not straightforward. It requires a good knowledge of the method functioning and spatial information of the scene to select a set of parameters suitable for each scenario. However, these parameters can be adjusted using a tuning method to reduce the time and effort devoted to this task. Section 5.4.3 presents an automatic tuning method that improves the intrusion monitoring accuracy while reducing the parameter tuning time.

5.4 Auto-tuning perception scheme for intrusion monitoring

This Section presents an auto-tuning scheme for autonomous intrusion monitoring with multirotors. It includes an auto-tuning mechanism to adjust the parameters of the *IM* algorithm to enhance the intrusion detection performance. The scheme integrates the event-based intrusion monitoring method of Section 5.3 which adjusts its parameters using the proposed auto-tuning mechanism. The proposed scheme is integrated into an aerial platform that performs periodic or on-demand surveillance tours programmed by the user. The background of the surveillance scenarios may vary from one tour to another, however, it is assumed that in the same tour the scenario remains mostly static. The intrusion monitoring scheme is robust to lighting conditions being operative in day and night scenes. Besides, it is robust to motion blur effects, which can be particularly severe in dark lighting conditions.

5.4.1 General scheme

The auto-tuning intrusion monitoring scheme is composed of four main functional modules. Figure 5.3 shows the block diagram of the proposed scheme. First, the

Navigation module includes submodules for localization, trajectory planning, and waypoint following. The localization is performed by fusing IMU (Inertial Measurement Unit) and RTK-GPS (Real Time Kinematics Global Positioning System) measurements from the sensors onboard the multirotor platform. The Lazy Theta* planner in [162] handles the trajectory planning. The architecture of the navigation module is typically used in different autonomous navigation works [163] [164]. Hence, it is not considered a contribution of this Ph.D. Thesis, and its detailed description is omitted. Second, the aerial platform mounts a DAVIS346 camera including a Dynamic Vision Sensor (DVS), a frame-based Active Pixel Sensor (APS), and an IMU. The frames and events provided by the DAVIS346 camera are the main perception data used for intrusion monitoring and auto-tuning. Third, at each surveillance mission, events are processed online and on board by the *Intrusion Monitoring* (*IM*) module of Section 5.3.1. It performs *event-by-event* processing for intrusion detection. Differently to the selective sampling method proposed in Section 5.3.2 to provide real-time processing, the auto-tuning scheme uses *ASAP* [150] package. *ASAP* synchronizes event packaging and processing by adjusting the number of events sent to the *IM* module such that the provided events are processed as soon as possible while avoiding processing overflow. Finally, at the parameter tuning stage, the parameters of *IM* are adjusted for a given surveillance scenario by the *Auto-tuning* module. This module implements a method based on Simulated Annealing (SA) and performs the parameter tuning task offline (i.e., on an external computer at the *Ground Station.*) using samples collected with different scene configurations, and under diverse weather and illuminations. The *Auto-tuning* method maximizes the similarity between the results provided by *IM* and a ground truth reference obtained by a state-of-art object detector (YOLO V3 [165]) using as input the grayscale frames from the APS.

The proposed system operates similarly to a "robotic security guard". The aerial robot performs surveillance tours for online intrusion monitoring, reporting, and logging. Sequences of waypoints define the robot monitoring trajectory, each waypoint is selected based on its good visibility and accessibility in the scenario. At each waypoint, the robot stays in hovering flight while performing intrusion monitoring using the *IM* method. Even on hovering flight, events are triggered by the static

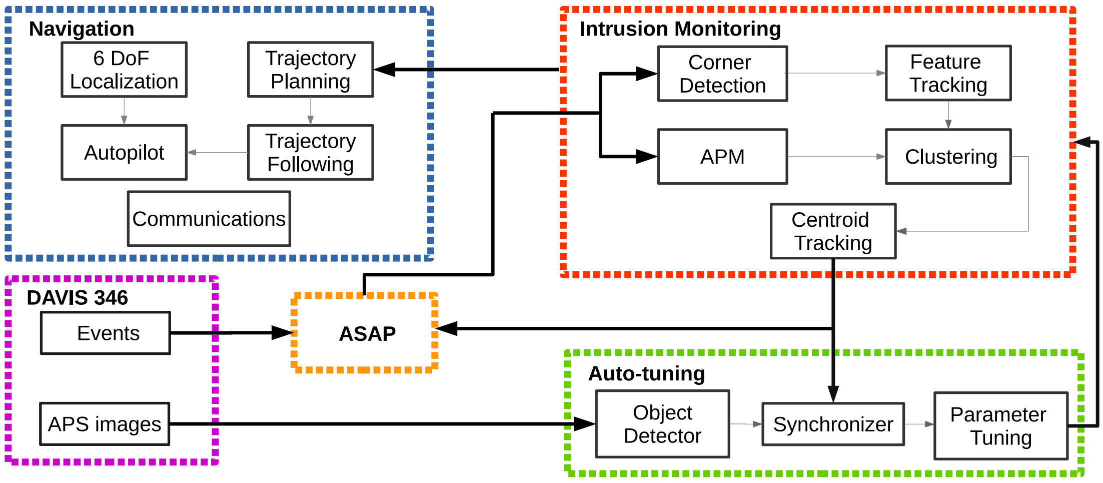


Figure 5.3: Block diagram of the intrusion monitoring scheme. It includes several modules (e.g., *Intrusion Monitoring* and *Auto-tuning*) and submodules such as the *Object Detector*, and the *Synchronizer*.

background due to multirotor motions and vibrations. The *IM* module includes specific mechanisms to distinguish between events produced by moving intruders and those triggered by the static background. If no intrusion is detected at that waypoint, the robot maintains its trajectory and moves to the next waypoint. In the case of intrusion detection, it is reported to the *Ground Station*. Depending on the surveillance policy adopted by the user, the robot can stay at the waypoint monitoring the zone with the detected intrusion, or it can continue the surveillance tour to avoid compromising the rest of the mission. The surveillance missions are performed by running the perception methods on the computer mounted on the UAV. Besides, the proposed architecture might support multi-robot coordination through the module *Communications*, however, the development and integration of a robot-coordination module is out of the scope of this Ph.D. Thesis.

Setting the *IM* parameters (see Section 5.3.3) depends on the object density, type, and complexity of the scenario, which in large environments can differ from one surveillance mission to another. To cope with this, the events and images collected at each tour are offline processed after the tour. The images from the APS sensor are processed by *Object Detector* submodule while the events from the DVS sensor are processed by *IM* module. The cases where *Object Detector* and *IM* disagree are submitted to an operator for additional validation. This procedure computes the *performance* of the *IM* module with the current set of parameters. In case of a low accuracy result by the *IM*, a new *Auto-tuning* process is activated updating the tuning data with the data obtained in the last tour. The parameter tuning stage is performed offline, on the *Ground Station*, and autonomously except for the cases in which the image-based *Object Detector* and event-based *IM* disagree.

5.4.2 Intrusion monitoring auto-tuning

A proper parameter adjustment of computer vision and Machine Learning (ML) algorithms is relevant to improve the methods' performance. A manual parameter selection leads to a large and inefficient iterative process. Besides, the complexity of the tuning process increases with the number of parameters for tuning. This Section presents a method to adjust the parameters of the *IM*. It only adjusts parameters that depend on the surveillance scenario such as the illumination conditions, and scene structure (i,e., the distribution of objects in the scene). The group of parameters unrelated to the scenario (e.g., κ_c, ν, A_T, and α_c) was set to fixed values in all the conducted experiments. This reduced the search space of the set of tuning parameters and the computational costs for parameter tuning. The set of parameters for automatic tuning is $\mathbf{v} = [n^T, \omega, l, n^A, r, n^C, \lambda]$. The dependency of $\mathbf{v}$ to the scene conditions was previously analyzed in Section 5.3.3. Besides, the search space for each parameter is shown in Table 5.1 in column *typical range of values.*

5.4.3 Simulated annealing

The proposed method adopts the metaheuristic Simulated Annealing (SA) algorithm [166] for parameter auto-tuning. SA was selected as it satisfies the requirements of the proposed *IM* problem [167]. First, SA focuses on approximating the global optimum solution in large search spaces. The search space of the *IM* problem is considered as a large search space by including seven parameters varying in different ranges of values. Second, unlike optimization methods based on gradient descent, SA provides robustness to local minima and does not require a known model of the parameters for being iteratively evaluated (e.g., [168]). This is particularly relevant for tuning the parameters of the *IM* module, as the problem does include a model relating the set of parameters with the *IM* performance. Furthermore, SA is suitable for problems that prioritize finding a global optimum approximation in a limited time instead of finding an accurate optimum in large time periods. This feature becomes relevant when the system requires re-training of the *IM* parameters between two consecutive surveillance tours. Finally, it is worth mentioning that the auto-tuning parameter scheme is adaptable to other black-box optimization methods, however, the evaluation of different optimization methods for parameter auto-tuning is out of the scope of this Ph.D. Thesis.

The SA algorithm uses the current solution $\hat{\mathbf{v}}$ to explore other solutions in the search space. Initial iterations consist of exploring distant solutions in the search space with high probability, this may not improve the current solution but allows a wide exploration of the search space. Conversely, at the last iterations, the algorithm searches at shorter distances admitting only solutions that improve the current solution. Besides, a perturbation $\Delta\mathbf{v}$ is added to the current solution $\hat{\mathbf{v}}$ at each iteration i to enhance the randomness of the search. Then, the cost function $J(\cdot)$ is evaluated using $\hat{\mathbf{v}} + \Delta\mathbf{v}$. The perturbation is sampled using a Gaussian distribution $N(0, \sigma\Upsilon_i)$, Υ_i is the temperature parameter controlling the annealing process, and σ is the perturbation standard deviation. The SA algorithm admits only solutions improving the validation of the cost function and prevents local minima by including a probabilistic mechanism to accept candidate solutions that do not improve the solution. A candidate solution $\mathbf{v}$ is accepted based on the Boltzman distribution as $p = e^{-\frac{\Delta J}{\Upsilon_i}}$, where $\Delta J = J(\hat{\mathbf{v}}) - J(\mathbf{v})$ is the cost difference between the candidate solution $\mathbf{v}$ and the current solution $\hat{\mathbf{v}}$. Large values of Υ_i lead to p close to 1. Conversely, small values of Υ_i lead to p close to 0. In this case, the search space is refined locally by accepting only the candidate solutions that improve $J(\hat{\mathbf{v}})$. Moreover, the adopted annealing controller is defined by the geometric model $\Upsilon_i = \Upsilon_0\eta^i$, where $\eta \in [0.7, 0.96]$ defines the annealing relation. This model is preferred over other annealing control schemes due to its remarkable performance in all experiments performed.

5.4.4 Parameter auto-tuning

The adopted optimization method aims at maximizing the similarity between the results of the *IM* and a ground truth reference. The ground truth is obtained from an automatic person detector which receives as input grayscale images from the APS of the DAVIS. The YOLO V3 [165] classifier was used for the person detection task. YOLO V3 is a fast-response object detector that processes the whole frame instead of using sliding windows or regions. The scheme of the auto-tuning method is shown in Figure 5.3. The *Object Detector* module inputs grayscale frames and outputs bounding boxes with the detected person. Conversely, the *IM* inputs the event stream and returns the centroid of the detected intrusions. The *Synchronizer* matches both outputs and compares them to evaluate the performance of *IM*. This comparison leads to four possible outcomes: (i) a True Positive (TP), when the object detector and *IM* report an intrusion and the distance between their centroids is lower than a threshold d_s; (ii) a False Positive (FP), when *IM* detects an intrusion which is not reported by the object detector; (iii) True Negative (TN), when the object detector and the *IM* report no intrusion; and (iv) False Negative (FN) which occurs in two cases, when *IM* does not report an intrusion while the object detector reports it, or when the distance between the centroids of both detected intrusions is greater than d_s.

Each auto-tuning process is described by a set of episodes, sequences of frames, and events recorded during multirotor flights in the intrusion monitoring scenario. The training approach uses episodes instead of single frames and batches of events as they capture the scene dynamics, and its complexity. The training set contains enough episodes to cover the range of conditions where *IM* should operate including object density and size, lighting conditions, and intruders in the scene, among others. At each SA iteration, the *Synchronizer* module estimates the *IM* performance using the current parameter configuration $\mathbf{v}$. The accuracy metric is used to compute the performance of the *IM* by:

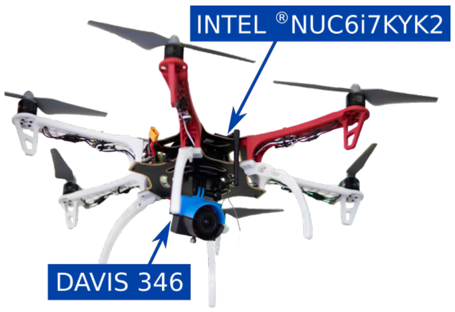
$$A(\mathbf{v}) = \frac{TP(\mathbf{v}) + TN(\mathbf{v})}{TP(\mathbf{v}) + FP(\mathbf{v}) + TN(\mathbf{v}) + FN(\mathbf{v})}, \tag{5.1}$$

where $TP(\mathbf{v})$, $TN(\mathbf{v})$, $FN(\mathbf{v})$, and $FP(\mathbf{v})$ correspond to the number of True Positives, True Negatives, False Negatives, and False Positives obtained by *IM* configured with parameter set $\mathbf{v}$.

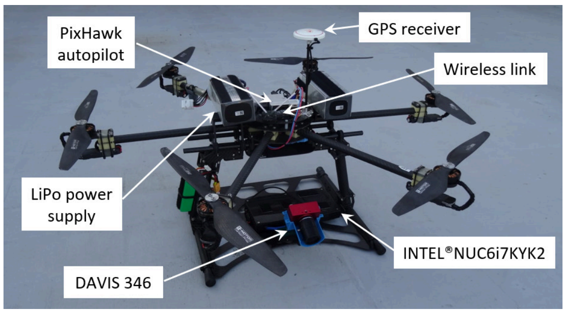
Moreover, the *Parameter Tuning* of Figure 5.3 corresponds to the SA algorithm. It evaluates the parameter set $\mathbf{v}$ using the cost function $J(\mathbf{v}) = 1 - A(\mathbf{v})$. At each iteration i, the *Object Detector* and *IM* process the tuning data. The *Synchronizer* computes the accuracy $A(\mathbf{v})$, which is subsequently used by the *Parameter Tuning* module to estimate and update the new parameter set $\hat{\mathbf{v}}$.

5.5 Experiments

This Section describes the experimental validation of the two main components of the event-based auto-tuning system for intrusion monitoring with multirotors. The validation experiments were performed in complex and unstructured scenarios with different illumination conditions, and with one or several intruders. Two multirotor platforms were used for the validation experiments. The first corresponds to a DJI Flamewheel F550, see Figure 5.4-a. It was equipped with a DAVIS346 pointing at a pitch angle of $-45°$ and an INTEL® NUC6i7KYK2 for logging and online computation. This first platform was used for validating the Intrusion Monitoring method in different scenarios with challenging illumination conditions and multiple intruders. A second platform was used to integrate and validate the autonomous system for surveillance and intrusion monitoring, which integrates the *IM* and auto-tuning modules. A large platform was used as the proposed solution requires additional sensors and extended flight time to perform the monitoring rounds. The second platform, see Figure 5.4-b, uses a custom-made frame endowed with a PixHawk 1 autopilot running a PX4 position-based low-level controller, a U-Blox Global Positioning System (GPS) receiver, and a DAVIS346 event camera mounted at $-45°$ pitch rotation. The platform integrates the INTEL® NUC6i7KYK2 embedded computer for onboard computation, and two LiPo batteries of 16 000 mA to enlarge the flight time of the aerial platform. The navigation system was implemented on top of the UAL abstraction layer [137] developed at the GRVC Robotics Laboratory.



(a)



(b)

Figure 5.4: Multirotor platforms used for the validation experiments. (a) DJI Flamewheel F550, and (b) a custom-made Hexarotor equipped with a PixHawk autopilot. Both platforms equip a DAVIS346 event camera mounted at $-45°$ pitch rotation.

5.5.1 Intrusion monitoring evaluation

These experiments describe the evaluation of the intrusion monitoring method proposed in Section 5.3.1. The experiments were performed at the laboratories of the School of Engineering of the University of Seville in two scenes with different background configurations and challenging illumination conditions. The platform used for the experiments was the customized DJI Flamewheel F550 which carried the onboard computer and the event camera. In these experiments, an expert user manually tuned the parameters described in Section 5.3.3 of the *IM* algorithm. The same set of tuned parameters was used for both scenarios in all daylight experiments. However, parameters n^C, n^A, and ω required additional tuning to adapt to the experiments with challenging illumination conditions.

A total of 36 outdoor experiments were performed. In each experiment, a human intruder moved into the monitoring area and tried to escape from the camera field of view simulating intrusion situations. First, the performance of the *IM* module on daylight conditions was analyzed. Figure 5.5-top-left depicts some monitoring results in day experiments. The DVS and APS outputs are overlaid for better visualization, and the intruder tracked is enclosed within a green window. It is worth mentioning that the *IM* algorithm processed only the events from DVS sensor in all experiments. The images from APS sensor were used only as ground truth for the method validation.

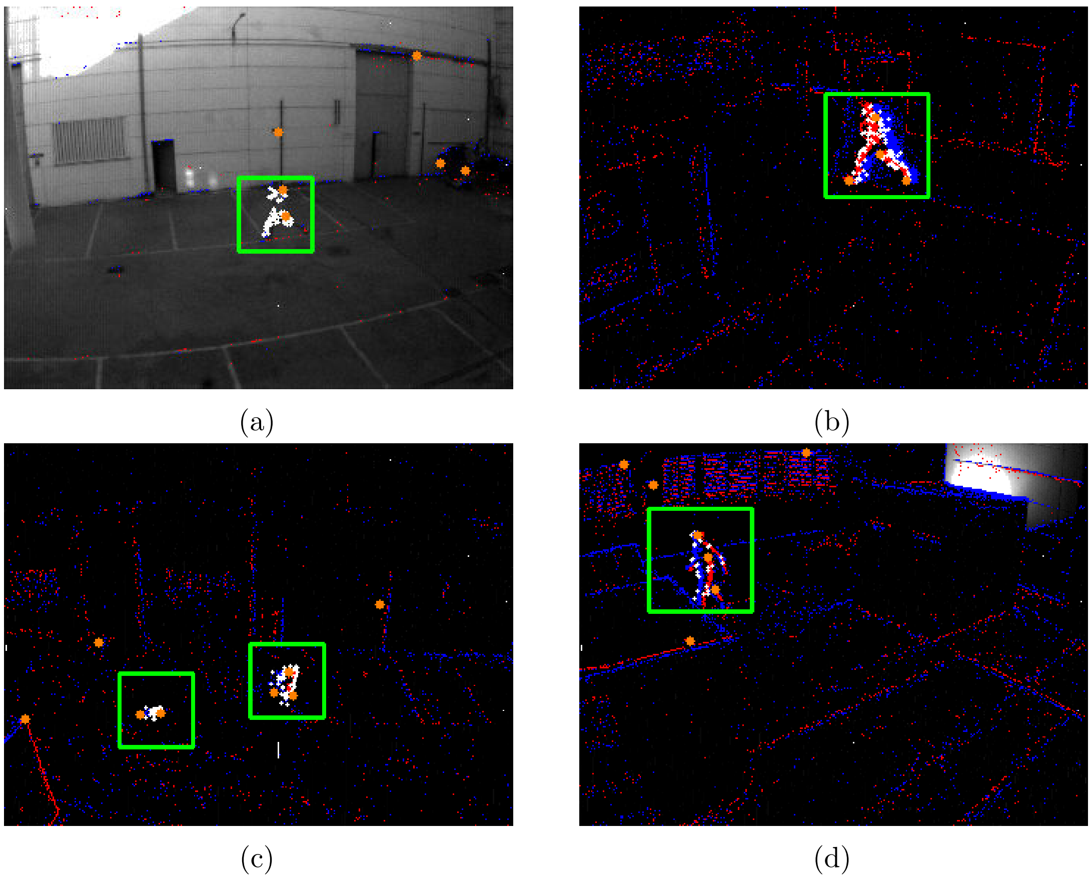


Figure 5.5: Results of the intruder monitoring evaluation in four experiments: (a) daylight, (b) night, (c) multi-object, and (d) illumination changes. The white and orange points represent the set of clustered events and events corresponding to feature tracks. These images were obtained by rendering the events, and *IM* results accumulated during the last 25 ms over the grayscale frames from the APS sensor.

