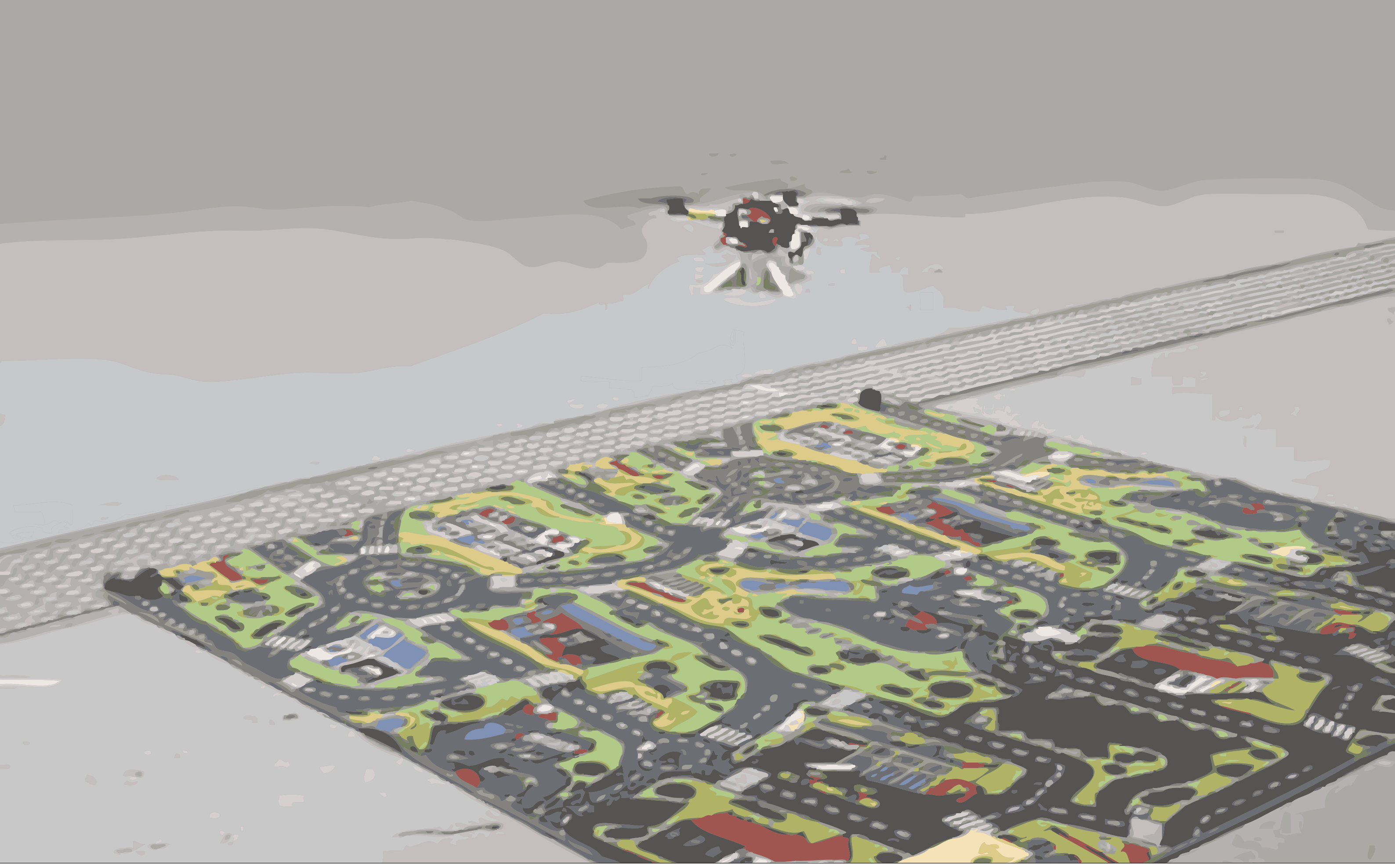
Faculty of Aerospace Engineering

Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow

B.J. Pijnacker Hordijk

Friday 9th December, 2016



Delft University of Technology

Challenge the future

Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

B.J. Pijnacker Hordijk

Friday 9th December, 2016

Faculty of Aerospace Engineering · Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow"** by **B.J. Pijnacker Hordijk** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: Friday 9th December, 2016

Readers:

Dr.ir. Q.P. Chu

Dr. G.C.H.E. de Croon

Ir. K.Y.W. Scheper

Dr. M. Snellen

Acknowledgements

This thesis would not have reached its current level without the guidance and support of my supervisors, Guido de Croon and Kirk Scheper, and I would like to thank them dearly for it. Guido has been an inexhaustible source of inspiration and enthusiasm throughout the project, always encouraging me to go further and to investigate uncovered ground. Kirk helped me tremendously with the implementation and experiments, getting me started with working with the sensor and programming in C, helping out with the hardware on the quadrotor, and piloting the drone during flight tests.

Also, a word of thanks goes out to my family and friends for keeping up my spirit during the project. In particular, I would like to thank my parents; your advice and never-ceasing support kept me on the right path in difficult times.

B.J. Pijnacker Hordijk
Delft, December 9, 2016

Abstract

Small flying robots can perform landing maneuvers using bio-inspired optical flow by maintaining a constant divergence. However, optical flow is typically estimated from frame sequences recorded by standard miniature cameras. This requires processing full images onboard, limiting the update rate of divergence measurements, thus the speed of the control loop and the robot. Event-based cameras overcome these limitations by only measuring pixel-level brightness changes at microsecond temporal accuracy, hence providing an efficient mechanism for optical flow estimation.

This thesis presents, to the best of our knowledge, the first research integrating event-based optical flow estimation into the control loop of a flying robot. We extend an existing 'local plane fitting' algorithm to obtain an improved and more computationally efficient optical flow estimation method, valid for a wide range of optical flow velocities. This method is validated for real event sequences. In addition, a method for estimating the divergence from event-based optical flow is introduced, which accounts for the aperture problem.

The developed algorithms are implemented in a constant divergence landing controller onboard of a quadrotor. Flight tests demonstrate that, using event-based optical flow, accurate divergence estimates can be obtained over a wide range of speeds. This enables the quadrotor to perform very fast landing maneuvers.

Acronyms

AE	Angular Error
AER	Address-Event Representation
ATIS	Asynchronous Time-based Image Sensor
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
CP	Cross-Product
DAVIS	Dynamic and Active pixel Vision Sensor
DSF	Direction Selective Filtering
DVS	Dynamic Vision Sensor
eDVS	Embedded Dynamic Vision Sensor
EE	Endpoint Error
EMD	Elementary Motion Detector
FoC	Focus of Contraction
FoE	Focus of Expansion
IM	Integration Method
IMU	Inertial Measurement Unit
LK	Lucas-Kanade
MAV	Micro Air Vehicle
meDVS	Miniature embedded Dynamic Vision Sensor
NES	Normal-to-Edge Search
NRMSE	Normalized Root Mean Square Error
PEE	Projection Endpoint Error
PF	Planar Flow
PM	Probability Mapping
PNF	Planar Normal Flow
RSS	Residual Sum of Squares
STPF	Spatiotemporal Plane Fitting

TSS	Total Sum of Squares
VSLAM	Visual Simultaneous Localization and Mapping

List of Symbols

Greek Symbols

α	Direction of an optical flow field
Δt_R	Refractory period for rejecting consecutive events
$\delta x, \delta y, \delta t$	Relative coordinates of an event in space-time
η	Optical flow estimate density
$\vartheta_x, \vartheta_y, \vartheta_z$	Observer translational velocities normalized by Z_0, referred to as the visual observables
$\mathbf{\Theta}$	Visual observables vector ($= [\vartheta_x, \vartheta_y, \vartheta_z]^T$)
$\mathbf{\Pi}$	Homogeneous plane parameter vector ($= [p_x,\, p_y,\, p_t,\, p_0]^T$)
$\mathbf{\Pi}^*$	Reduced plane parameter vector ($= \left[p_x^*, p_y^*\right]^T$)
ρ_F	Optical flow estimation rate
Σ_e	Surface of active events in space-time
Σ_S^i	Sum of all values of S along optical flow direction i
τ	Time-to-contact
ϕ, θ, ψ	Euler angles representing the camera attitude
ω_x, ω_y	Ventral flows in x- and y-direction