The experimental evaluation was performed by comparing the output of the *IM* module with its respective ground truth (i.e., grayscale frame). The output of the *IM* algorithm and the ground truth images were temporally synchronized using their timestamps. Each comparison led to a possible result: False Positive (FP), False Negative (FN), True Positive (TP), and True Negative (TN). Counting the previous metrics the *Accuracy*, *Precision*, and *Recall* (also known as True Positive Rate (*TPR*)) metrics [169] were computed to evaluate the detection success rate and noise rejection capabilities of the method. These metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{P + N}, \tag{5.2}$$

$$Precision = \frac{TP}{TP + FP}, \tag{5.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

The average results of the *daylight* experiments are summarized in the second row of Table 5.2. The method reported an *Accuracy*=0.98, a *Precision*=0.99, and a *Recall*=0.97. The previous results validate the outstanding performance of the proposed method.

Table 5.2: Intrusion monitoring performance under different experimental conditions.

Experiment	**Precision**	**Recall**	**Accuracy**
Daylight	0.99	0.97	0.98
Night	0.97	0.96	0.97
Multi-track	0.96	0.96	0.95
Dynamic lighting	0.97	0.88	0.91

5.5.1.1 Robustness validation

The robustness of the method was validated in different conditions that typically occur in surveillance tasks. Three types of conditions were evaluated: i) experiments performed at *night*, ii) *multi-track-night* experiments emulating two intruders in the scene, and iii) *dynamic lighting* experiments with strong changes of illumination in the scenario produced by either moving or flashing lights in the scene.

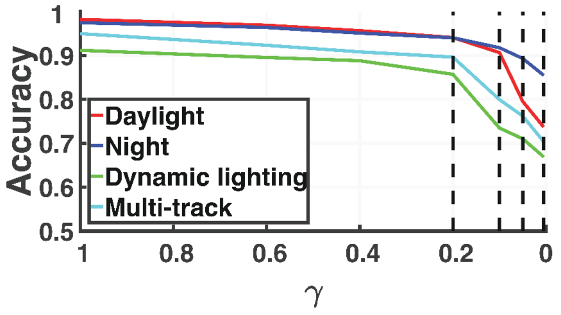
A summary of the results obtained with each condition is depicted in Figure 5.5. Additionally, Table 5.2 summarizes the average *Accuracy*, *Precision*, and *Recall* obtained for each condition. As was expected, the *night* experiments reported a lower signal-to-noise ratio (i.e., 1-2%) with respect to the *daylight* results.

Moreover, the multiple intruder tracking results reported some issues when one intruder moves significantly faster than the other. In this case, the APM assigned high priority to the intruder that moved considerably faster. This occurred very few times as both intruders moved at a similar speed during the majority of the experiments. Besides, the slight differences in *Accuracy* and *Precision* with respect to previous results were caused by the False Positives produced by noise. Further, the *dynamic lighting* experiments represented the most challenging conditions. These experiments were performed at night to increase the effect of the illumination changes on the scene. Two spotlight lamps were used to modify the light conditions. They either moved by mimicking a lighthouse or flashed irregularly. The lighting changes produced noisy events in the DVS sensor leading to False Positive and False Negatives. The latter were mainly due to the noise events that occurred around the intruder which affect the intruder detection. Although these effects reduced the method performance, the results still reported remarkable robustness given the challenges caused by the strong changes of illumination in the scene.

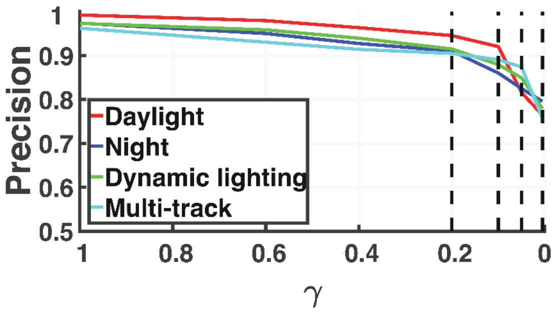
5.5.1.2 Real-time processing

The results of Table 5.2 were obtained by setting the selective sampling parameter $\gamma = 1.0$. Thus, all triggered events were processed by the intrusion monitoring method. This Section evaluates the use of parameter γ to adapt the method for real-time processing while keeping a good intruder accuracy detection. Figure 5.6 shows the *Accuracy*, *Precision*, and *Recall* obtained by using different values of γ for each type of experiment in Table 5.2. Each experiment was repeated 10 times and their results were averaged to cancel the randomness of the γ sampling. The Figure depicts that the metrics tend to degrade as the value of γ tends to zero. However, this performance decay is smooth and small for $\gamma \geq 0.2$. The results validate the use of the selective sampling approach to reduce the computational cost without significant degradation in performance. It is worth mentioning that at $\gamma = 0$, the clustering module receives only tracked corners from both moving and static objects which reduces the performance by increasing the number of False Positives.

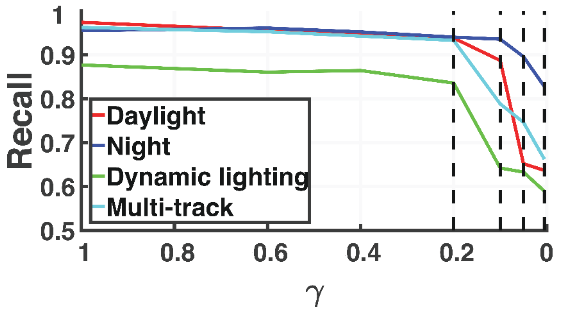
In the aforementioned experiments, each event was processed in $\sim 5.72\,\mu s$ in the onboard hardware. The camera triggered an average 115 k events per second, in the *daylight*, *night*, and *multi-track* experiments. Hence, events triggered in one second were processed in 0.66 s which allowed real-time processing in each experiment.



(a)



(b)



(c)

Figure 5.6: Performance results for all experiments. (a) *Accuracy*, (b) *Precision*, and (c) *Recall* as a function of γ.

Moreover, the method was capable of keeping remarkable results by processing only 20% of events with the clustering module without reporting significant performance degradation. A similar result was obtained in the *dynamic changes* experiments except when the spotlight lamp pointed toward the event camera. In this situation, the camera triggered $\gg$ 175 k events per second which caused some processing overloads. However, the γ sampling was effective by discarding events without significant overall performance degradation.

5.5.2 Auto-tuning surveillance scheme validation

The proposed auto-tuning scheme for intrusion monitoring is evaluated in this Section. The evaluation was divided into three parts. First, an initial evaluation of the method under *daylight* conditions is described in Section 5.5.2.1. Second, the performance of the method using two different auto-tuning approaches is evaluated in Section 5.5.2.2.

Third, the adaptability of the scheme to dark scenarios and its robustness to different illumination conditions is summarized in Section 5.5.2.3.

Unlike the previous experiments, the aerial platform used for this evaluation consisted of a bigger platform with a higher payload to mount additional sensors and batteries. The latter are necessary to extend the flight time of the UAV to complete the surveillance mission. Figure 5.4 shows the platform with its onboard hardware. The proposed scheme was implemented in C++ using Kinetic version of the Robot Operating System (ROS). The *IM* parameter auto-tuning was performed offline in a ground station computer with an NVIDIA GeForce GTX 1070 Ti Graphics Processing Unit (GPU) and an AMD Ryzen 5 2600 processor.

The experimental evaluation was conducted in the laboratories of the School of Engineering of the University of Seville. The surveillance missions consisted of flight trajectories defined by a set of control points (6 DoF waypoints) where the multirotor performs the intrusion monitoring task. Each control point corresponds to a surveillance zone. Each zone was determined by the mission requirements, the UAV location in the scene was chosen to maximize the monitoring area covered by the camera FoV while avoiding possible collisions with the structures in the scenario. A surveillance mission plan is shown in Figure 5.7 where the multirotor travels $\sim 420\,\mathrm{m}$ (round trip). The mission includes 15 surveillance zones (control points) with high variation in object distribution, size, and intrinsic motion. In particular, this mission includes a high diversity of objects including industrial objects such as containers, and vegetation such as trees and bushes. The path followed by the robot during the mission is shown in red, and the camera FoV at each control point is in blue.

Before evaluating the accuracy of the system, prior flights were performed to collect the necessary data for parameter auto-tuning. The conditions of the scenarios varied between flights. The datasets included recording with variations in the light (morning, midday, and night) and weather conditions. At each flight, the robot took off and started the trajectory. At each control point, the robot hovered while collecting sequences of $10\,\mathrm{s}$ of the DAVIS camera. 50% of the datasets were recorded with people emulating intrusion situations, while the other 50% without intruders to prevent biases in the parameter tuning process.

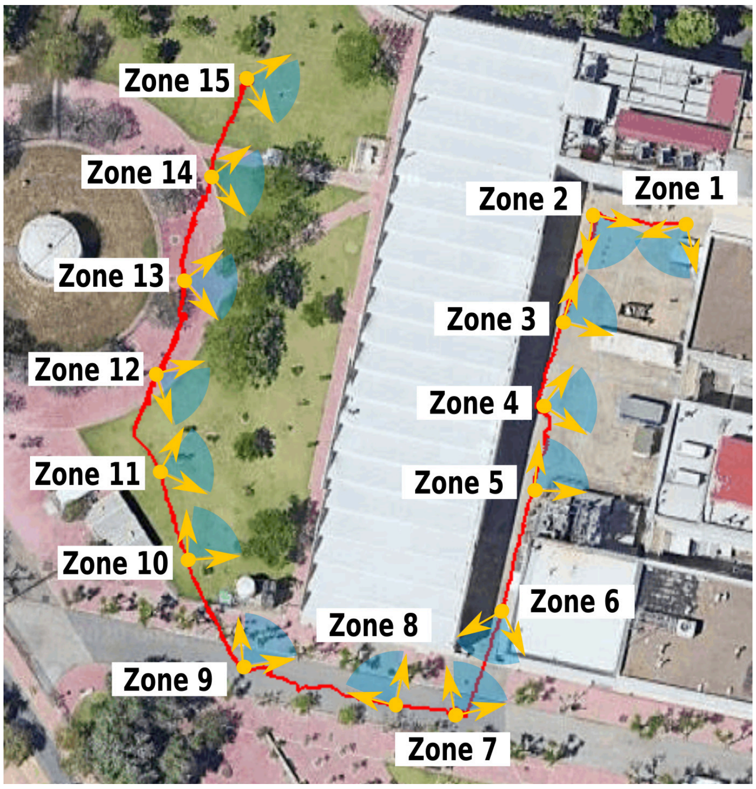


Figure 5.7: An example of a surveillance mission including scenarios with industrial objects and nature. The frames describe the 6 DoF control points of the robot in the monitoring mission. The blue semicircular areas describe the camera field of view at each control point.

After tuning the parameters, the scheme was evaluated in intrusion monitoring missions. At each waypoint, the robot performed online and onboard intrusion monitoring using the tuned set of parameters. The performance of the scheme was assessed using the metrics of Eqs. 5.2, 5.4, and the False Positive Rate (FPR) defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{5.5}$$

The submodules of the *IM* were carefully improved to reduce the computational cost and enhance fast processing. Moreover, the *IM* module includes several filtering steps. In the reported experiments the *APM* discarded 59% of the events which were considered as triggered by the background. These events were not analyzed by the *Clustering* module that represents ∼70% of the processing time of *IM*. Additionally, differently from Section 5.5.1 where a selective sampling method was proposed for real-time processing, the proposed scheme adopts *ASAP*, which dynamically adjusts the number of events processed by *IM* to avoid computational overflow. ASAP was configured to adjust the number of events minimum to 20% as it is the limit for the *IM* to provide good performance as reported in Section 5.5.1.2. The average event rate was $\sim$ 625 k events per second in the performed experiments. The combined filtering effect of the modules of the *IM* along with the ASAP integration allowed the processing hardware to execute the proposed scheme online and onboard.

5.5.2.1 Performance evaluation

This Section describes the performance evaluation of the auto-tuning method by adjusting the *IM* parameters for each zone of the surveillance area. The evaluation was performed in different zones with samples recorded under *daylight* conditions. Figure 5.8 shows some snapshots of each zone depicting the type of scenario and the distribution of objects in the scene. This approach aims at capturing the particularities of each zone by tuning *IM* parameters with samples collected at each zone. Thus, the approach implements a *zone-dependent* auto-tuned parameter tuning where the robot changes the *IM* parameter set from one zone to another.

The SA optimization algorithm was configured with parameters $\Upsilon_0 = 1.0$, $\eta = 0.955$, and $\sigma = 0.35$ for all experiments. These parameters were chosen such that Υ_i tends to zero in a fixed number of epochs. The number of epochs is crucial in applications requiring fast deployment such as intrusion monitoring with UAS. Performing many epochs may require an unreasonable amount of time, whereas performing a few iterations might cause inaccurate parameter tuning. Using 150 epochs was found as a suitable trade-off between tuning time and accuracy for all the experiments. For all experiments, an epoch consisted of processing a 10 s sequence of events and frames.

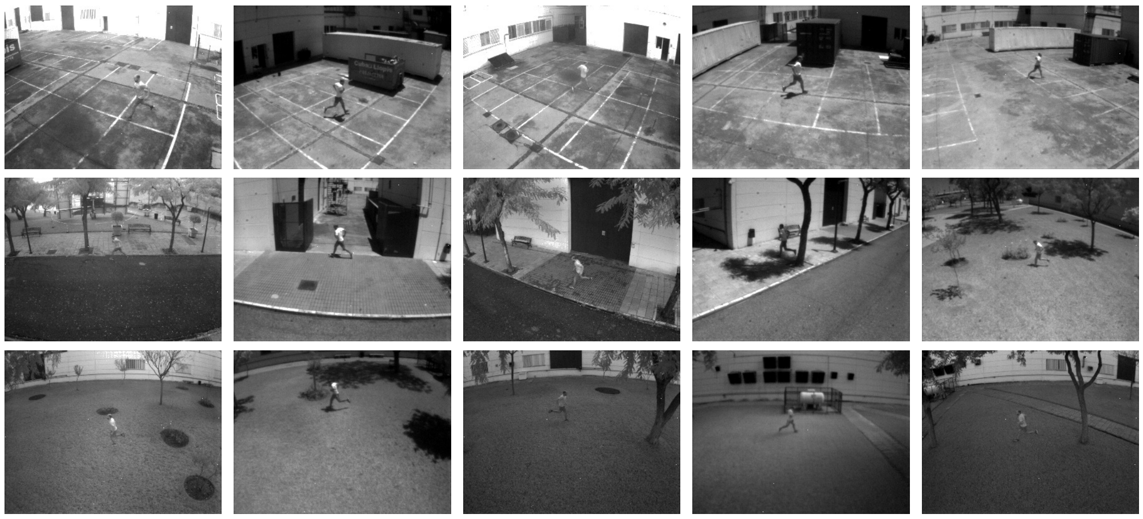


Figure 5.8: Snapshots of the 15 surveillance zones (ordered from left to right, and from top to bottom) of the surveillance mission shown in Figure 5.7.

Each auto-tuning process began with an initial set of parameters, which were randomly selected within their parameter bounds (see Table 5.1). At each new epoch, the cost function $J(\cdot)$ was evaluated for the current candidate set of parameters $\mathbf{v}$ and used to compute ΔJ. Afterward, a new set of parameters $\mathbf{v}$ was selected to perform further exploration aiming at retrieving configurations that reduce the error of the cost function. Figure 5.9 depicts the evolution of the values of $J(\hat{\mathbf{v}})$ and $J(\mathbf{v})$ during the parameter tuning process for zone 7. $J(\mathbf{v})$ represents the cost obtained for each candidate configuration $\mathbf{v}$ along the tuning process. Conversely, $J(\hat{\mathbf{v}})$ describes the cost obtained using $\hat{\mathbf{v}}$ as prior to estimate the next set of candidates $\mathbf{v}$ at each epoch. The Figure depicts that $J(\hat{\mathbf{v}})$ only decreases when a new set of candidate parameters report a cost lower than the cost obtained with $\hat{\mathbf{v}}$. Figure 5.10 depicts the evolution of $J(\hat{\mathbf{v}})$ during the auto-tuning process for each zone of the surveillance area. The initial cost among the different zones differs due to random parameter initialization. As it was mentioned in Section 5.4.3, $J(\hat{\mathbf{v}})$ may increase during the first epochs (i.e., when Υ_i is close to 1) to enhance the exploration of the search space and prevent local minima. Figure 5.10 shows the fast convergence of the tuning method. After running 20 epochs, the cost function went below 0.2 for all zones. Besides, after the tuning process was completed (i.e.,150 epochs), the cost function reached a value 0.1 in all experiments.

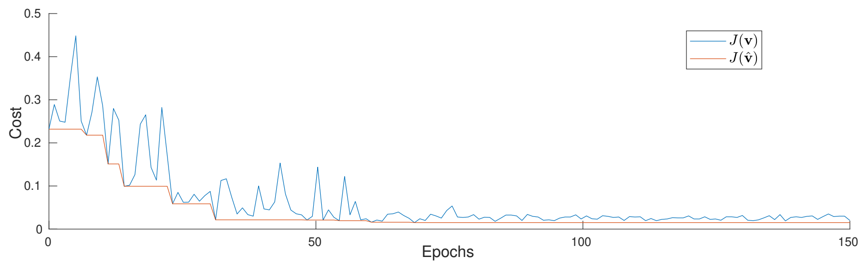


Figure 5.9: Evolution of the cost functions $J(\hat{\mathbf{v}})$ and $J(\mathbf{v})$ for the parameter tuning process of zone 7. $J(\mathbf{v})$ represents the performance of the solutions analyzed along the exploration of the search space.

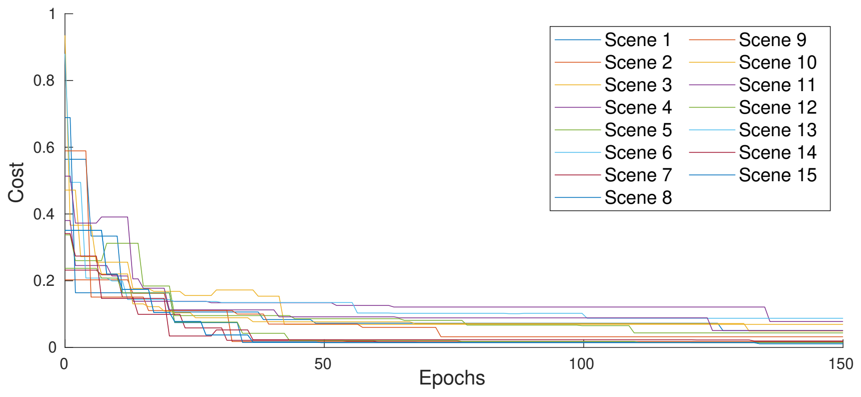


Figure 5.10: The evolution of $J(\hat{\mathbf{v}})$ for each zone during the tuning process.

After tuning a set of parameters for each zone, more than 30 evaluation missions were performed under different illumination and weather conditions. Table 5.3 summarizes the average performance obtained at each zone. The tuned scheme reported an accuracy of 96%. Besides, the *FPR* and *Accuracy* results indicate a low number of false alarms evidencing the high False Positive rejection of the scheme. Further, the *TPR* outcomes suggest reasonable missing detection capabilities.

Zone	*Accuracy (%)*	*Precision (%)*	*TPR (%)*	*FPR (%)*
1	95	97	87	1
2	98	97	96	1
3	96	90	90	5
4	92	94	85	3
5	96	96	91	2
6	91	92	83	4
7	98	98	97	1
8	98	98	98	1
9	95	92	94	3
10	97	94	97	3
11	95	96	92	2
12	95	96	90	1
13	99	98	98	1
14	98	98	95	1
15	97	98	95	1

Table 5.3: Performance of the method in each zone using the *zone-dependent* auto-tuned parameters.

5.5.2.2 Analysis of zone sensitivity

This Section describes the validation of the proposed scheme using two different tuning approaches. The first uses the same set of parameters $\mathbf{v}'$ for all zones in the surveillance area. Unlike the approach in Section 5.5.2.1, this approach uses a single *zone-independent* parameter set. It uses a single dataset including sequences from all zones in the mission for tuning $\mathbf{v}'$. Thus, at each epoch, several random sequences recorded in different zones of the surveillance area are used for parameter tuning. Second, a *parameter averaging* approach is validated. Similarly to the *zone-independent* approach, it uses a single parameter set for all zones. However, this single parameter set is computed by averaging the parameters previously tuned for every single zone of the surveillance area (i.e., the parameters adjusted following the approach in Section 5.5.2.1). The experiments performed with both approaches used the SA parameter configuration $\Upsilon_0 = 1.0$, $\eta = 0.955$, and $\sigma = 0.35$.

	Zone-independent				*Parameter averaging*			
Scene	*Accuracy (%)*	*Precision (%)*	*TPR (%)*	*FPR (%)*	*Accuracy (%)*	*Precision (%)*	*TPR (%)*	*FPR (%)*
1	91	99	72	1	91	97	74	1
2	95	96	84	1	92	76	95	9
3	80	84	60	7	67	54	62	30
4	86	96	66	2	73	60	83	32
5	88	98	74	1	86	89	73	6
6	87	96	68	2	81	71	81	19
7	96	96	94	2	90	79	98	14
8	97	99	93	1	95	90	97	7
9	93	92	88	4	94	88	97	7
10	92	96	84	3	86	80	89	15
11	88	96	75	2	86	88	74	6
12	93	98	85	1	82	97	60	1
13	99	98	98	1	98	97	97	1
14	93	98	81	1	89	98	70	1
15	92	96	85	1	91	96	81	1

Table 5.4: Evaluation of the *IM* performance by tuning the set of parameters following the *zone-independent* (left), and the *parameter set averaging* (right) approaches.

The results obtained with both approaches are summarized in Table 5.4. The second column corresponds to the *zone-independent* approach, while the third column describes the results obtained with the *parameter averaging* approach. The results obtained with the *zone-independent* approach reported 5% greater accuracy than the results obtained with the *parameter averaging* approach. In general *parameter averaging* reported satisfactory results in some zones while poor performance in zones with higher complexity such as zones 3 and 4. Conversely, the *zone-independent* approach reported an average accuracy of 91%, and single-zone accuracy higher than 80% for all zones. Moreover, despite the *zone-independent* approach reported a performance lower than the *zone-dependent* approach, it provided satisfactory results while involving a simpler tuning process, enhancing scalability to large surveillance areas. Thus, the *zone-independent* approach proposes a trade-off solution for large scenarios when all the zones of a surveillance mission are relatively homogeneous, whereas the *zone-dependent* approach provides a suitable parameter solution otherwise.

5.5.2.3 Performance under low illumination conditions

Previous experimental validation shows the outstanding capabilities of the auto-tuning scheme in experiments performed under *daylight* conditions. Conversely, this Section focuses on analyzing the robustness of the scheme in experiments performed in pitch-dark conditions typically occurring at night.

Initially, the scheme was validated with the set of *zone-independent* parameters tuned with daylight sequences. The performance of the method was validated in sequences recorded at the same zone under pitch-dark conditions. The experimental results are summarized in Table 5.5-top. The accuracy values obtained in the different zones varied in the range from 73% to 84%. These results reported a significantly lower performance than the performance obtained in *daylight* experiments. Furthermore, the *TPR* reported worse results due to the high number of False Negatives produced by the increment of noise in the event stream. Additionally, the *Precision* and *FPR* statistics depict good results as these metrics are independent of the number of False Negatives. Despite this initial approach did not provide satisfactory results, it serves as a first step towards exploring other solutions for adapting the parameters obtained with *daylight* data to the data collected under pitch-dark conditions.

Parameter set	*Accuracy* (%)	*Precision* (%)	*TPR* (%)	*FPR* (%)
Parameters trained with daylight data	82	98	62	2
Parameters trained with dark lighting data	94	94	92	5
Parameters trained with daylight data adapted to night conditions	92	93	89	5

Table 5.5: Performance of the proposed scheme in pitch dark conditions using parameters tuned with: top) daylight data, center) pitch dark samples, and bottom) daylight data adapted to dark conditions.

Unlike the previous approach, an intuitive solution is to focus on tuning the parameters using the data collected on missions performed in pitch-dark conditions. However, this approach misses the point of the proposed scheme by requiring a manual annotation of the ground truth as YOLO poorly performs with grayscale images collected under pitch-dark conditions. Besides, manually annotating the collected data requires expertise with event-based data as the collected frames are almost useless under these conditions. This may imply additional training for the operator in charge of setting the mission parameters and delays for the monitoring mission. Conversely, this subsection focuses on adapting the parameters tuned for daylight conditions to the pitch-dark situation. To understand the problem, the data collected in both illumination conditions were extensively analyzed. First, in the experiments performed in all zones, the number of events triggered under dark lighting conditions was at least twice greater than those generated under daylight conditions. More than 60% of these events were triggered around the edges of the objects in the scene while the rest corresponded to noise distributed around the image plane. This noise phenomenon was previously reported in [84]. Second, an additional analysis between the parameters tuned in *daylight* conditions and those tuned with data collected in pitch-dark conditions was performed. For this analysis, manual ground truth was extracted directly from event data for the sequences recorded in each zone. The performance results using manually annotated ground truth with dark illuminated conditions samples are summarized in the third row of Table 5.5. The main differences between the tuned parameters occurred in ω, n^C, and n^A. For instance, the value of n^A (i.e., the size of the *APM* buffer) was four times greater in dark light conditions than in the *daylight* experiments. Moreover, ω (the *APM* threshold) increased ∼30% in dark lighting experiments with respect to the values reported in *daylight* conditions. This was mainly due to the higher noise level, the *APM* increased its filtering effect to enhance the restrictions in order to consider an event as originated by an intruder. Conversely, n^A and n^C (i.e., the cluster buffer) reported smaller values in dark lighting conditions, 50% smaller than in *daylight* experiments. The higher noise level in dark lighting experiments caused large clusters including noise. Low values of n^C partially discarded the noise by limiting the size of these clusters.

The bottom part of Table 5.5 depicts the performance obtained by using adapted parameters to dark lighting conditions. Despite the performance being slightly worse than the performance obtained with manually annotated ground truth, the parameter adaptation offers a trade-off solution between keeping high performance in both illumination conditions without requiring additional tuning and annotated data for pitch-dark conditions.

5.6 Conclusions

Event cameras provide several advantages for intrusion monitoring applications with aerial robots. First, they directly provide pixel information of moving objects. Second, their high dynamic range allows day and night monitoring operations without requiring additional hardware or sensor modification. Finally, these sensors are robust to motion blur, which may arise during aerial robot flight.

This Chapter proposes an event-based auto-tuning scheme for intrusion monitoring and surveillance with multirotors. The proposed scheme includes two main modules. First, an event-based method to detect intrusions using only event data. It performs *event-by-event* processing to exploit the asynchronous nature of event cameras. The *IM* module integrates different event-based algorithms for corner detection, tracking, and clustering. The second module is an offline semi-supervised mechanism to adjust the parameters of *IM* to a particular scenario and problem. It maximizes the similarity between the output of the *IM* algorithm and the ground truth given by a state-of-the-art object detector. Both perception modules were implemented in ROS and integrated into a fully autonomous architecture for intrusion monitoring. The proposed architecture was validated in challenging scenarios with a wide variety of illumination conditions, including day and night experiments.

A limitation of the proposed scheme is that it does not identify the type of intruder (e.g., a person, a car, and a bike, among others). Intruder identification using event-based Deep Learning techniques is the object of future research. Indeed, the presented work served as an inspiration to the development of DL methods for person detection using event data [170]. Furthermore, a scaling parameter approach is proposed to deal with the additional challenges arising by performing *IM* in dark illumination conditions. Future work focuses on studying event data generation under different lighting conditions. This study aims at finding a set of general considerations for adapting the parameters of event-based algorithms (e.g., the proposed *IM*) to the challenges presented in pitch-dark scenarios. Finally, future work aims at providing an intrusion monitoring solution for flapping-wing robots. The current *IM* approach may serve as an inspiration for this work, however, the new method may require specialized algorithms to distinguish events triggered by moving intruders, particularly when the robot moves faster than the intruder. Additionally, the method has to deal with the high vibrations caused by the flapping strokes during flight.

Chapter 6

Datasets and Simulation

6.1 Introduction

The potential advantages of flapping-wing robots over rotary-wing and fixed-wing platforms such as low energy consumption and safety in populated environments have motivated significant Research and Development (R&D) efforts. These technological advances have resulted in the development of small-scale [56], mid-scale [44], and large-scale [34] ornithopters. Nonetheless, very few of the reported flapping-wing platforms include onboard sensing and processing hardware for robot perception due to their payload and size restrictions. Thus, there is an ample gap in the development and integration of autonomous perception systems for ornithopter robots.

The development of perception systems for flapping-wing robots requires the study and analysis of the sensing information produced during the flight of these platforms. Ornithopters report relevant challenges for robotic perception mainly due to high flight velocities, and the mechanical vibrations caused by downward and upward flapping strokes. Besides, ornithopters report a reduced payload which hinders the installation of perception sensors and mechanical stabilizers to collect perception information on board these platforms. Despite there are several datasets for aerial robot perception, none of them included sensing information recorded onboard flapping-wing robots until the development of this Ph.D. Thesis. Moreover, the aerodynamic complexity of ornithopters limits the accurate simulation of the platform dynamics and their effect on the measurements gathered by onboard sensors. Hence, additional efforts are required to provide real or simulated perception data to pave the way toward the development of perception systems for ornithopters.

This Chapter aims at reducing the gap between robotic perception and flapping-wing flight by presenting two tools for ornithopter robot perception, a simulation tool, and a perception-oriented dataset. The first is a multi-sensor simulator developed to emulate the sensing information captured during the execution of bioinspired landing maneuvers. It includes a bioinspired trajectory generator that relies on *tau-theory* to compute landing trajectories while the simulator scheme provides measurements from different perception sensors. The second tool is a perception dataset recorded onboard a large-scale flapping-wing robot. The dataset includes measurements from an event camera, a conventional camera, and two Inertial Measurement Units (IMUs), along with ground truth pose information from a laser tracker or a motion capture system.

This Chapter describes **Contribution 4** and **Contribution 5** of this Ph.D. Thesis. Besides, the experimental validation on the ornithopter and the software tools are part of **Contribution 6** and **Contribution 7**. Moreover, the work presented in this Chapter corresponds to publications [64], [63], and [43].

This Chapter is organized as follows. Section 6.2 summarizes the main works related to the topics addressed in this Chapter. Section 6.3 presents the proposed sensing simulation scheme for bioinspired landing. It describes the main modules of the simulator together with a synthetic dataset recorded in different scenarios. Section 6.4 presents the GRIFFIN perception dataset. The dataset has been recorded onboard a real ornithopter and it includes sensing measurements relevant sensors for flapping-wing perception. Finally, Section 6.5 presents the conclusions related to the work presented in the Chapter.

6.2 Related work

Robotic simulators are suitable tools to test and validate robot dynamics, control policies, perception algorithms, multi-robot strategies, among others, before performing experiments on real platforms. Simulators enhance safety, reduce costs, and decrease evaluation time by emulating the behavior of robots without endangering the robotic platform. V-REP [171] and Gazebo [172] are some of the most popular robotic simulators. The first is a simulation tool for validating and designing robots and simple sensors. The second includes several robot models and sensors, and it is integrated with the Robot Operating System (ROS) framework simplifying the use of additional libraries and packages for robotics. Game engines have also been used for robotic simulation purposes. Engines are preferred in computer vision, Machine Learning (ML), and augmented reality applications due to their photorealistic rendering capabilities. Airsim [173] and CARLA [174] are some of the robotics simulators based on game engines.

Few event camera simulators have been reported in the literature. The work in [175], computes the difference between consecutive images to generate edges that resemble the events produced by the edges of moving objects. The simulator in [93] uses a 3D computer graphic software to sample images at high rates to emulate a continuous time frame generation. Events are generated by analyzing the intensity difference between consecutive frames similar to [175]. The work in [176] introduces ESIM. The simulator relies on high sensing rates of a game engine to simulate the asynchronous behavior of the event cameras and trigger events based on the prediction of the camera motion. ESIM can be integrated into both Airsim and CARLA.

Current research in flapping-wing robot simulators mainly focuses on the study of flight control and robot dynamics. In [177], the controllability and stability of the robot are analyzed to control the non-linear flight of an ornithopter. The multi-body dynamics of a flapping-wing robot are simulated in [178] including fluid-structure interaction and flight dynamic behavior to design a robotic model capable of flying in trim. The dynamic simulator of the Flappy hummingbird [179] is an open-source tool to facilitate the design and validation of flight control architectures for small-scale ornithopter robots. Despite these tools simulate flapping-wing robot dynamics, kinematics, and aerodynamic effects, none of them provides perception information collected during the robot flight.

Moreover, several UAV datasets have been introduced for robotic perception tasks. The *AU-AIR* dataset [180] provides annotated data of traffic surveillance onboard an Unmanned Aerial Vehicle (UAV) together with data from a frame-based camera, a Global Positioning System (GPS), and an IMU. The work in [181] presents a dataset including real and photo-realistic synthetic data to evaluate place recognition methods with respect to viewpoint tolerance. The *Mid-Air* dataset [182] presents a large collection of synthetic perception data recorded onboard quadcopters flying in unstructured scenarios. The datasets were recorded by simulating different weather and illumination conditions using the Airsim simulator. A summary of the main reported datasets for aerial robot perception is shown in Table 6.1. The majority of these were collected on Micro Aerial Vehicles (MAVs) and multirotors. The MAV dataset in [183] includes onboard sensor information collected in flights within urban streets. The *EuRoC* dataset [184] provides data on MAV flights in two different indoor scenarios: an industrial scenario for evaluating visual-inertial localization, and a room equipped with a motion capture system for evaluating 3D reconstruction techniques. The *Blackbird UAV dataset* [185] approaches the problem of agile and autonomous operation of aerial vehicles in outdoor environments with special emphasis on visual inertial navigation, 3D reconstruction, and depth estimation. The data from real flights has been extended by generating additional synthetic data through simulation. A dataset of fast flights using a quadrotor in an outdoor scenario is presented in [186]. Four different flights were performed in an airport runway repeating the same trajectory at four different speeds in the range [5,17.5] $\mathrm{m\,s^{-1}}$. All the above datasets include high-resolution images, GPS, and IMU data. Some of them also provide additional measurements, such as stereo vision data (the *EuRoC* dataset) or rotor tachometer information, and depth images (the *Blackbird UAV* dataset). However, none of these datasets provide sensor measurements collected on a flapping-wing robot. Thus, they do not provide the type of sensing measurements that arise during flapping-wing flight.

Few datasets have been reported in the literature including any sensor information collected during the flight of ornithopter robots. The work in [39] presents a dataset useful for flapping-wing robot control. It includes experimental control data of an ornithopter performing landing maneuvers in a scenario equipped with a motion capture system, however, it lacks onboard sensing data for robot perception. The authors in [187] provide inertial information from several ornithopter gliding flights under low-wind conditions. Besides, the work includes frames recorded from three different views to triangulate the robot's position. However, the dataset does not include any data from perception sensors mounted on board the flapping-wing robot. To the best of the author's knowledge, no dataset with experimental onboard measurements suitable for flapping-wing perception has been reported in the literature before this Ph.D. Thesis.

The strict weight distribution and payload capacity restrictions of ornithopters entail a careful selection of the sensors to mount onboard these platforms. The work in [42] analyzes the suitability of Laser imaging Detection and Ranging sensors (LIDARs) along with standard and event cameras for ornithopters, concluding that event-based vision provides a promising solution to the majority of these perception challenges. The use of event-based vision for UAVs perception has increased in problems such as motion segmentation [189], surveillance [60], visual servoing [57], robot localization [105], rotor failure-recovery [121], and onboard computational load management [150], among others.

Several event camera datasets have been presented to explore the advantages of these sensors onboard multirotors. The dataset presented in [188], includes measurements from several IMUs, cameras (e.g., traditional and event), and a LiDAR.

Dataset	**Platform**	**Event camera**	**Data type**	**Scenario**
AU-AIR [180]	Multirotor	No	Real	Outdoors
V4RL Place Recognition [181]	Multirotor	No	Synt. & Real	Outdoors
Mid Air [182]	Multirotor	No	Synthetic	Outdoors
Zurich Urban[183]	MAV	No	Real	Outdoors
EuRoC [184]	MAV	No	Real	Indoors
Blackbird [185]	MAV	No	Synt. & Real	Indoors
UPenn Fast Flight [186]	Multirotor	No	Real	Outdoors
Maldonado et al.[39]	Ornithopter	No	Real	Indoors
Lopez et al.[187]	Ornithopter	No	Real	Outdoors
Multivehicle Stereo Event [188]	Multirotor	Yes	Real	Ind. & outdoors
Mitrokhin et al.[84]	Multirotor	Yes	Real	Indoors
Rodriguez et al.[60]	Multirotor	Yes	Real	Outdoors
UZH-FPV Drone Racing [152]	Multirotor	Yes	Real	Ind. & outdoors
ROSS-LAN	**Bio-inspired trajectory**	**Yes**	**Synthetic**	**Indoors**
GRIFFIN Perception	**Ornithopter gliding and flapping**	**Yes**	**Real**	**Indoors & Outdoors**

Table 6.1: Summary of main reported datasets for aerial robot perception.

It includes recordings onboard different vehicles such as a multirotor, a car, and a motorcycle. The works in [84] and [60] provide sequences recorded onboard quadrotors used to evaluate event-based methods for moving object detection and tracking. The dataset for autonomous drone racing in [152] includes measurements from conventional stereo cameras, event cameras, IMU, and the ground truth pose. However, none of the aforementioned works have explored the use of event-based vision onboard flapping-wing robots.

6.3 ROSS-LAN: RObotic Sensing Simulation scheme for bioinspired robotic bird LANding

This Section proposes a software tool to simulate bioinspired trajectories for landing and perching while emulating the measurements from different perception sensors. To the best of our knowledge, **it is the first simulation tool developed to reduce the gap between bioinspired aerial platforms and robot perception**. Besides, a dataset with synthetic sensor information collected in different landing trajectories has been publicly released [1] to pave the way toward the development of novel perception sensors for aerial robots.