Roman Symbols

$\mathbf{A}$	Regression matrix in a linear least-squares system
D	Flow field divergence
d	Distance of an event to a fitted plane.
$\mathcal{D}$	Pixel undistortion map

$\mathbf{e}$	Event tuple generated by an event-based camera
f	Focal length
h	Height above ground
I	Light intensity perceived by a pixel
K	Total confidence value for visual observable estimates
$k_{\rho_F}, k_{\text{Var}\{S\}}, k_{R^2}$	Individual confidence values for ρ_F, $\text{Var}\{S\}$, R^2 respectively
$k_1, \ldots, k_5$	Lens distortion model parameters
k_f	Time constant applied during recursive updating of the flow field
k_P	Proportional control gain in divergence controller
k_S	Timestamp difference factor for clustering events
k_t	Time constant in a low-pass filter
L	List of recent events
M	Map of last event timestamps
$\mathbf{n}$	Normal flow vector
m	Number of directions in the visual observables estimator
n_{min}	Minimal number of events for a reliable plane fit
n_R	Maximum number of rejected events in optical flow estimation
P	Polarity of an event
p, q, r	Angular body rate components of the observer's ego-motion
q_0, q_x, q_y, q_z	Quaternions representing the camera attitude
R^2	Coefficient of determination
S_i	Projected optical flow position along a direction i
T	Vertical thrust
t	Time
U, V, W	Translational velocity components corresponding to X, Y, Z
u, v	Optical flow vector components, pixels per second
$\hat{u}, \hat{v}$	Normalized optical flow vector components on the image plane
V_i	Projected optical flow velocity along a direction i
$\mathbf{V}$	Local optical flow vector ($= [u,\ v]^T$)
$\mathbf{W}$	Weight matrix for a weighted least-squares system
W_i	Individual weight of a direction i
X, Y, Z	Metric position in Cartesian coordinates.
x, y	Pixel x- and y-locations on a pixel array

x_u, y_u	Undistorted pixel x- and y-locations
$\hat{x}_f, \hat{y}_f$	Location of the Focus of Expansion on the image plane
$\hat{x}, \hat{y}$	Normalized x- and y-coordinates of a point on the image plane
x_p, y_p	Camera principal point coordinates
$\mathbf{y}$	Observations matrix in a linear least-squares system
Z_0	Distance of the nodal point to the ground plane along the optical axis
Z_X, Z_Y	Gradients of a planar surface with respect to the image plane

Subscripts

$\mathcal{B}$	Body-fixed reference frame
$\mathcal{C}$	Camera-fixed reference frame
$\mathcal{W}$	World reference frame

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid advances in micro-electronics catalyzed the development of tiny flying robots (Floreano & Wood, 2015), formally referred to as Micro Air Vehicles (MAVs). Due to their size and agility, MAVs excel at applications in confined and cluttered environments. However, achieving autonomous flight with very small MAVs (for example, the 20-gram DelFly Explorer (De Wagter, Tijmons, Remes, & De Croon, 2014)) is a significant challenge due to strict weight and power limitations for on-board equipment. For overcoming this challenge, visual navigation techniques observed in flying insects are highly interesting.

Visual navigation in flying insects is primarily based on optical flow, the apparent motion of brightness patterns perceived by an observer due to relative motion with respect to the environment (Gibson, 1979). In essence, optical flow provides information on the ratio of velocity to distance, such that the actual metric distance to the environment is not directly available. Instead, flying insects navigate based on certain visual observables extracted from the optical flow field that relate to ego-motion. When performing landings, honeybees were seen to control their descent based on the *divergence* of the optical flow field perceived from the ground (Baird, Boeddeker, Ibbotson, & Srinivasan, 2013). When looking down, flow field divergence conveys the vertical velocity scaled by the height. By maintaining a constant divergence in downward motion, an observer approaches the ground while exponentially decreasing its downward speed. For flying robots capable of vertical landing, this is an interesting strategy, which has been explored in several experiments with rotorcraft MAVs (e.g. De Croon, 2016; Herissé, Hamel, Mahony, & Russotto, 2012).