The *RObotic Sensing Simulation scheme for bioinspired robotic bird LANding*, better known as *ROSS-LAN* includes two main simulation modules. First, a bioinspired trajectory generator to resemble the movements performed by birds during landing and perching. The simulation tool adopts *tau-theory* for trajectory generation, which describes the principle used by animals and humans to guide their intended motion to make contact with an object or surface. The theory has been used in aerial robotics to guide and control multirotor platforms for docking and landing [133] [190]. The second module of *ROSS-LAN* emulates sensors widely used for robotic perception. The sensor selection considers the information necessary for object identification, guidance, localization, and obstacle avoidance. Thus, sensors are selected from a robotic perception perspective and the simulator includes sensors typically used in

[1]`https://grvc.us.es/bioinspired-landing-trajectory-sensor-dataset/`

aerial robotics such as LiDARs, IMUs, altimeters, and frame-based cameras. The simulation tool also includes event cameras to deal with the fast-motion, motion blur effect, and strong illumination changes arising during the flight of ornithopters. Fusing event information together with classical perception data (e.g., frames, point clouds, and IMU measurements) have increased the robustness of perception algorithms for feature tracking [90], Simultaneous Localization And Mapping (SLAM) [105], and obstacle avoidance [148]. The most relevant aspects of the *tau-trajectory* and the architecture of the simulation tool are detailed below.

6.3.1 Tau-theory planning

ROSS-LAN relies on *tau-theory* [134] to approximate bio-inspired landing and perching maneuvers. This theory proposes that humans and animals use simple strategies together with the time variable τ to guide the majority of their intended movements. Assuming constant speed, *Tau* (τ) regards the time an observer would take to make contact with an object or surface. The value of τ corresponds to a first-order approximation of the Time-To-Contact (TTC) for a given gap χ:

$$\tau(t) = \frac{\chi(t)}{\dot{\chi}(t)}, \tag{6.1}$$

where $\chi(t)$ is the gap, and $\dot{\chi}(t)$ represents its closure rate at time-step t. By convention, χ is defined negative and its initial closure rate $\dot{\chi}(0)$, positive.

In [134], the author states that birds tend to maintain the rate of the gap closure constant to control their flight deceleration. This behavior defines the constant *tau-dot* strategy. Additionally, the research showed that zero velocity at contact is achieved by keeping $\dot{\tau}$ constant and within the range $[0, 0.5)$. The work in [133] proposes a variation of the *tau-dot* strategy to guide a breaking maneuver using $\hat{\tau}(t) = k_\tau t + \tau(0)$. The approach proposes a practical approximation to *tau-dot* avoiding the estimation of $\dot{\tau}$. It should be noted that *tau-theory* postulates that animals do not require cognitive processing for TTC as it is available at the neural circuit level [134]. Hence, $\chi(t)$ and $\dot{\chi}(t)$ can be computed as follows:

$$\chi(t) = \chi(0)\Big(1 + k_\tau t \frac{\dot{\chi}(0)}{\chi(0)}\Big)^{\frac{1}{k_\tau}}, \tag{6.2}$$

$$\begin{aligned} \dot{\chi}(t) &= \chi(0)\Big(1 + k_\tau t \frac{\dot{\chi}(0)}{\chi(0)}\Big)^{(\frac{1-k_\tau}{k_\tau})}, \\ \ddot{\chi}(t) &= \frac{\dot{\chi}(0)^2}{\chi(0)}(1-k_\tau)\Big(1 + k_\tau t \frac{\dot{\chi}(0)}{\chi(0)}\Big)^{(\frac{1-2k}{k_\tau})}, \end{aligned} \tag{6.3}$$

where $\chi(0)$ is the initial value of the gap, and defining $0 < k_\tau \leq 0.5$ guarantees χ and $\dot{\chi}$ reaching zero at the same finite time $T = \frac{-\tau(0)}{k_\tau}$. An additional analysis depicts that defining k_τ within the range $(0.5, 1)$ leads to a collision as the gap closes with $\dot{\chi}(0) \neq 0$. Figure 6.1 shows some examples of the gap closure by varying k_τ between 0.2 and 1.0. In the Figure, the main gap (e.g., the altitude of the robot) is positive, different from the original convention being consistent with a landing trajectory.

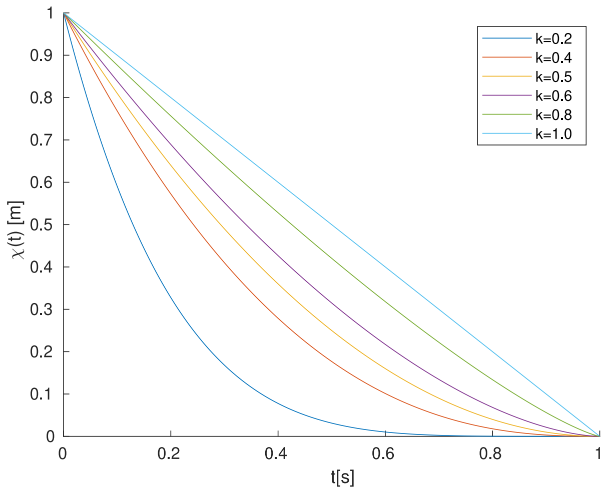


Figure 6.1: Different trajectories χ for $0 < k_\tau \leq 1$ closing the gap at $t = 0$. For $k_\tau = 1$ (light blue curve) the gap closes with $\dot{\chi}$ constant leading to a collision.

Furthermore, the *tau-coupling* strategy allows the closing of multiple gaps. This is particularly useful for closing the position and orientation gaps of the robot. *Tau-coupling* consists of coupling additional gaps ϑ to the main gap χ. It is computed as $\tau^{\chi} = \kappa_\tau \tau^{\vartheta}$, where κ_τ is defined as the closure constant ratio between both gaps. The coupled gap ϑ, its closure rate $\dot{\vartheta}$, and acceleration $\ddot{\vartheta}$ at time t are obtained by:

$$\begin{aligned} \vartheta(t) &= c\chi(t)^{\frac{1}{\kappa_\tau}-1}, \\ \dot{\vartheta}(t) &= c\frac{1}{\kappa_\tau}\dot{\chi}(t)\chi(t)^{\frac{1}{\kappa_\tau}-1}, \\ \ddot{\vartheta}(t) &= c\frac{1}{\kappa_\tau}\left(\left(\frac{1}{k}-1\right)\dot{\chi}(t)^2+\chi(t)+\ddot{\chi}(t)\right)\chi(t)^{\frac{1}{\kappa_\tau}-2}, \end{aligned} \tag{6.4}$$

where $c = \frac{\vartheta(0)}{\chi(0)^{(1/\kappa_\tau)}}$.

To compute the time trajectories of the robot, the proposed simulator integrates *tau-constant* and *tau-coupling* strategies. The *tau-trajectory* generation is performed by computing the trajectory that closes the main gap, and the set of coupled gaps for the robot position and orientation. The main gap is the z-axis (i.e., the robot's altitude), and the coupled gaps correspond to the coordinates in x and y, and the roll θ, pitch ϕ, and yaw ψ angles. The notation uses a superscript for the gap, and a subscript for the time. For instance, the main gap (z-axis) and a couple gap on the yaw ψ angle at time t are denoted by χ_t^z and ϑ_t^{ψ} respectively. The trajectory S_t at time t includes the gap values, closure velocities, and accelerations with respect to a reference frame $\mathcal{F}$ aligned with the goal pose. The pose transformation from $\mathcal{F}$ to any other reference frame $\mathcal{W}$ can be directly defined in a straightforward manner on the simulator.

6.3.2 Architecture of the simulator

The proposed multi-sensor perception simulator describes an architecture based on the Robot Operating System (ROS). It integrates the well-known *Gazebo* simulator and the *Unreal Engine 4* (UE4) graphic engine. Gazebo simulates traditional perception sensors for robotics such as altimeters, proximity sensors, LiDARs, and frame-based cameras. Conversely, UE4 simulates a photo-realistic version of the scenario to capture images of the scene at a very high frame rate. The photo-realistic images are used to produce simulated event data using ESIM [176]. Besides, the simulator includes the bioinspired trajectory generator based on *tau-theory*, which has been coded in C++ and integrated as a ROS package. The sensor measurements, the robot pose, and trajectory estimations are published in ROS topics.

The block diagram of ROSS-LAN is shown in Figure 6.2. The simulator requires a set of parameters for the simulation. First, the parameters to define the trajectory S_t (see Eqs. 6.2, 6.3, and 6.4). Second, the configuration file for the scene, which defines the poses of all objects in the scene including the initial pose of the robot. Third, the sensor configuration files that define the extrinsic calibration of the sensors with respect to the robot, and the intrinsic calibration data such as the camera matrix for the frame-based and event cameras. The latter are provided in *yaml* format. The current version of the simulator includes six types of perception sensors: (i) event cameras, (ii) LiDARS (Velodyne HDL-32), (iii) frame-based cameras, (iv) lasers, (v) altimeters, and (vi) IMUs. However, the modularity of the simulator allows easy integration of additional sensors such as stereo cameras and depth sensors. At each experiment, the output of the sensing data and robot pose are saved in a rosbag file.

The proposed simulation tool is divided into two main modules: the *Tau Trajectory Generator*, and the *Perception Sensor Simulator*. The first computes trajectory S_t following the approach described in Section 6.3.1. The module provides the pose $(\vartheta_t^x, \vartheta_t^y, \chi_t^z, \vartheta_t^\theta, \vartheta_t^\phi, \vartheta_t^\psi)$, velocities $(\dot{\vartheta}_t^x, \dot{\vartheta}_t^y, \dot{\chi}_t^z, \dot{\vartheta}_t^\theta, \dot{\vartheta}_t^\phi, \dot{\vartheta}_t^\psi)$, and accelerations $(\ddot{\vartheta}_t^x, \ddot{\vartheta}_t^y, \ddot{\chi}_t^z, \ddot{\vartheta}_t^\theta, \ddot{\vartheta}_t^\phi, \ddot{\vartheta}_t^\psi)$ for each time t. This module receives as input the initial pose of the robot in the target reference frame $\mathcal{F}$ along with the parameters k^z,κ^x, κ^y, κ^θ, κ^ϕ, and κ^ψ for each gap.

Moreover, the *Perception Sensors Simulator* module includes the simulation model of each of the aforementioned perception sensors. The simulator uses the trajectory S_t to modify the reference frame of each sensor for the entire landing trajectory. This module inputs the extrinsic and intrinsic calibration files of the different sensors.

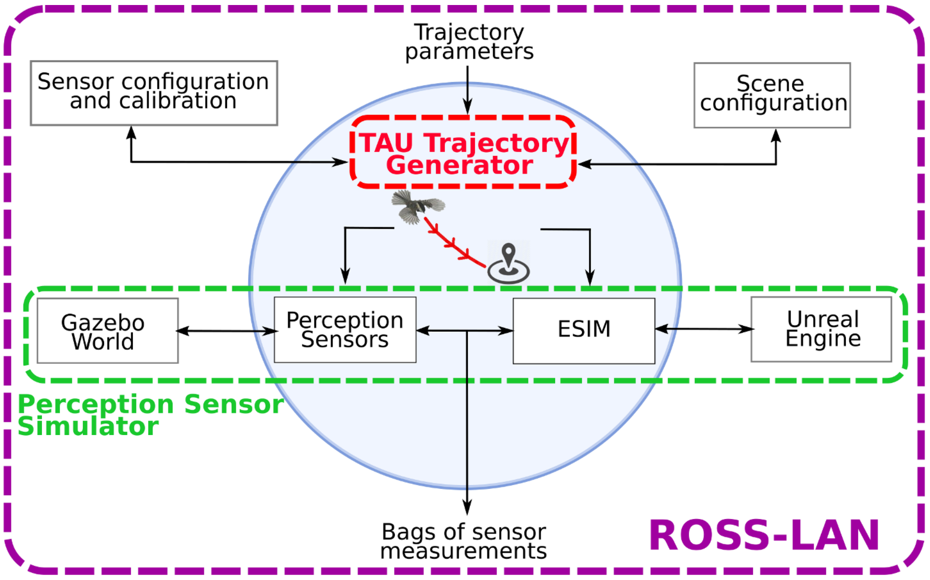


Figure 6.2: Block diagram of ROSS-LAN.

Besides, it integrates the *ESIM* event camera simulator. ROSS-LAN uses a modified version of *ESIM* that inputs S_t from the *Tau Trajectory Generator* to set the event camera pose during the simulation.

6.3.3 Datasets

A set of simulation datasets have been released for further research in bioinspired flight perception. The datasets simulate the sensor measurements obtained by performing bioinspired landing trajectories. Each dataset corresponds to a different simulation where the set of sensors move given the trajectory computed by the *Tau Trajectory Generator*. The simulated trajectories were computed with a final approach angle of $\pi/6$. The trajectories were sampled in time-steps of $\Delta t = 0.1\,\text{s}$ to $t = T$. The value of T is computed as described in Section 6.3.1. Figure 6.3 depicts some trajectories obtained from the *Tau Trajectory Generator* for different simulated landing maneuvers. The robot poses are shown as vectors where the tail represents the robot position (i.e., x, y, z), while its magnitude and direction correspond to the velocity components. The trajectories are normalized referring them to the same scale. Further, the values of κ^x and κ^y varied as depicted in the Figure, and k^z was set to 0.5 to provide smooth descending without colliding with the ground.

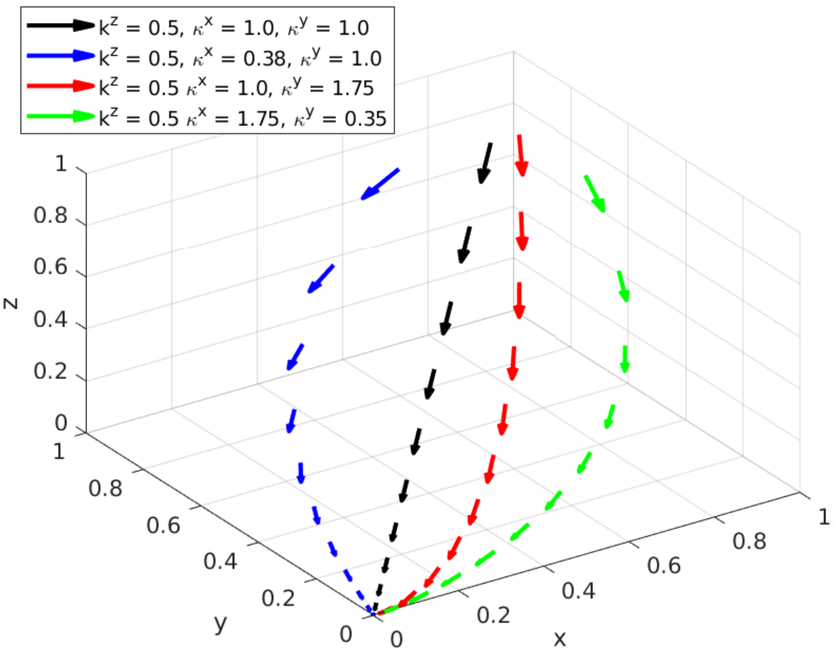


Figure 6.3: A set of normalized trajectories describing bioinspired simulated trajectories. The magnitude and direction of the vectors describe the robot's velocity, while the tail of each vector corresponds to the robot's position.

Figure 6.4 shows the two simulation scenarios used to collect the samples of the dataset. The scenes are inspired by industrial environments such as warehouses and factories where aerial robots perform tasks such as payload delivery, surveillance, intrusion monitoring, and remote sensing. Each object in the scenario was designed in 3D computer graphics software and imported into the scene. The simulation environments and their objects were designed following the approach in [63]. The distribution of the objects in the scenario is defined by a configuration file which is used by Gazebo and UE4 to keep the same scene configuration (i.e., position, orientation, and scale) for both simulators.

The proposed simulation tool has been designed to emulate all sensor measurements produced during the entire landing trajectory. However, generating all synthetic

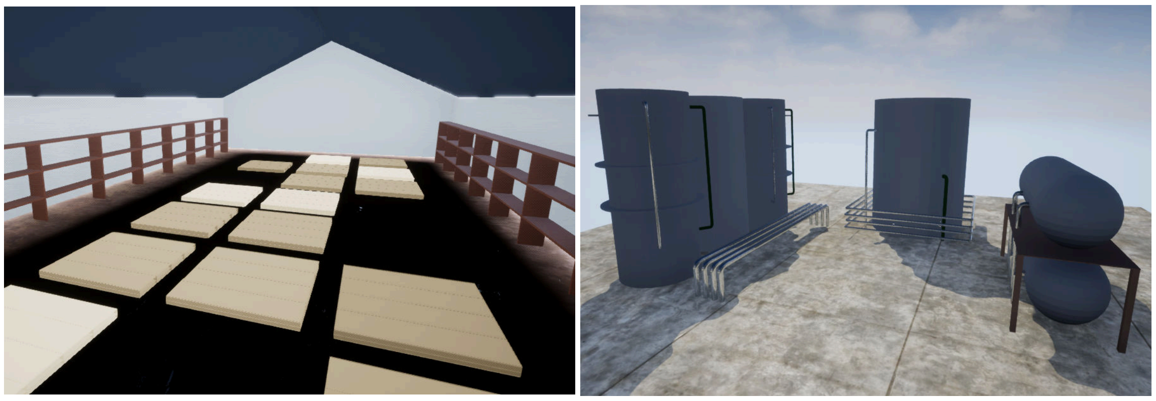


Figure 6.4: Perspective views of the simulation scenarios used to record the dataset. (a) A warehouse and (b) an oil refinery .

measurements in real time is computationally demanding. This is a common issue in the majority of the robotic simulation tools with complex sensors and robot platforms [174] [173] [171]. For instance, simulating event data requires rendering several frames per second and predicting their optical flow to obtain asynchronous simulated events [176]. To deal with this limitation, the datasets were recorded at a low rate to estimate all sensing measurements simultaneously. Thus, the simulation duration D_s of each dataset is longer than the trajectory duration D_t. The dataset rate can be adjusted by setting the factor r while running the bagfile to D_s/D_t.

The bioinspired landing trajectory dataset contains six simulations with different landing trajectories. Each dataset includes the measurements of a monocular frame-based camera, an event camera, a sonar, an IMU, and a LiDAR. The pose ground truth of robot and sensor poses were recorded as well for each dataset, see Figure 6.5. The modular architecture of the simulation tool allows easy integration of additional sensors such as the sensors included in *gazebo_ros_package* (e.g., kinect, lasers, and depth cameras). Each of the datasets includes the following files:

- The full dataset in a rosbag file.
- A README file with the instructions to execute the bag.
- The bioinspired trajectory computed with *tau-theory*.