While such optical flow based navigation strategies are bio-inspired, most visual sensors employed for measuring optical flow differ significantly from their natural counterparts. Commonly used miniature cameras operates in a *frame-based* manner: full frames are obtained by periodically measuring brightness at all pixels. This is a relatively inefficient process for motion perception, since the information output rate is independent of the actual dynamics in the scene. Static parts of a frame are recorded as well as changing parts, even though only the latter are relevant for motion perception. Therefore, follow-up processing of a full frame is necessary, which at present still requires significant processing. In addition, fast inter-frame displacements lead to motion blur, which limits the accuracy of resulting optical flow estimates or requires complex algorithms to account for this. These characteristics are undesirable for optical flow measurement on-board miniature flying robots, which are subject to strict computational limits and exhibit fast dynamics.

In contrast, biological vision systems, such as the compound eyes of insects, employ an *event-driven* mechanism; they measure changes in the perceived scene at the moment of detection (Posch, Serrano-Gotarredona, Linares-barranco, & Delbrück, 2014). Several researchers have attempted to minic the sensory system in insects in order to measure optical flow. For example, Ruffier and Franceschini (2005, 2014) developed a 2-photodetector neuromorphic chip for measuring translational optical flow on-board of a tethered rotorcraft MAV, and Floreano et al. (2013) designed a miniature curved compound eye, highly based on the fruit fly Drosophila.

In particular, neuromorphic *event-based cameras* are a promising class of sensors for optical flow perception. When a pixel of an event-based camera measures a relative increase or decrease in brightness, it registers an event, mapping its pixel location to the timestamp and sign of the change. This timestamp is obtained with microsecond resolution and latencies in the range of 3 to 15 s. This fundamentally differs from conventional frame-based visual sensing, since the sensor only provides output when and where a change occurs, without being restricted to a fixed sampling rate. This difference is illustrated in Figure 1-1 for a rotating dot. In addition, event-based pixel architectures enable intrascene dynamic ranges exceeding 120 dB (Yang, Liu, & Delbruck, 2015). These characteristics are highly desirable in robotic visual navigation. Experiments using event-based cameras demonstrated high performance of visual control systems through low-latency state updates, efficient data processing, and operation over a wide range of illumination conditions (Conradt, Berner, et al., 2009; Delbruck & Lang, 2013).

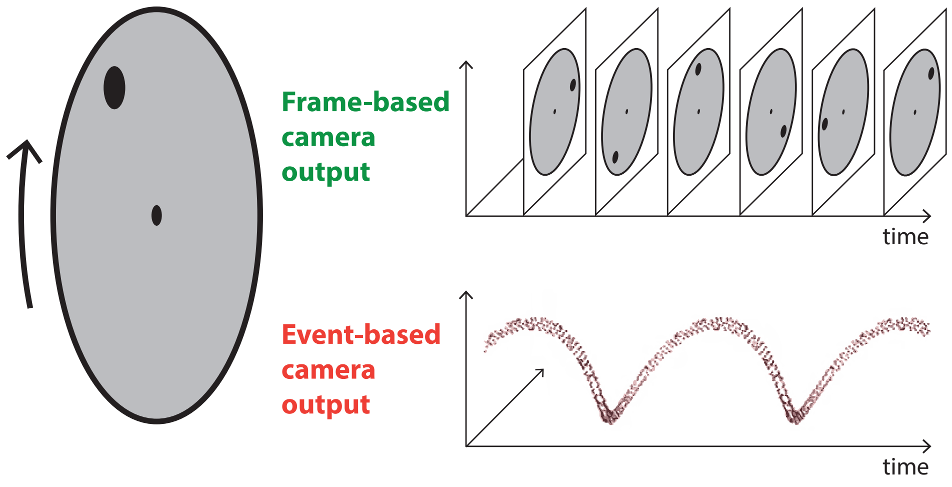


Figure 1-1: Output from frame-based (top) and event-based (bottom) cameras observing a rotating dot stimulus, represented in space-time. Adapted from Mueggler et al. (2014a).