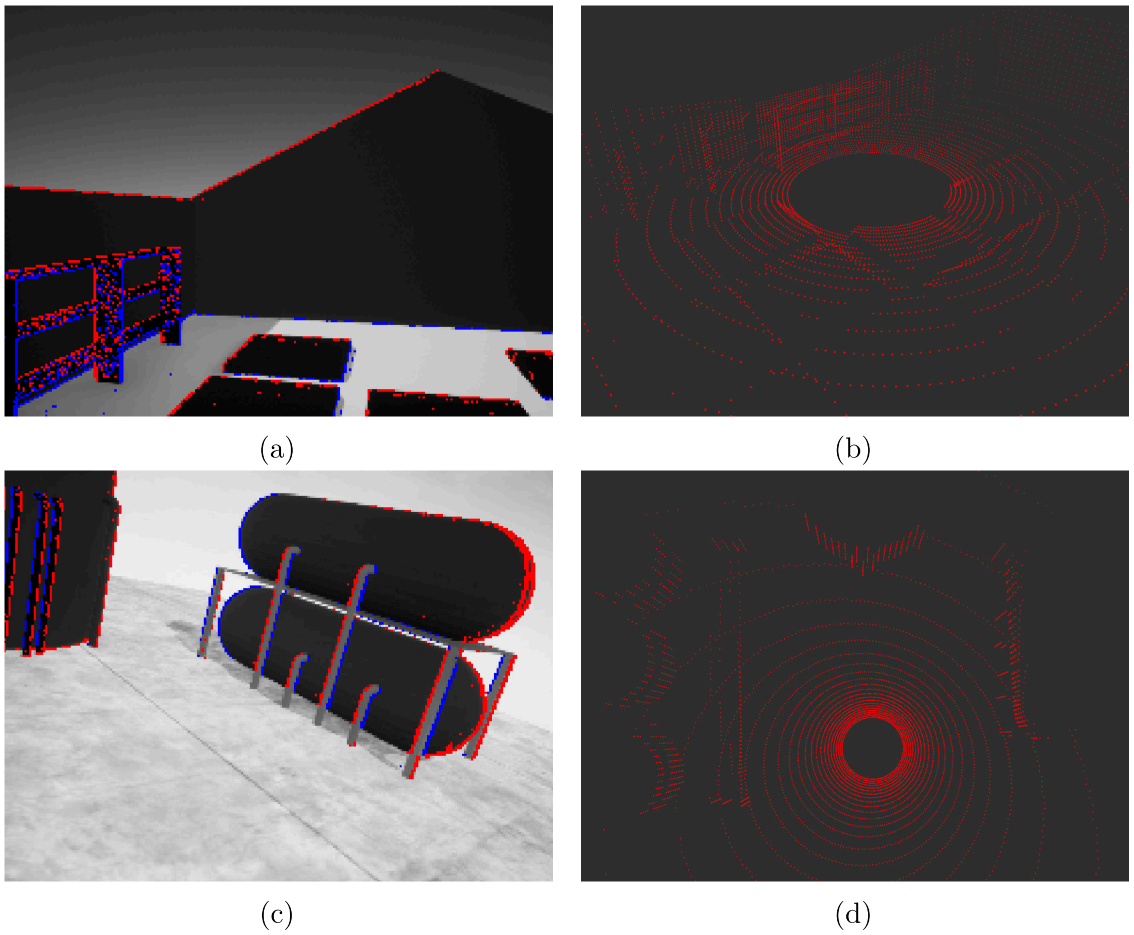


Figure 6.5: A visual representation of some of the sensing measurements collected in the datasets. (a,c) Grayscale frames along with the simulated events accumulated in time windows of 10 ms. (b,d) The point clouds from the LiDAR .

- A text file with the simulated events following the format $(timestamp, x, y, polarity)$.
- The sensor models including the extrinsic and intrinsic calibration.

6.4 The GRIFFIN perception dataset

This Section describes the development and structure of the *The GRIFFIN Perception Dataset*. The dataset aims at paving the way toward the development of perception algorithms for flapping-wing robots. To the best of our knowledge, **it is the first**

perception dataset recorded onboard an ornithopter robot. The dataset includes sensing measurements from a frame-based camera, an event camera, and two IMUS. Besides, it includes the pose ground truth of the robot obtained from a Total Station and a motion capture system. The perception sensors were carefully mounted onboard the *Eye-Bird* robot, an ornithopter designed at the GRVC Robotics Laboratory. The dataset was collected in different scenarios including indoor and outdoor environments and it has been publicly released [2].

6.4.1 The *Eye-Bird* ornithopter

The ornithopter used for the dataset collection is the *Eye-Bird*, a modified version of *E-Flap* which has been customized to carry sensors and electronics for perception research. Ornithopters report a limited payload and weight distribution. Increasing the payload of an ornithopter requires additional lift and thrust which can be obtained by increasing the flapping frequency. This induces higher stresses over the structural and mechanical parts. The empty weight of 450 g and 1.5 m wingspan of the robot is optimized for maximum payload capacity with several attachment locations using nuts, bolts, or cable ties over the robot body. The additional electronics, sensors, and batteries represent a payload of 250 g for the data collection flights. The sensor location along the robot body strongly affects the Center of Gravity (CoG) and inertia, and therefore the stability and maneuverability of the platform. The 39 cm length of the tail brings the neutral point of the platform back to 25 cm from the head. This defines the limit at which the CoG can be moved back to maintain the flapping-wing robot stable.

The ornithopter design and hardware allocation is shown in Figure 6.6. Allocating the camera and batteries at the front of the platform moves forward the CoG for better stability. The onboard computer is placed at the safest point, below the wings, and at the center of the body tube. The robot design does not include a body fuselage to cover the electronics to simplify hardware maintenance and add versatility.

[2]`http:\grvc.us.es/eye-bird-dataset`

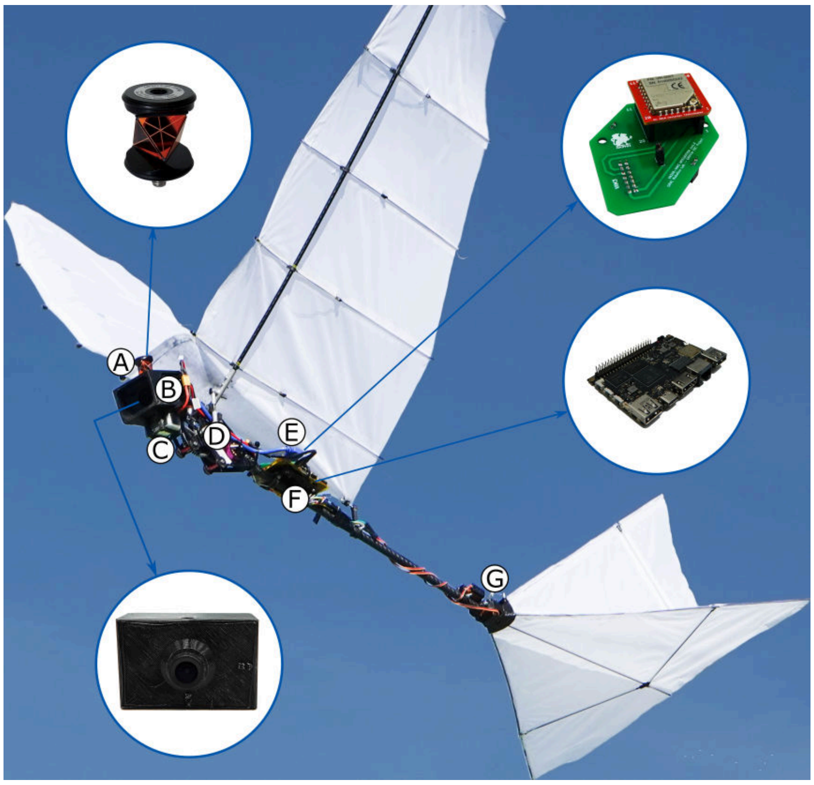


Figure 6.6: Distribution of the perception hardware on the GRIFFIN *Eye-Bird* in an experimental scenario: (A) Leica GRZ101 MiniPrism, (B) DAVIS346 camera, (C) battery, (D) the flapping mechanism, (E) VectorNav VN-200 IMU, (F) Khadas VIM3 board, and (G) tail servos.

The wings and tail are built with a nylon fabric attached to a lightweight carbon fiber structure composed of rods and tubes. The shape of the wing offers low-speed flapping flight aerodynamics, and low descent gliding capabilities making the robot suitable for safe landing even in emergency cases, different from multirotors. The wing weights 82 g with a total aerodynamic surface of $0.44\,\mathrm{m}^2$.

The tail consists of a triangular horizontal stabilizer of $0.1\,\mathrm{m}^2$ and a triangular ruder of half of the tail size. This configuration provides longitudinal and directional stability and attitude control. The tail is actuated by two concatenated servo motors. The first acts over the tail, and the second, over the rudder direction. The forward speed influences the tail pitch as it controls the aircraft's nose pitch. The roll stability of the dihedral wing and the directional stability provided by the rudder mitigate the lack of roll actuation. The tail is trimmed based on the weight distribution using the maximum glide ratio criteria. It is worth mentioning that the design and construction of the *Eye-Bird* robot is not part of the contributions of this Ph.D. Thesis. This section only aims to describe the most relevant technical aspects of the platform.

6.4.2 Sensors

The perception sensors are selected considering their relevance for flapping-wing perception and the *Eye-Bird* weight and size restrictions. The sensing measurements are acquired and recorded using the low-weight Khadas VIM3 board. It has a 6-core ARM Central Processing Unit (CPU), a 16GB eMMC storage unit, and three USB-3.0 interfaces. The Khadas uses Ubuntu 18.04 with ROS Melodic, and records the collected measurements in rosbag files.

Reducing the weight of the hardware components plays a key role in the dataset collection. Table 6.2 shows the main specifications of the components on board the ornithopter. Some sensors have been modified for their integration into the flapping-wing platform. The final weight after adding the sensors, electronics, batteries, and wires is lower than 250 g, which is close to the maximum payload of the platform with the desired maneuverability.

The main sensors mounted on the ornithopter are the VectorNav VN-200 and the iniVation DAVIS346. The first is a navigation device including several sensors such as a high-end IMU, a barometer, a magnetometer, and an external GPS. It includes an Extended Kalman Filter delivering coupled position, velocity, and attitude. The VectorNav is installed as close as possible to the center of gravity of the robot. The measurements from the sensor are available through the Universal Asynchronous Receiver-Transmitter (UART) protocol and published using a ROS custom package[3] specially designed for the sensor. The DAVIS346 includes three different sensors: (i) a 346x260 Active Pixel Sensor (APS) providing grayscale frames at 40 Hz, (ii) a

[3] `https://github.com/grvcPerception/vn_ros_integration`

Sensor	Characteristics	Weight
DAVIS346	DVS: 346x260 up to 12 MHz APS: 346x260@40 Hz FoV: 68 vert., 83 horiz. IMU: MPU 9250@1 kHz	With custom case & lens: 57 g
VN-200	IMU readings @80 Hz Gyroscope, accelerometer, and magnetometer available GPS and barometer disabled	With adaptation board: 21 g
Leica MS50 TotalStation	Cent.-level tracking@10 Hz GRZ101 prism on the bird	Prism: 30 g
OptiTrack	Cent.-level tracking@100 Hz 5x850 nm LEDs on the bird	LEDs and cables: 5 g

Table 6.2: Specifications of the onboard sensors and ground truth instruments.

Dynamic Vision Sensor (DVS) providing events with a temporal resolution of 1 µs with a maximum throughput of 12 MHz, and (iii) an IMU running at 1 kHz.

The original lens of the DAVIS346 weighs $\sim$ 100 g, which is heavy for the limited payload of the robot. The lens is changed by two different lightweight lenses, one for outdoors and one for indoor experiments. The indoor lens includes an IR cut-off filter to cope with the motion capture system IR emitters. Each lens weights 5 g, with a focal distance of 3.6 mm, a horizontal Field of View (FoV) of 83°, and vertical 68°. Additionally, the case of the camera is replaced with a PLA (Polylactic Acid) 3D-printed version. The weight of the modified event camera is 52 g, less than a third of its original weight (i.e., $\sim$ 170 g). Additionally, the DAVIS346 is mounted at the front of the ornithopter with an angle of pitch of 30° using a lightweight protective case. The case is 3D printed with flexible Thermoplastic PolyUrethane (TPU) filament, that acts as an impact absorber in case of a frontal collision. The DAVIS346 is mounted upside down (see Figure 6.7) to facilitate its installation on the ornithopter.

The ground truth position of the robot is recorded using an MS50 TotalStation (in outdoor experiments) and an OptiTrack (in indoors). The TotalStation accurately provides range and bearing measurements of a prism reflecting the target attached to the robot. The GRZ101 360° MiniPrism is installed at the front of the ornithopter

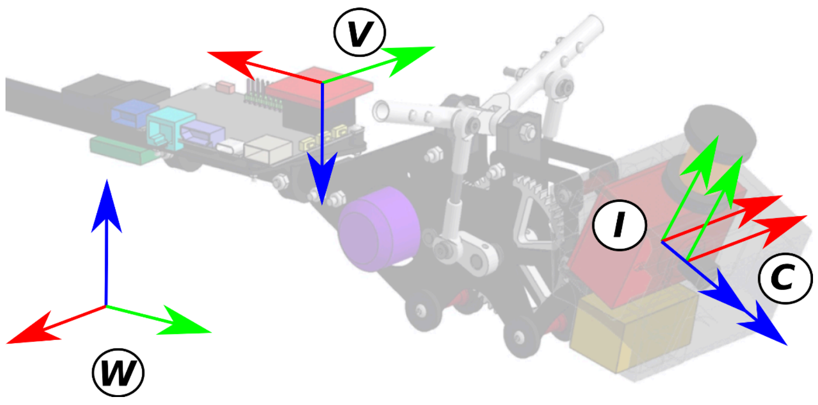


Figure 6.7: The GRIFFIN *Eye-Bird* ornithopter with the reference frames of each sensor: (V) VectorNav, (C) DAVIS346 APS and DVS, (I) DAVIS346 IMU, and (W) world reference frame either from the TotalStation or the OptiTrack. The DAVIS346 was mounted upside down, affecting frames (C) and (I). The TotalStation prism is located at coordinate (0, 37.2, 0) mm at frame (I).

above the DAVIS346. This location is preferred to avoid possible target occlusions due to the flapping motion of the wings. Besides, the Total station is located as far as possible from the flight zone to increase the Field of View of the laser. The experiments conducted with the Totalstation describe outdoor flights where the robot faces the TotalStation to minimize possible occlusions of the prism during the trajectory. Conversely, the Motion capture system is used for indoor flights. An OptiTrack system with 28 cameras is used in indoor experiments providing millimeter accuracy ground truth position and orientation at 100 Hz. To track the ornithopter with the motion capture system five infrared (850 nm) Light Emitting Diodes (LEDs) are installed at the main frame of the robot. The reference frame of the rigid body representing the robot is located at the same position and orientation as the TotalStation prism.

The sensor measurements are recorded in rosbag files. The format of the sensing measurements is given by the timestamped standard ROS message libraries. The events from the DAVIS346 DVS follow the format in [93]. The frames of the APS include the pixel raw data, image resolution, and encoding. Both DAVIS346 and VectorNav use the IMU message of ROS. The ground truth poses from both TotalStation and

Optitrack use the timestamped poses format of ROS. Each sensing measurement is referenced to its sensor frame. The frames of each sensor are shown in Figure 6.7.

Finally, a calibration file with the intrinsic and extrinsic camera calibrations is provided in *yaml* format. Additional datasets are provided for camera calibration, in these sequences the camera moves in different directions in front of an AprilTag [191] board. The calibration data were obtained using the Kalibr toolbox [192]. The calibration samples are provided along with the perception dataset in case the user prefers to estimate the camera calibration with a different tool. The intrinsic calibration of the camera was recomputed for each scenario to prevent possible biases due to changes in the camera lens configuration. However, the extrinsic calibration was computed only once as the sensor configuration along the robot body was the same for all experiments. Besides, an additional dataset with IMU measurements to extract the internal bias of the sensors is provided. Finally, both intrinsic and extrinsic calibration files are validated using a state-of-art Visual Inertial Odometry (VIO) method in Section 6.4.5.

6.4.3 Ornithopter v.s. quadrotor flight

An initial analysis of the flapping effect on the onboard perception was performed by comparing the data collected on *Eye-Bird* and on a quadrotor. Several flights were performed to obtain similar flight trajectories between both platforms. The quadrotor is a customized platform with a DJI FlameWheel F450 frame, and a PixRacer autopilot (see Section 3.7.2 for further details). The DAVIS346 and the Khadas VIM3 board were mounted in a similar configuration to the ornithopter. Figure 6.8 shows both platforms flying together in an experiment.

The experimental results depict that the flapping-wing robot suffers from high stronger vibrations compared to the quadrotor. In general, the DAVIS346 IMU reported 3 times greater accelerations in the ornithopter. These vibrations increased the number of triggered events. On average *Eye-Bird* reported 6 times more events than the quadrotor. Figure 6.9-a,b shows the events triggered by each platform, this frame representation was obtained by accumulating events in time windows of 25 ms

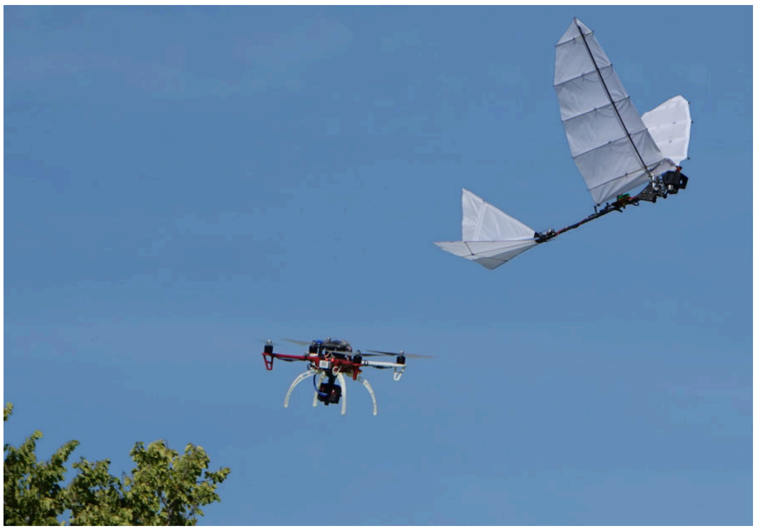
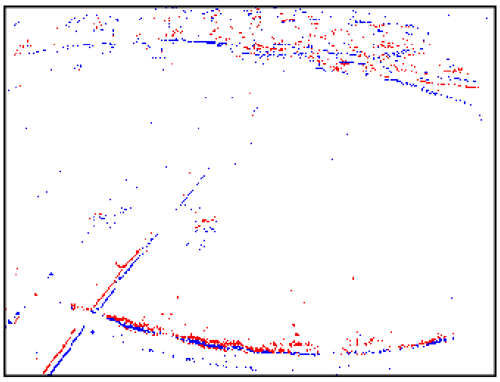


Figure 6.8: The quadrotor platform and the *Eye-Bird* flying together in an experiment.

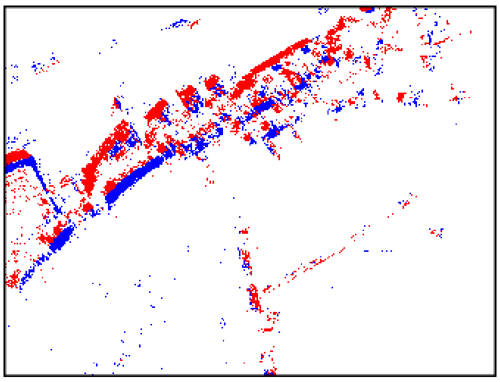
and warping them in the image. Figure 6.9-b depicts that the ornithopter triggers considerably more events than the multirotor. Moreover, Figure 6.9-c depicts the number of events accumulated every millisecond while both platforms described a similar trajectory. The results confirm the intuition that the flapping-wing robot generated more events during flight. Additionally, the sequences recorded with the ornithopter reported underexposed and overexposed grayscale frames due to the strong variation in the robot pitch angle produced by flapping strokes. Thus, adding lighting robustness requirements to vision-based algorithms for flapping-wing robots.

6.4.4 The GRIFFIN perception datasets

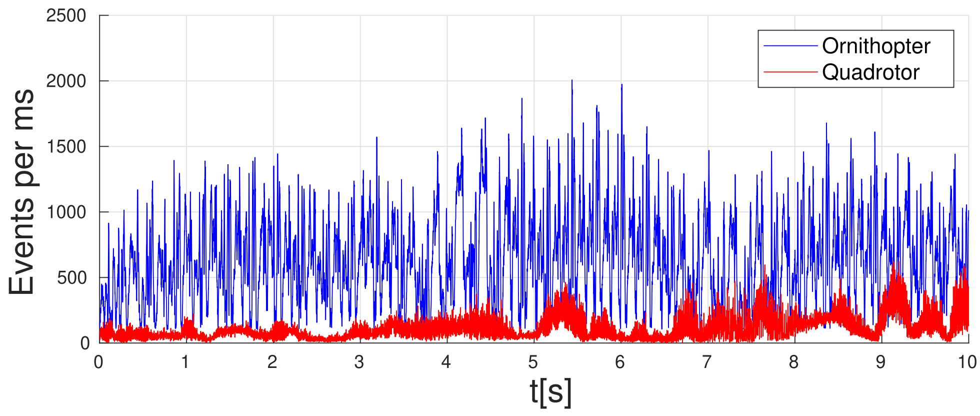
The dataset recording consisted of acquiring and saving the sensing measurements of different flights. The samples from the onboard sensors were recorded in the onboard computer (i.e., the Khadas VIM3), while the ground truth poses were recorded on an external computer. To guarantee time consistency between the data collected, the clock of the onboard computer and the external processing board were synchronized following the Network Time Protocol (NTP) [193].



(a)



(b)



(c)

Figure 6.9: The number of events per millisecond generated in *Eye-Bird* and the quadrotor while describing a similar trajectory: (a-b) Frames of accumulated events at 40Hz on the quadrotor (a) and on the ornithopter (b); and (c) the number of events accumulated every millisecond in both platforms.

In each data collection experiment, the recording of the sensing data of the onboard and external sensors was initialized before taking off. The ornithopter was initially oriented toward a calibration pattern to add visual features at initialization. Afterward, the ornithopter was launched by the pilot towards the flight arena. The pilot controlled the ornithopter to perform different trajectories. The event data generation describes the flight stages as it is shown in Figure 6.10-top (i.e., *Hills Base 3* flight). The green

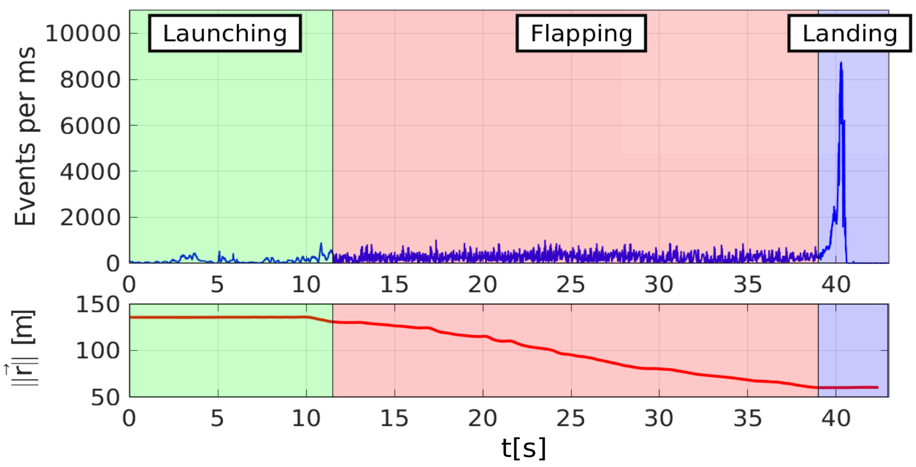


Figure 6.10: Event generation describing the flight stages of the ornithopter – *launching*, *flapping*, and *landing*. Top) the events triggered per millisecond, down) and $\|\vec{r}\|$, the norm of the position vector in dataset *Hills Base 3*.

area corresponds to the *launching* stage where the ornithopter was almost static in the pilot's hands. The red zone describes the *flapping* stage where events were mainly triggered by the motion of the robot and the flapping strokes. The blue area represents the landing stage that reports a peak of events triggered due to the impact on the ground at landing. Additionally, Figure 6.10-bottom depicts $\|\vec{r}\|$, the norm of the position vector in world coordinates tracked by the TotalStation in this experiment. The rest of the dataset sequences have different durations depending on the flight trajectory performed by the ornithopter in each experiment. The GRIFFIN perception dataset includes three types of sequences:

- *Base*: A set of datasets where the ornithopter performed agile trajectories. The scenario did not include artificial markers.

- *ArUco*: In these datasets the ornithopter performed smooth trajectories by flying over ArUco markers located on the ground of the scene.

- *People*: A set of datasets that include visual information of people and objects in the scene. This dataset may be useful for training and testing object detection algorithms. In these datasets the ornithopter performed longer trajectories without having any altitude limitation. The ground truth pose information was not recorded due to the complexity of tracking the robot with the TotalStation under these conditions.

The datasets were recorded in three different scenarios. The *Soccer* scenario describes an outdoor area with a soccer field surrounded by objects of different sizes (e.g., threes, benches, and fences). The area of the scenario is 48 × 54 m. In this scenario, the ornithopter was launched from an elevated platform located at ∼ 74 m from the TotalStation. The *Hills* scene corresponds to an open space with irregular ground surfaces. It has a total area of 170 × 100 m enabling large flight without collision with obstacles. The distance between the launching spot and the TotalStation was ∼ 135 m during the dataset collection. The *Testbed* scenario is 15 × 21 × 8 m close space specially designed for testing ornithopters. It is equipped with an Optitrack motion capture system to record the pose of the robot. Figure 6.11 shows a picture

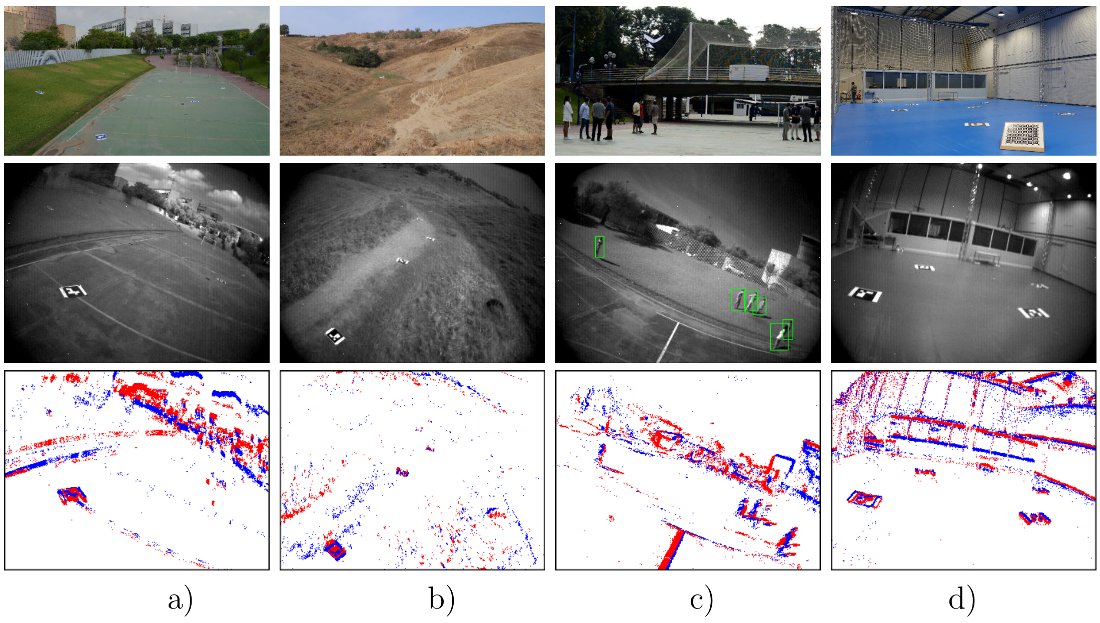


Figure 6.11: Some of the perception data collected in the dataset along with a picture of each scenario: (a) *Soccer*, (b) *Hills*, (c) *People* flight in the *Soccer* scenario, (d) *Testbed*.

of each scenario along with some of the visual data collected on them. The second row corresponds to grayscale frames from the APS, while the third column shows frame representations of the collected event data. A simplified version of the method inspired in [39] was used in this scenario to perform landing trajectories with the ornithopter. Additional aggressive maneuvers were performed by the pilot in this scenario to provide richer trajectories that exploit the ornithopter maneuverability.

Table 6.3 shows the list of the 21 recorded datasets; 9 of them were recorded in the *Testbed*, 7 were conducted in the *Soccer* area, and 5 datasets were collected in the *Hills* scenario. A total of 9 *Base* and 10 *ArUco* datasets were provided as they are particularly useful for developing vision algorithms for flapping-wing robots in structured (*ArUco*) and fully unstructured (*Base*) environments. In the *People* dataset manual annotations of people were provided to facilitate training people detection algorithms. Table 6.3 includes the main specifications of each dataset.

6.4.5 Validation

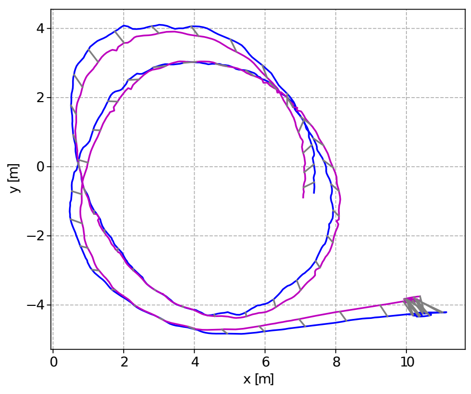
The dataset validation was performed by comparing the recorded ground truth with the output of a VIO algorithm in all collected sequences. This validation confirms the validity and usability of the dataset by assessing the provided extrinsic and intrinsic calibrations together with the ground truth samples. Further, the validation provides a baseline result for future comparison with other VIO algorithms. The method selected for validation was the well-known ROVIO [194], a robust method that estimates the robot trajectory without requiring exhaustive parameter tuning. The robot trajectory estimated with ROVIO against the ground truth in one of the datasets is shown in Figure 6.12. Besides, the absolute translation Root Mean Square Error (RMSE) between the estimated trajectory and the ground truth is provided in Table 6.3. For fairness, the execution of the VIO method was finalized before landing to avoid errors due to the lack of visual features in the camera Field of View. Moreover, the error provided by ROVIO in the *Hills ArUco2* dataset was mainly due to drift confirming the limitations of the exiting VIO methods for ornithopter robot flight.

Scenario	Dataset	t s	d m	$\|\bar{v}\|$ $\mathrm{m\,s^{-1}}$	$\|\bar{w}\|$ $\mathrm{rad\,s^{-1}}$	Max EPms event/ms	t_{RMSE} m	Ground Truth	Annotation
Testbed	Base1	27.99	96.12	3.16	0.29	3575	0.68	OptiTrack	No
	Base2	18.99	76.26	3.91	0.28	8171	0.59	OptiTrack	No
	Base3	16.99	57.61	3.14	0.25	4094	1.03	OptiTrack	No
	ArUco1	20.98	61.37	2.81	0.21	4295	0.41	OptiTrack	No
	ArUco2	24.99	94.10	3.42	0.27	5073	0.55	OptiTrack	No
	ArUco3	26.98	108.38	3.86	0.29	8726	0.46	OptiTrack	No
	ArUco4	4.80	12.17	2.57	0.14	8631	0.06	OptiTrack	No
	ArUco5	3.97	11.08	2.83	0.13	8616	0.13	OptiTrack	No
	ArUco6	3.98	11.98	2.99	0.12	8667	0.21	OptiTrack	No
Hills	Base1	39.21	107.76	2.77	–	8704	1.20	TotalStation	No
	Base2	39.90	110.21	2.80	–	8784	2.48	TotalStation	No
	Base3	42.91	98.75	2.52	–	8772	2.46	TotalStation	No
	ArUco1	31.80	97.95	2.97	–	8840	2.14	TotalStation	No
	ArUco2	27.21	114.93	4.31	–	6615	6.46	TotalStation	No
Soccer	Base1	24.89	54.65	2.22	–	8610	0.83	TotalStation	No
	Base2	20.01	43.41	2.16	–	8549	1.63	TotalStation	No
	Base3	27.90	54.42	1.92	–	7166	1.91	TotalStation	No
	ArUco1	15.31	51.49	3.47	–	7534	1.01	TotalStation	No
	ArUco2	53.02	19.50	2.79	–	8757	1.12	TotalStation	No
	People1	79.98	–	–	–	8737	–	None	Yes
	People2	88.04	–	–	–	8729	–	None	Yes

Table 6.3: Specifications of each dataset: t, flight time duration; d, total traversed distance; $\|\bar{v}\|$ and $\|\bar{w}\|$, mean inertial linear and angular velocities; the maximum number of events per millisecond (EPms) along each flight: t_{RMSE}, absolute translation root mean square error of the robot position estimated using ROVIO; and availability of annotated data. It is worth mentioning that $\|\bar{w}\|$ is not provided in *Hills* and *Soccer* scenarios as the TotalStation provided only position data.

6.5 Conclusions

The development of perception techniques for ornithopter robots faces several challenges. First, the lack of flapping-wing platforms with enough payload to carry several perception sensors together with the strict weight distribution of these platforms set a serious entry barrier for the development of perception algorithms for these robots. Additionally, the ornithopters' principle of operation poses additional implementation challenges hindering the experimentation and validation of perception systems for these platforms. Furthermore, the sudden motion and high vibrations of flapping-wing



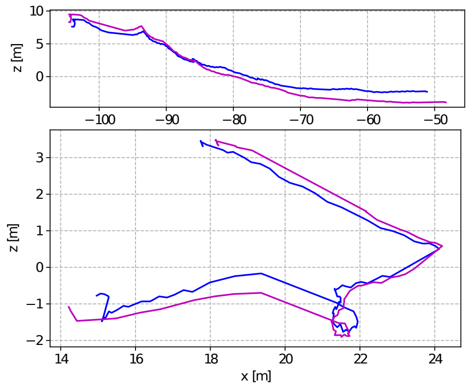


Figure 6.12: Robot trajectories estimated using ROVIO (in blue) against the ground truth (in magenta) in the flights *Testbed ArUco1* (Left), *Hills ArUco1* (Right-up), and *Soccer ArUco1* (Right-down).

robots produce motion blur and changes in lighting conditions which affect traditional vision-based perception algorithms.

In this Chapter, two different tools are introduced to provide valid solutions for some of the aforementioned issues. First, a simulation architecture to retrieve the sensing measurements generated during the execution of bioinspired landing and perching maneuvers. A dataset with sensor measurements collected by following different bioinspired trajectories is publicly available online. Each dataset provides the bioinspired trajectory followed during the simulation along with the sensing information from several perception sensors. Second, a perception dataset for large-scale flapping-wing robots was recorded onboard the *Eye-Bird* GRIFFIN ornithopter. It includes measurements from a frame-based camera, an event camera, and two IMUs together with ground truth data from a laser tracker (in outdoor scenarios) and a motion capture system (in indoor scenarios).

The work described in this Chapter intends to set the baseline to boost the development of flapping-wing perception algorithms. The simulation tool aims at providing a software solution to retrieve synthetic perception data for bioinspired perching and landing. Conversely, the dataset paves the way for the development of perception algorithms that consider the perception challenges that arise in the flight of large-scale ornithopters. Although the presented work sets an initial step toward the study of sensing data for flapping-wing robot perception, further efforts are needed to study the suitability of other sensors that may be particularly useful for ornithopter perception (e.g., depth cameras and pixel processing arrays, among others). Besides, the development of 3D dynamic models for ornithopters would enlarge the capabilities of current simulation tools such as the one proposed in this Chapter.

Chapter 7

Conclusions and Future Work

This Chapter is divided into two Sections. The first Section presents the conclusions regarding the research enclosed within this Ph.D. Thesis. The second Section proposes future work to extend some of the proposed approaches and design new event-based perception methods for aerial robotics.

7.1 Conclusions

This thesis confirms the use of event cameras as valid perception sensors for multirotors and ornithopters. In particular, this Ph.D. shows the advantages of using event-based vision for flapping-robot perception within the context of the ERC GRIFFIN project. The asynchronous nature of the event generation along with their microsecond resolution allowed the development of fast response perception algorithms (see Chapters 2, 3, 4, and 5). Several of these algorithms (e.g., see Sections 2.3.1, 2.3.2, 3.5, 4.7.1, and 5.3) were validated on aerial platforms performing agile and fast maneuvers for robot visual guidance and obstacle avoidance. Besides, the high dynamic range of event cameras permitted extending the capabilities of the pipelines proposed in Chapters 3, 4, and 5 to successfully perform in pitch-dark illumination conditions. Moreover, the event cameras' robustness to motion blur was studied in a preliminary study to this Ph.D. Thesis [42], that evidenced an important advantage of using event-based vision for flapping-wing robot perception compared to standard cameras.

The research enclosed within the development of this Ph.D. Thesis also describes some of the challenges of using event cameras for robot perception. First, the single-pixel event representation cannot be directly applied to many traditional perception methods relying on frames, which is a general challenge for event-based robot perception and computer vision. In this Ph.D. Thesis several algorithms for low-level perception (e.g., corner detection and tracking, clustering, optical flow estimation, and pixel undistortion, among others) were designed or adapted [74] [81]. Several of these methods are integrated as modules for the high-level event-based perception pipelines of Chapters 3, 4, and 5. Second, the fast response capabilities of the proposed algorithms required careful technical implementation to reduce as much as possible their processing rates. Third, further tools were proposed (see Section 5.3.2) and adopted [100] to reduce the computation load of processing additional events caused by the fast motion of the robots, the vibrations produced by external perturbations (e.g., flapping strokes), and the noisy events triggered under low-illumination conditions.

Moreover, the experimental results of this Ph.D. Thesis evidence the benefits of tuning the internal parameters of the methods proposed in Chapters 2, 4, and 5. These parameters were carefully adjusted to improve the methods' performance. However, the tuning process may require time and effort as adjusting the parameters of several algorithms becomes complex by increasing the number of variables to tune. This task also requires a base knowledge of the algorithm functioning to understand the relevance of each parameter on the method performance. The results in Chapter 5 show that using an automatic tuning tool for this task improves the performance of the proposed method. The results in Chapter 5 show that this task can be addressed by an automatic tuning tool that improves the performance of the proposed method while avoiding exhaustive manual adjustments.

The experimental validation obtained in this Ph.D. Thesis positively depicts the applicability of event-based vision to the visual servoing problem. Although, Event-Based Visual Servoing (EBVS) encloses similar challenges to Image-Based Visual Servoing (e.g., recovering after losing visual features in the camera Field of View), the remarkable features of event cameras may be used to improve the visual servoing performance. The works in [57], [115], and [58] report some EBVS schemes for robotics.

In general, all methods provide two additional advantages compared to traditional IBVS: (i) a faster computation of the visual features, and (ii) the adaptability of the proposed methods to perform under dark illumination conditions. The **Contribution 1** of this Ph.D. reports a method that provides high perception rates (i.e., $\sim 350\,\mathrm{Hz}$) by accurately tracking line references for visual servoing. This allows fast perception responses which are particularly useful in applications where aerial robots move at fast velocities. Moreover, the aforementioned works include experimental validation in scenarios with low illumination conditions depicting the EBVS capabilities in dark-pitch situations.