This novel approach to visual sensing is in general incompatible with state-of-the-art computer vision algorithms for estimating optical flow, due to the lack of absolute brightness measurements. Therefore, several event-based methods for optical flow estimation (e.g. Bardow, Davison, & Leutenegger, 2016; Benosman et al., 2014) as well as benchmarking datasets (Barranco, Fermuller, Aloimonos, & Delbruck, 2016; Ruckauer & Delbruck, 2016) have been developed. Of the existing techniques, the local plane fitting approach by Benosman et al. (2014) is the most promising based on its application to estimating time-to-contact (the reciprocal of flow field divergence) in simple scenes (Clady et al., 2014) and good results in Ruckauer and Delbruck (2016). However, until now, no study has discussed an implementation of event-based optical flow into an optical flow based control loop of an MAV.

1-1 Motivation and research question

The Micro Air Vehicle Laboratory (MAVLab) of Delft University of Technology performs, among other projects, research regarding the application of optical flow to landing of MAVs. Recently experiments have been executed regarding constant divergence landing tasks using downward-facing frame-based cameras on-board of quadrotors (De Croon, 2016; Ho & De Croon, 2016; Ho, de Croon, van Kampen, Chu, & Mulder, 2016). The application of event-based cameras in this task is of great interest, since it has the potential to significantly increase the speed of landing maneuvers through its more efficient sensing mechanism, and to enable navigation over a wider range of illumination conditions.

This thesis is a continuation of prior work performed for the MAVlab (Paz Gomes Verdugo, 2015). The aim is to develop a fully functional landing control system for MAVs using event-based optical flow and estimates of relevant visual observables (in particular, flow field divergence). In the preliminary work of this thesis, we compare the performance of recently developed event-based optical flow methods and compatible estimators for visual observables. In the second stage, the most suitable methods are further developed, implemented on a quadrotor MAV, and validated experimentally during closed-loop landing experiments.

The main research question is formulated as follows:

Can event-based optical flow estimation improve the performance of optical flow based landing control systems for Micro Air Vehicles?

1-2 Structure of this thesis

The main contributions of this thesis are presented in the scientific paper in Part I. This paper can be read as a standalone document and consists primarily of (1) a concise overview of related work, (2) a description of the developed algorithms and evaluation of their performance, (3) flight test results of a quadrotor with the developed algorithms implemented in a constant divergence landing controller, and (4) the main conclusions and recommendations. The remainder of this thesis provides appendices to support the paper.

Part II contains a more in-depth review of relevant literature and serves as background material for the unacquainted reader. In Chapter 2 several concepts related to optical flow based navigation and frame-based optical flow estimation are introduced. Next, Chapter 3 introduces event-based cameras, their working principle, and related research. Chapter 4 separately reviews existing approaches for estimating event-based optical flow and visual observables. The part is closed in Chapter 5 which provides a synthesis of the literature results.

Next, Part III documents a preliminary comparison between available methods for estimating optical flow and visual observables. The findings in this comparison form the basis for the final algorithms presented in the paper. To start with, in Chapter 6 the methodology and datasets used in this analysis are presented. Subsequently, specific results for estimating local optic flow are presented in Chapter 7 followed by the process of estimating visual observables in Chapter 8. The preliminary results are discussed in Chapter 9, in which we also briefly relate the findings to the final algorithms presented in the paper.

Part IV contains individual appendices that support the final state of the work described in the paper. This part starts with Chapter 10, which discusses how the event-based camera used in this work is calibrated. Then, Chapter 11 discusses the experimental setup in detail. Chapter 12 describes the details of the implementation of the algorithms in the experimental setup. Last, Chapter 13 presents additional results to support observations in the paper.