Additionally, we would like to provide some relevant considerations to perform experimental validations with large-scale flapping-wing robots. They are the result of several hours of experimentally validating some of the proposed perception algorithms on the *Eye-Bird* and the *E-Flap* robots, which were designed and developed at the GRVC Robot Laboratory within the context of the ERC GRIFFIN grant. First, integrating additional hardware onboard ornithopters is a critical task as each device adds weight and modifies the robot's Center of Gravity (CoG). This task requires particular attention as installing the hardware attachments and structures should affect as minimally as possible the flying capabilities of the robot. Second, running experiments with ornithopters requires additional safety protocols to avoid damaging the robot while landing. In the majority of the presented experiments, a set of mats and nets were deployed around the landing areas to prevent damage to the robot body. Installing these safety mechanisms is a time-demanding task especially by performing experiments in different scenarios (indoor and outdoor). Third, a successful validation requires teamwork and planning. Ornithopters are quite novel platforms compared to multirotors and fix-wing vehicles. The development of ornithopter platforms with fully autonomous capabilities is a topic of current research [41] [58] [59] [30]. However, several of the current platforms still require take-off and landing mechanisms, increasing the difficulty to perform experimental validations. For instance, sometimes an operator is needed to launch and recover the ornithopter, increasing the number of people involved in each experiment. Moreover, finding the right scenario is crucial. Flying almost any type of aerial platform requires a minimum set of conditions to perform a good flight. It is important to check and monitor the weather (e.g., humidity and wind velocity) and evaluate any potential risk for the people around the scenario and participating in the experimental validation. The previous considerations aim at technically contributing to the flapping-wing robot community by sharing some of our experience working with ornithopter robots.

7.2 Future work

The research entailed in this Ph.D. Thesis opens a wide field of further research. Next, the topics for future research most directly related to the presented work are summarized:

- **Evading dynamic and static obstacles with large-scale ornithopters**. Chapter 4 proposes a dynamic *sense-and-avoid* method for large-scale flapping-wing robots. It analyzes the spatial-temporal information of events to detect moving obstacles and estimate their direction of motion. However, the current method lacks a strategy to detect and evade static obstacles in the scene. Future work aims at implementing a vision-based algorithm to detect near-static objects and combining it with the proposed dynamic *sense-and-avoid* pipeline. The final goal is to provide an avoidance system to evade both types of obstacles with large-scale flapping-wing robots. The work in [41] proposes a stereo vision method to evade static obstacles with a small flapping-wing robot. It integrates a lightweight custom-made vision system to extract grayscale stereo-images and compute disparity maps with the static obstacles. A similar approach may be followed to detect near-static obstacles using either event-based or traditional computer vision. Using stereo cameras benefits dynamic and static obstacle detection as the objects' depth can be estimated without requiring prior information from their shape [103]. Another approach may focus on fusing measurements from a depth camera and an event vision sensor [104]. In this case, depth information is directly obtained without increasing the computational cost of estimating it. However, these approaches require additional hardware to mount on ornithopter robots, which typically report very limited payload capabilities.

- **A general intrusion monitoring approach**. Chapter 5 presents an event-based method to detect moving intruders on multirotors. The approach aims at performing Intrusion Monitoring (*IM*) while the platform hovers and maintains the desired pose. Under these conditions, events are mainly triggered by the intruder's movement and the motion of the platform. The latter is principally due to the rotor vibrations and the motions performed by the platform to keep its desired pose. The current method does not include a module to detect moving intruders while the robot performs medium and fast flight. This entails further challenges as the robot and the intruder may move at similar velocities. The method proposed in Section 5.3 assumes that intrudes move at a higher speed than the robot. Future work aims at extending the capabilities of the proposed algorithm to perform intrusion detection while the camera describes medium and fast motions. Some works have proposed solutions for this particular problem [84] [101] [189]. However, they include either optimization steps [84] [101] that hinder real-time detection, or Deep Learning (DL) solutions [189] that require specific hardware. The future *IM* approach focuses on detecting obstacles in real-time, as it is one of the main requirements for *IM*, while running in lightweight processors suitable for aerial platforms.

- **A wider perception dataset for ornithopter perception**. Section 6.4 of this Ph.D. Thesis presents a perception dataset collected on a large-scale ornithopter. The dataset provides useful information for event-based and frame-based monocular perception. However, it misses relevant sensing information for robot perception such as depth data, stereo images, and high-resolution RGB (Red, Green, Blue) frames. These data are particularly useful for implementing and validating perception algorithms for 3D reconstruction, mapping, Visual Odometry (VO), and object detection. For instance, ground truth depth information is required for evaluating the quality of 3D reconstruction methods. Besides, depth measurements and robot poses can be fused to compute ground truth optical flow [195]. Moreover, stereo-visual information is used in several works [196] [53] [197] to simplify and improve the performance of VO and 3D reconstruction methods. Additionally, RGB frames are preferred to extract additional information of the scene. For instance, some perception approaches use color information to simplify the detection of specific objects in the scene [198]. This future work aims at providing a wider dataset for robot perception on ornithopters that serves as a general benchmark for validating frame-based and event-based perception algorithms. The development of the dataset requires flapping-wing platforms with a higher payload to mount the missing sensors (e.g., depth and stereo cameras) while maintaining its maneuverability. Further, it may require large capacity boards to simultaneously collect the information from all sensors. This dataset may be collected using the next generation of the GRIFFIN ornithopters, which aim at providing flapping-wing robots with higher payload and maneuverability for aerial robot applications.

- **Stabilizing event data for flapping-wing robot perception**. The flapping strokes produced by ornithopters represent an additional challenge for aerial robot perception. They generate perturbations in the camera frame causing motion blur in standard frames and triggering additional events [42]. The *event-by-event* methods of Chapters 3 and 4 integrate ASAP [100] to control event packaging, and analyze current and previous event information to provide more stable perception estimations. However, these algorithms require a careful tuning process to enhance their performance based on the robot specifications and the scene configuration. Flapping stroke perturbations may be reduced by stabilizing the camera motion. For instance, the work in [146] proposes a mechanical stabilizing system for a large-scale flapping-wing robot. It uses two actuators to stabilize the pitch and roll variations on the camera orientation. However, the installation of a mechanical stabilizer on a flapping-wing robot is constrained by the payload of the platform and its power capacity. Future work focuses on implementing an event stabilization algorithm that behaves similarly to an electronic stabilizer [199], and studying its advantages for aerial robot perception. The algorithm aims at reducing the additional events displacement caused by external disturbances such as flapping strokes. It would be particularly useful for methods relying on *event images* by providing sharper edges compared to frames obtained with non-stabilized events. Preliminary results of this research have been presented in [65], where the stabilization algorithm prevents large event displacements on samples of the dataset described in Section 6.4 (i.e., The GRIFFIN perception dataset [43]). Current work focuses on proposing a general framework for event stabilization and exploring its additional advantages for ornithopter perception.

- **Visual odometry for flapping-wing robots**. This Ph.D. Thesis proposes low-level and high-level perception approaches validated on multirotors and ornithopters. However, there are still many perception topics to be addressed by event-based perception for aerial robots. For instance, the current literature lacks approaches that focus on the problem of estimating the motion of ornithopters during flight using either standard or/and event cameras. The validation of the dataset in Section 6.4 depicts an initial result of performing Visual Inertial Odometry (VIO) using grayscale frames collected on a flapping-wing robot. This problem entails several challenges including the restricted payload of the platform to mount powerful processors and the vibrations produced by the flapping strokes. Future work aims at studying the Visual Odometry (VO) problem in flapping-wing robots using event-based vision. The solution could fuse information from standard and event cameras as in [90] together with inertial measurements for Visual Intertial Odometry [200]. Recently, some works have addressed the problem of estimating the camera position using event-based monocular [201] and stereo VO [202] [197]. These works could serve as an inspiration for the development of an event-based VO solution for ornithopters.

Bibliography

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.

[2] J. Faigl, P. Váňa, R. Pěnička, and M. Saska. Unsupervised learning-based flexible framework for surveillance planning with aerial vehicles. *Journal of Field Robotics*, 36(1):270–301, 2019.

[3] J. P. Rodríguez-Gómez, A. Gómez Eguíluz, J. R. Martínez-De Dios, and A. Ollero. Auto-tuned event-based perception scheme for intrusion monitoring with uas. *IEEE Access*, 9:44840–44854, 2021.

[4] R. Lopez Lopez, M. J. Batista Sanchez, M. Perez Jimenez, B. C. Arrue, and A. Ollero. Autonomous uav system for cleaning insulators in power line inspection and maintenance. *Sensors*, 21(24):8488, 2021.

[5] G. Silano, R. Baca, T.and Penicka, D. Liuzza, and M. Saska. Power line inspection tasks with multi-aerial robot systems via signal temporal logic specifications. *IEEE Robotics and Automation Letters*, 6(2):4169–4176, 2021.

[6] L. Teixeira, I. Alzugaray, and M. Chli. Autonomous aerial inspection using visual-inertial robust localization and mapping. In *Field and Service Robotics*, pages 191–204. Springer, 2018.

[7] P. Rudol and P. Doherty. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. In *2008 IEEE Aerospace Conference*, pages 1–8, 2008.

[8] V. Spurnỳ, T. Báča, Martin Saska, R. Pěnička, T. Krajník, J. Thomas, D. Thakur, G. Loianno, and V. Kumar. Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles. *Journal of Field Robotics*, 36(1):125–148, 2019.

[9] A. Tagliabue, M. Kamel, R. Siegwart, and J. Nieto. Robust collaborative object transportation using multiple mavs. *The International Journal of Robotics Research*, 38(9):1020–1044, 2019.

[10] A. Lopez-Lora, P.J. Sanchez-Cuevas, A. Suarez, A. Garofano-Soldado, A. Ollero, and G. Heredia. Mhyro: Modular hybrid robot for contact inspection and maintenance in oil & gas plants. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1268–1275, 2020.

[11] A.E. Jiménez-Cano, D. Sanalitro, M. Tognon, A. Franchi, and J. Cortés. Precise cable-suspended pick-and-place with an aerial multi-robot system. *Journal of Intelligent & Robotic Systems*, 105(3):1–13, 2022.

[12] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, D. Scaramuzza, and L. M. Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[13] A. Santamaria-Navarro, G. Loianno, J. Sola, V. Kumar, and J. Andrade-Cetto. Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors. *Autonomous robots*, 42(6):1263–1280, 2018.

[14] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[15] F. Alsadik, B.and Remondino. Flight planning for lidar-based uas mapping applications. *ISPRS international journal of geo-information*, 9(6):378, 2020.

[16] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[17] R. Caballero, J. Parra, M. Trujillo, F. J. Pérez-Grau, A. Viguria, and A. Ollero. Aerial robotic solution for detailed inspection of viaducts. *Applied Sciences*, 11(18):8404, 2021.

[18] V. Valseca, J. Paneque, J. R. Martínez-de Dios, and A. Ollero. Real-time lidar-based semantic classification for powerline inspection. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 478–486. IEEE, 2022.

[19] G. Belloni, M. Feroli, A. Ficola, S. Pagnottelli, and P. Valigi. An autonomous aerial vehicle for unmanned security and surveillance operations: design and test. In *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 1–4, 2007.

[20] K. Zhang, P. Chermprayong, D. Tzoumanikas, W. Li, M. Grimm, M. Smentoch, S. Leutenegger, and M. Kovac. Bioinspired design of a landing system with soft shock absorbers for autonomous aerial robots. *Journal of Field Robotics*, 36(1):230–251, 2019.

[21] Y. Chen, S. Xu, Z. Ren, and P. Chirarattananon. Collision resilient insect-scale soft-actuated aerial robots with high agility. *IEEE Transactions on Robotics*, 37(5):1752–1764, 2021.

[22] A. Suárez, A. M. Giordano, K. Kondak, G. Heredia, and A. Ollero. Flexible link long reach manipulator with lightweight dual arm: Soft-collision detection, reaction, and obstacle localization. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 406–411, 2018.

[23] A. Braithwaite, T. Alhinai, M. Haas-Heger, E. McFarlane, and M. Kovač. Tensile web construction and perching with nano aerial vehicles. In *Robotics research*, pages 71–88. Springer, 2018.

[24] F. Ruiz, B. Arrue, and A. Ollero. Sophie: Soft and flexible aerial vehicle for physical interaction with the environment. *arXiv preprint arXiv:2205.12883*, 2022.

[25] S. H. Song, G. Y. Yeon, and H. R. Shon, H. W.and Choi. Design and control of soft unmanned aerial vehicle "s-cloud". *IEEE/ASME Transactions on Mechatronics*, 26(1):267–275, 2021.

[26] E. Ajanic, M. Feroskhan, S. Mintchev, F. Noca, and D. Floreano. Bioinspired wing and tail morphing extends drone flight capabilities. *Science Robotics*, 5(47):eabc2897, 2020.

[27] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza. The foldable drone: A morphing quadrotor that can squeeze and fly. *IEEE Robotics and Automation Letters*, 4(2):209–216, 2019.

[28] A. Fabris, K. Kleber, D. Falanga, and D. Scaramuzza. Geometry-aware compensation scheme for morphing drones. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 592–598, 2021.

[29] W. Stewart, L. Guarino, Y. Piskarev, and D. Floreano. Passive perching with energy storage for winged aerial robots. *Advanced Intelligent Systems*, page 2100150, 2021.

[30] R. Zufferey, J. Tormo Barbero, D. Feliu, S. Nekoo, J. A. Acosta, and A. Ollero. How ornithopters can perch autonomously on a branch. *arXiv preprint arXiv:2207.07489*, 2022.

[31] A. K. Stowers and D. Lentink. Folding in and out: passive morphing in flapping wings. *Bioinspiration & biomimetics*, 10(2):025001, 2015.

[32] E. Savastano, V. Perez-Sanchez, B.C. Arrue, and A. Ollero. High-performance morphing wing for large-scale bio-inspired unmanned aerial vehicles. *IEEE Robotics and Automation Letters*, 7(3):8076–8083, 2022.

[33] F. Rodríguez, J.l Díaz-Báñez, E. Sanchez-Laulhe, J. Capitán, and A. Ollero. Kinodynamic planning for an energy-efficient autonomous ornithopter. *Computers & Industrial Engineering*, 163:107814, 2022.

[34] R. Zufferey, J. Tormo-Barbero, M. M. Guzman, F. J. Maldonado, E. Sanchez-Laulhe, P. Grau, J. A. Perez, M.and Acosta, and A. Ollero. Design of the high-payload flapping wing robot e-flap. *IEEE Robotics and Automation Letters*, 6(2):3097–3104, 2021.

[35] M.M. Guzmán, C. Ruiz Páez, F. J. Maldonado, R. Zufferey, J. Tormo-Barbero, J.Á Acosta, and A. Ollero. Design and comparison of tails for bird-scale flapping-wing robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6358–6365, 2021.

[36] W. . Roderick, M. Cutkosky, and D. Lentink. Bird-inspired dynamic grasping and perching in arboreal environments. *Science Robotics*, 6(61):eabj7562, 2021.

[37] W. Shyy, H. Aono, S. K. Chimakurthi, P. Trizila, C-K Kang, C. Cesnik, and H. Liu. Recent progress in flapping wing aerodynamics and aeroelasticity. *Progress in Aerospace Sciences*, 46(7):284–327, 2010.

[38] C. Ruiz, JÁ. Acosta, and A. Ollero. Aerodynamic reduced-order volterra model of an ornithopter under high-amplitude flapping. *Aerospace Science and Technology*, 121:107331, 2022.

[39] F. J. Maldonado, J. A. Acosta, J. Tormo-Barbero, P. Grau, M. M. Guzman, and A. Ollero. Adaptive nonlinear control for perching of a bioinspired ornithopter. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1385–1390, 2020.

[40] F. Helbling and W. Robert J. A review of propulsion, power, and control architectures for insect-scale flapping-wing vehicles. *Applied Mechanics Reviews*, 70(1), 2018.

[41] S. Tijmons, G. de Croon, B. Remes, C. De Wagter, and M. Mulder. Obstacle avoidance strategy using onboard stereo vision on a flapping wing mav. *IEEE Trans. Robot.*, 33(4):858–874, 2017.

[42] A. Gómez Eguíluz, J.P. Rodríguez-Gómez, J.L. Paneque, P. Grau, J.R. Martínez de Dios, and A. Ollero. Towards flapping wing robot visual perception: Opportunities and challenges. In *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, pages 335–343, 2019.

[43] J. P. Rodríguez-Gómez, R. Tapia, J. Paneque, P. Grau, A. Gómez Eguíluz, J. R. Martínez-de Dios, and A. Ollero. The griffin perception dataset: Bridging the gap between flapping-wing flight and robotic perception. *IEEE Robotics and Automation Letters*, 6(2):1066–1073, 2021.

[44] G. A. Folkertsma, W. Straatman, N. Nijenhuis, C. H. Venner, and S. Stramigioli. Robird: A robotic bird of prey. *IEEE Robotics Automation Magazine*, 24(3):22–29, 2017.

[45] K.A. Boahen. A burst-mode word-serial address-event link-i: transmitter design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(7):1269–1280, 2004.

[46] S. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas. *Event-based neuromorphic systems*. John Wiley & Sons, 2014.

[47] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128$\times$ 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[48] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, S. Debole, M.and Esser,

T. Delbruck, M. Flickner, and D. Modha. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[49] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, L. Camunas-Mesa, R. Berner, M. Rivas-Perez, T. Delbruck, S. Liu, R. Douglas, P. Hafliger, G. Jimenez-Moreno, A. Civit Ballcels, T. Serrano-Gotarredona, A. J. Acosta-Jimenez, and B. Linares-Barranco. Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing– learning–actuating system for high-speed visual object recognition and tracking. *IEEE Transactions on Neural Networks*, 20(9):1417–1438, 2009.

[50] J. Conradt. On-board real-time optic-flow for miniature event-based vision sensors. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1858–1863, 2015.

[51] I. Alzugaray and M. Chli. Ace: An efficient asynchronous corner tracker for event cameras. In *2018 International Conference on 3D Vision (3DV)*, pages 653–661, 2018.

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

[53] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza. Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 126(12):1394–1414, 2018.

[54] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *British Machine Vision Conference (BMVC 2017)*. University of Zurich, 2017.

[55] P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[56] G. de Croon, M. A. Groen, C. De Wagter, B. Remes, R. Ruijsink, and B. W. van Oudheusden. Design, aerodynamics and autonomy of the DelFly. 7(2):025003, may 2012.

[57] A. Gómez Eguíluz, J.P. Rodríguez-Gómez, J.R. Martínez-de Dios, and A. Ollero. Asynchronous event-based line tracking for time-to-contact maneuvers in uas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5978–5985, 2020.

[58] A. Gómez Eguíluz, J.P. Rodríguez-Gómez, R. Tapia, F.J. Maldonado, J.Á. Acosta, J.R. Martínez-de Dios, and A. Ollero. Why fly blind? event-based visual guidance for ornithopter robot flight. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1958–1965, 2021.

[59] J. P. Rodríguez-Gómez, R. Tapia, M. M. Guzmán Garcia, J. R. Martínez-de Dios, and A. Ollero. Free as a bird: Event-based dynamic sense-and-avoid for ornithopter robot flight. *IEEE Robotics and Automation Letters*, 7(2):5413–5420, 2022.

[60] J.P. Rodríguez-Gomez, A. Gómez Eguíluz, J.R. Martínez-de Dios, and A. Ollero. Asynchronous event-based clustering and tracking for intrusion monitoring in uas. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8518–8524, 2020.

[61] J.R. Martínez-de Dios, A. Gómez Eguíluz, J.P. Rodríguez-Gómez, R. Tapia, and A. Ollero. Towards uas surveillance using event cameras. In *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 71–76, 2020.

[62] J.P. Rodríguez-Gómez, R. Tapia, A. Gómez Eguíluz, J. R. Martínez-de Dios, and A. Ollero. Uav human teleoperation using event-based and frame-based cameras. In *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*, pages 1–5, 2021.

[63] J. P. Rodríguez-Gómez, M. Di Cicco, S. Nardi, and D. Nardi. Mapping infected crops through uav inspection: The sunflower downy mildew parasite case. In *Advances and Trends in Artificial Intelligence. From Theory to Practice*, pages 495–503. Springer International Publishing, 2019.

[64] J. P. Rodríguez-Gómez, A. Gómez Eguíluz, J. R. Martínez-de Dios, and A. Ollero. ROSS-LAN: RObotic Sensing Simulation scheme for bioinspired robotic bird LANding. In *Robot 2019: Fourth Iberian Robotics Conference*, pages 48–59. Springer International Publishing, 2020.

[65] J. P. Rodríguez-Gómez, G. Gallego, J. R. Martínez-de Dios, and A. Ollero. Stabilizing event data on flapping-wing robots for simpler perception. In *IEEE ICRA Workshop on Challenges of Flapping-wing aerial robots*, pages 1–1, 2022.

[66] J. Manderscheid, A. Sironi, N. Bourdis, D. Migliore, and V. Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[67] R. Benosman. Vision without frames: A semiotic paradigm of event based computer vision. *Biosemiotics*, 3(1):1–16, 2010.

[68] R. Benosman, C. Clercq, X. Lagorce, S. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):407–417, 2014.

[69] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Regnier. Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics. *IEEE Transactions on Robotics*, 28(5):1081–1089, 2012.

[70] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. Benosman. Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2017.

[71] R. Li, D. Shi, Y. Zhang, K. Li, and R. Li. Fa-harris: A fast and asynchronous corner detector for event cameras. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6223–6229, 2019.

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

[73] V. Vasco, A. Glover, and C. Bartolozzi. Fast event-based harris corner detection exploiting the advantages of event-driven cameras. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4144–4149, 2016.

[74] E. Mueggler, C. Bartolozzi, and D. Scaramuzza. Fast event-based corner detection. In *BMVC*, 2017.

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

[76] A. Glover, A. Dinale, L. De Souza Rosa, S. Bamford, and C. Bartolozzi. luvharris: A practical corner detector for event-cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

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

[78] F. Barranco, C. Fermuller, and E. Ros. Real-time clustering and multi-target tracking using event-based sensors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5764–5769, 2018.

[79] C. Yizong. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[80] M. Liu and T. Delbruck. Block-matching optical flow for dynamic vision sensors: Algorithm and fpga implementation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[81] L. Min and T. Delbruck. Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. In *British Machine Vision Conference (BMVC) 2018*. Proc. of BMVC 2018, September 2018.

[82] J. Wu, K. Zhang, Y. Zhang, X. Xie, and G. Shi. High-speed object tracking with dynamic vision sensor. In L. Wang, Y. Wu, and J. Gong, editors, *Proceedings of the 5th China High Resolution Earth Observation Conference (CHREOC 2018)*, pages 164–174, Singapore, 2019. Springer Singapore.

[83] V. Vasco, A. Glover, E. Mueggler, D. Scaramuzza, L. Natale, and C. Bartolozzi. Independent motion detection with event-driven cameras. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 530–536, 2017.

[84] A. Mitrokhin, C. Fermüller, C. Parameshwara, and Y. Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[85] D. Falanga, S. Kim, and D. Scaramuzza. How fast is too fast? the role of perception latency in high-speed sense and avoid. *IEEE Robotics and Automation Letters*, 4(2):1884–1891, 2019.

[86] 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 *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3867–3876, 2018.

[87] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):1964–1980, 2021.

[88] L. Pan, C. Scheerlinck, X. Yu, R. Hartley, M. Liu, and Y. Dai. Bringing a blurry frame alive at high frame-rate with an event camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[89] S. Tulyakov, D. Gehrig, S. Georgoulis, J. Erbach, M. Gehrig, Y. Li, and D. Scaramuzza. Time lens: Event-based video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16155–16164, June 2021.

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

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

[92] C. Tomasi and T. K. Detection. Tracking of point features. *Int. J. Comput. Vis*, 9:137–154, 1991.

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

[94] J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[95] M. Ester, H. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[96] C. M Bishop and N. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[97] S. Uppada. Centroid based clustering algorithms—a clarion study. *International Journal of Computer Science and Information Technologies*, 5(6):7309–7313, 2014.

[98] A. Jaegle, S. Phillips, and K. Daniilidis. Fast, robust, continuous monocular egomotion computation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 773–780, 2016.

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

[100] R Tapia, J. R. Martínez-de Dios, A. Gómez Eguíluz, and A. Ollero. Asap: Adaptive transmission scheme for online processing of event-based algorithms. *arXiv preprint arXiv:2209.08602*, 2022.

[101] T. Stoffregen, G. Gallego, T. Drummond, L. Kleeman, and D. Scaramuzza. Event-based motion segmentation by motion compensation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7243–7252, 2019.

[102] A. Mitrokhin, Z. Hua, C. Fermuller, and Y. Aloimonos. Learning visual motion segmentation using event surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

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

[104] B. He, H. Li, S. Wu, D. Wang, Q. Zhang, Z.and Dong, C. Xu, and F. Gao. Fast-dynamic-vision: Detection and tracking dynamic objects with event and depth sensing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3071–3078, 2021.

[105] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[106] L. Gao, Y. Liang, J. Yang, S. Wu, C. Wang, J. Chen, and L. Kneip. Vector: A versatile event-centric benchmark for multi-sensor slam. *IEEE Robotics and Automation Letters*, 2022.

[107] P. Negri, M. Soto, B. Linares-Barranco, and T. Serrano-Gotarredona. Scene context classification with event-driven spiking deep neural networks. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 569–572, 2018.

[108] Y. Li, H. Zhou, B. Yang, Y. Zhang, Z. Cui, H. Bao, and G. Zhang. Graph-based asynchronous event processing for rapid object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 934–943, October 2021.

[109] Y. Deng, Y. Li, and H. Chen. Amae: Adaptive motion-agnostic encoder for event-based object classification. *IEEE Robotics and Automation Letters*, 5(3):4596–4603, 2020.

[110] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, and D. Koltun, V.and Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[111] H. Jabbari, G. Oriolo, and H. Bolandi. An adaptive scheme for image-based visual servoing of an underactuated uav. *International Journal of Robotics and Automation*, 29(1):92–104, 2014.

[112] S. Kim, H. Seo, S. Choi, and H. J. Kim. Vision-guided aerial manipulation using a multirotor with a robotic arm. *IEEE/ASME Transactions on Mechatronics*, 21(4):1912–1923, 2016.

[113] J. Thomas, G. Loianno, K. Sreenath, and V. Kumar. Toward image based visual servoing for aerial grasping and perching. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2113–2118, 2014.

[114] G. Spedicato, S.and Notarstefano, H. H. Bülthoff, and A. Franchi. Aggressive maneuver regulation of a quadrotor uav. In *Robotics Research*, pages 95–112. Springer, 2016.

[115] R. Muthusamy, A. Ayyad, M. Halwani, D. Swart, D. Gan, L. Seneviratne, and Y. Zweiri. Neuromorphic eye-in-hand visual servoing. *IEEE Access*, 9:55853–55870, 2021.

[116] G. Dong and Z. Zhu. Position-based visual servo control of autonomous robotic manipulators. *Acta Astronautica*, 115:291–302, 2015.

[117] Y. He, J. Gao, and Y. Chen. Deep learning-based pose prediction for visual servoing of robotic manipulators using image similarity. *Neurocomputing*, 491:343–352, 2022.

[118] D. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–8, 2016.

[119] B. J. Pijnacker Hordijk, K. Y. W. Scheper, and G. de Croon. Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, 35(1):69–90.

[120] N. J. Sanket, C. M. Parameshwara, C. D. Singh, Ashwin V. Kuruttukulam, C. Fermüller, D. Scaramuzza, and Y. Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657, 2020.

[121] S. Sun, G. Cioffi, C. de Visser, and D. Scaramuzza. Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters*, 6(2):580–587, 2021.

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

[123] C. Brändli, J. Strubel, S. Keller, and T. Scaramuzza, Dand Delbruck. Elised — an event-based line segment detector. In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–7, 2016.

[124] Rafael G. Von G., J. Jakubowicz, J. Morel, and G. Randall. Lsd: a line segment detector. *Image Processing On Line*, 2:35–55, 2012.

[125] D. Reverter Valeiras., X. Clady, S. Ieng, and R. Benosman. Event-based line fitting and segment detection using a neuromorphic visual sensor. *IEEE Transactions on Neural Networks and Learning Systems*, 30(4):1218–1230, 2019.

[126] L. Everding and J. Conradt. Low-latency line tracking using event-based dynamic vision sensors. *Frontiers in Neurorobotics*, 12:4, 2018.

[127] R. Gomez-Ojeda, F. Moreno, D. Zuñiga-Noël, D. Scaramuzza, and J. Gonzalez-Jimenez. Pl-slam: A stereo slam system through the combination of points and line segments. *IEEE Transactions on Robotics*, 35(3):734–746, 2019.

[128] F. Nardi, B. Della Corte, and G. Grisetti. Unified representation and registration of heterogeneous sets of geometric primitives. *IEEE Robotics and Automation Letters*, 4(2):625–632, 2019.

[129] P. VC Hough. Method and means for recognizing complex patterns. *US patent*, 3(6), 1962.

[130] P. Duda, R.and Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[131] Zhengyou Z. Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 666–673 vol.1, 1999.

[132] P. I Corke and O. Khatib. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

[133] Farid K. Four-dimensional guidance and control of movement using time-to-contact: Application to automated docking and landing of unmanned rotorcraft systems. *The International Journal of Robotics Research*, 33(2):237–267, 2014.

[134] D. Lee. General tau theory: evolution to date. *Perception*, 38(6):837, 2009.

[135] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[136] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[137] F. Real, A. Torres-González, P. Ramón Soria, J. Capitán, and A. Ollero. Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles. *International Journal of Advanced Robotic Systems*, 17(4):1–13, 2020.

[138] H. K Khalil. *Nonlinear control*, volume 1. Pearson, 2014.

[139] S. Hrabar. Reactive obstacle avoidance for rotorcraft uavs. In *IEEE/RSJ IROS*, pages 4967–4974, 2011.

[140] H. Oleynikova, D. Honegger, and M. Pollefeys. Reactive avoidance using embedded stereo vision for mav flight. In *ICRA*, 2015.

[141] J. Borenstein, Y. Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[142] E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. Towards evasive maneuvers with quadrotors using dynamic vision sensors. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–8, 2015.

[143] Z. Ma, C. Wang, Y. Niu, X. Wang, and L. Shen. A saliency-based reinforcement learning approach for a uav to avoid flying obstacles. *Robot. Auto. Syst.*, 100:108–118, 2018.

[144] F. Garcia Bermudez and R. Fearing. Optical flow on a flapping wing robot. In *IEEE/RSJ IROS*, pages 5027–5032, 2009.

[145] G. de Croon, E. de Weerdt, C. de Wagter, and B. Remes. The appearance variation cue for obstacle avoidance. In *IEEE ICRB*, pages 1606–1611, 2010.

[146] E. Pan, X. Liang, and W. Xu. Development of vision stabilizing system for a large-scale flapping-wing robotic bird. *IEEE Sensors Journal*, 20(14):8017–8028, 2020.

[147] X. Clady, C. Clercq, S. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, and R. Benosman. Asynchronous visual event-based time-to-contact. *Frontiers in Neuroscience*, 8:9, 2014.

[148] C. Walters and S. Hadfield. Evreflex: Dense time-to-impact prediction for event-based obstacle avoidance. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1304–1309, 2021.

[149] D. Zhang and G. Lu. Segmentation of moving objects in image sequence: A review. *Circuits, Systems and Signal Processing*, 20(2):143–183, 2001.

[150] R. Tapia, A. Gómez Eguíluz, J. R. Martınez-de Dios, and A. Ollero. ASAP: Adaptive scheme for asynchronous processing of event-based vision algorithms. In *IEEE ICRA Workshop on Unconventional Sensors in Robotics*, pages 1–3, 2020.

[151] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun. Dense monocular depth estimation in complex dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[152] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719, 2019.

[153] A. Dietsche, G. Cioffi, J. Hidalgo-Carrió, and D. Scaramuzza. Powerline tracking with event cameras. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6990–6997, 2021.

[154] J. J. Acevedo, I. Maza, A. Ollero, and B.C. Arrue. An efficient distributed area division method for cooperative monitoring applications with multiple uavs. *Sensors*, 20(12):3448, 2020.

[155] S. Sarkar, M. W. Totaro, and K. Elgazzar. Intelligent drone-based surveillance: application to parking lot monitoring and detection. In Charles M. Shoemaker, Hoa G. Nguyen, and Paul L. Muench, editors, *Unmanned Systems Technology XXI*, volume 11021, pages 13 – 19. International Society for Optics and Photonics, SPIE, 2019.

[156] S. Berrahal, J. Kim, S. Rekhis, N. Boudriga, D. Wilkins, and J. Acevedo. Border surveillance monitoring using quadcopter uav-aided wireless sensor networks. *Journal of Communications Software and Systems*, 12(1):67–82, 2016.

[157] R. S. Dimitrova, M. Gehrig, D. Brescianini, and D. Scaramuzza. Towards low-latency high-bandwidth control of quadrotors using event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4294–4300, 2020.

[158] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, and Y. Sandamirskaya. Event-driven vision and control for uavs on a neuromorphic chip. In *2021 IEEE*

International Conference on Robotics and Automation (ICRA), pages 103–109, 2021.

[159] F. Mahlknecht, D. Gehrig, J. Nash, F. M. Rockenbauer, B. Morrell, J. Delaune, and D. Scaramuzza. Exploring event camera-based odometry for planetary robots. *arXiv preprint arXiv:2204.05880*, 2022.

[160] J. Balaram, M. Aung, and M. P. Golombek. The ingenuity helicopter on the perseverance rover. *Space Science Reviews*, 217(4):1–11, 2021.

[161] A.Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4470. IEEE, 2017.

[162] J. R. Martínez de Dios, F. J. Pérez-Grau, A. Torres-González, J. J. Acevedo, J. Paneque, A. Viguria, D. Fuego, J. R. Astorga, and A. Ollero. *GRVC-CATEC: Aerial Robot Co-worker in Plant Servicing (ARCOW)*, pages 211–242. Springer International Publishing, Cham, 2020.

[163] A. Cesetti, E. Frontoni, A. Mancini, P. Zingaretti, and S. Longhi. A vision-based guidance system for uav navigation and safe landing using natural landmarks. *Journal of intelligent and robotic systems*, 57(1):233–257, 2010.

[164] F. J. Perez-Grau, R. Ragel, F. Caballero, A. Viguria, and A. Ollero. An architecture for robust uav navigation in gps-denied areas. *Journal of Field Robotics*, 35(1):121–145, 2018.

[165] P. Adarsh, P. Rathi, and M. Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694. IEEE, 2020.

[166] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[167] F. Rojas, C.G. Puntonet, M. Rodriguez-Alvarez, I. Rojas, and R. Martin-Clemente. Blind source separation in post-nonlinear mixtures using competitive learning, simulated annealing, and a genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(4):407–416, 2004.

[168] R.S. Tavares, T.C. Martins, and M.S.G. Tsuzuki. Simulated annealing with adaptive neighborhood: A case study in off-line robot path planning. *Expert Systems with Applications*, 38(4):2951–2965, 2011.

[169] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[170] M.A. Pérez-Cutiño, A. Gómez Eguíluz, J.R. Martínez-de Dios, and A. Ollero. Event-based human intrusion detection in uas using deep learning. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 91–100, 2021.

[171] E. Rohmer, S. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[172] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[173] Shital Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Marco Hutter and Roland Siegwart, editors, *Field and Service Robotics*, pages 621–635, Cham, 2018. Springer International Publishing.

[174] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken

Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[175] J. Kaiser, J. C. Vasquez Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. M. Zöllner. Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 127–134, 2016.

[176] H. Rebecq, D. Gehrig, and D. Scaramuzza. Esim: an open event camera simulator. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 969–982. PMLR, 29–31 Oct 2018.

[177] J. Han, J. Lee, and D. Kim. Ornithopter modeling for flight simulation. In *2008 International Conference on Control, Automation and Systems*, pages 1773–1777, 2008.

[178] A. T Pfeiffer, J. Lee, J. Han, and H. Baier. Ornithopter flight simulation based on flexible multi-body dynamics. *Journal of Bionic Engineering*, 7(1):102–111, 2010.

[179] F. Fei, Z. Tu, Y. Yang, J. Zhang, and X. Deng. Flappy hummingbird: An open source dynamic simulation of flapping wing robots and animals. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9223–9229, 2019.

[180] I. Bozcan and E. Kayacan. Au-air: A multi-modal unmanned aerial vehicle dataset for low altitude traffic surveillance. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8504–8510, 2020.

[181] F. Maffra, L. Teixeira, Z. Chen, and M. Chli. Real-time wide-baseline place recognition using depth completion. *IEEE Robotics and Automation Letters*, 4(2):1525–1532, 2019.

[182] M. Fonder and M. Van Droogenbroeck. Mid-air: A multi-modal dataset for extremely low altitude drone flights. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[183] A. Majdik, C. Till, and D. Scaramuzza. The zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*, 36(3):269–273, 2017.

[184] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[185] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman. The blackbird dataset: A large-scale dataset for uav perception in aggressive flight. In J. Xiao, T. Kröger, and O. Khatib, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 130–139, Cham, 2020. Springer International Publishing.

[186] K. Sun, K. Mohta, B. Pfrommer, S. Watterson, M.and Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[187] R. Lopez-Lopez, V. Perez-Sanchez, P. Ramon-Soria, A. Martín-Alcántara, R. Fernandez-Feria, B.C. Arrue, and A. Ollero. A linearized model for an ornithopter in gliding flight: Experiments and simulations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7008–7014, 2020.

[188] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

[189] A. Mitrokhin, C. Ye, C. Fermüller, Y. Aloimonos, and T. Delbruck. Ev-imo: Motion segmentation dataset and learning pipeline for event cameras. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6105–6112, 2019.

[190] C. Luo, X. Li, Y. Li, and Q. Dai. Biomimetic design for unmanned aerial vehicle safe landing in hazardous terrain. *IEEE/ASME Transactions on Mechatronics*, 21(1):531–541, 2016.

[191] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[192] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart. Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311, 2016.

[193] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[194] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072, 2017.

[195] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*, 2018.

[196] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017.

[197] Y. Zhou, G. Gallego, and S. Shen. Event-based stereo visual odometry. *IEEE Transactions on Robotics*, 37(5):1433–1450, 2021.

[198] T. Zhou, D. Fan, M. Cheng, J. Shen, and L. Shao. Rgb-d salient object detection: A survey. *Computational Visual Media*, 7(1):37–69, 2021.

[199] T. Delbruck, V. Villanueva, and L. Longinotti. Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2636–2639, 2014.

[200] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. Continuous-time visual-inertial odometry for event cameras. *IEEE Transactions on Robotics*, 34(6):1425–1440, 2018.

[201] J. Hidalgo-Carrió, G. Gallego, and D. Scaramuzza. Event-aided direct sparse odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5781–5790, June 2022.

[202] Y. Zuo, J. Yang, J. Chen, X. Wang, Y. Wang, and L. Kneip. Devo: Depth-event camera visual odometry in challenging conditions. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2179–2185, 2022.