Part I

Paper

Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow

B.J. Pijnacker Hordijk, K.Y.W. Scheper, and G.C.H.E. De Croon
Control & Simulation, Department Control & Operations
Delft University of Technology, Delft, The Netherlands

***Abstract*—Small flying robots can perform landing maneuvers using bio-inspired optical flow by maintaining a constant divergence. However, optical flow is typically estimated from frame sequences recorded by standard miniature cameras. This requires processing full images on-board, limiting the update rate of divergence measurements, thus the speed of the control loop and the robot. Event-based cameras overcome these limitations by only measuring pixel-level brightness changes at microsecond temporal accuracy, hence providing an efficient mechanism for optical flow estimation. This paper presents, to the best of our knowledge, the first work integrating event-based optical flow estimation into the control loop of a flying robot. We extend an existing 'local plane fitting' algorithm to obtain an improved and more computationally efficient optical flow estimation method, valid for a wide range of optical flow velocities. This method is validated for real event sequences. In addition, a method for estimating the divergence from event-based optical flow is introduced, which accounts for the aperture problem. The developed algorithms are implemented in a constant divergence landing controller on-board of a quadrotor. Experiments show that, using event-based optical flow, accurate divergence estimates can be obtained over a wide range of speeds. This enables the quadrotor to perform very fast landing maneuvers.**

***Index Terms*—Event-based vision, neuromorphic, optical flow, Micro Air Vehicles, autonomous landing.**

I. Introduction

RAPID advances in micro-electronics catalyzed the development of tiny flying robots [1], formally referred to as Micro Air Vehicles (MAVs). Due to their size and agility, MAVs excel at applications in confined and cluttered environments. However, achieving autonomous flight with very small MAVs (for example, the 20-gram DelFly Explorer [2]) is a significant challenge due to strict weight and power limitations for on-board equipment. For overcoming this challenge, visual navigation techniques observed in flying insects are highly interesting.

Visual navigation in flying insects is primarily based on optical flow, the apparent motion of brightness patterns perceived by an observer due to relative motion with respect to the environment [3]. In essence, optical flow provides information on the ratio of velocity to distance, such that the actual metric distance to the environment is not directly available. Instead, flying insects navigate based on certain visual observables extracted from the optical flow field that relate to ego-motion. Honeybees were seen to control their descent during landings based on the *divergence* of the optical flow field perceived from the ground [4]. When looking down, flow field divergence conveys the vertical velocity scaled by the height. By maintaining a constant divergence in downward motion, an observer approaches the ground while exponentially decreasing its downward speed. For flying robots capable of vertical landing, this is an interesting strategy. This application has been explored in several experiments with rotorcraft MAVs [5]–[8].

While such optical flow based navigation strategies are bio-inspired, most visual sensors employed for measuring optical flow differ significantly from their natural counterparts. Commonly used miniature cameras operate in a *frame-based* manner: full frames are obtained by periodically measuring brightness at all pixels. This is a relatively inefficient process for motion perception, since the information output rate is independent of the actual dynamics in the scene. Static parts of a frame are recorded as well as changing parts, even though only the latter are relevant for motion perception. Therefore, follow-up processing of a full frame is necessary, which at present still requires significant processing. In addition, fast inter-frame displacements lead to motion blur, which limits the accuracy of resulting optical flow estimates or requires complex algorithms to account for this. These characteristics are undesirable for optical flow measurement on board miniature flying robots, which are subject to strict computational limits and exhibit fast dynamics.

In contrast, biological vision systems, such as the compound eyes of insects, employ an *event-driven* mechanism; they measure changes in the perceived scene at the moment of detection [9]. Several researchers have attempted to minic the sensory system in insects in order to measure optical flow. For example, in [10] a tethered rotorcraft MAV was equipped with a 2-photodetector neuromorphic chip for measuring translational optical flow. In [11] a miniature curved compound eye design, highly based on the fruit fly Drosophila, was presented.

In particular, *event-based cameras* are a promising class of sensors for optical flow perception. When a pixel of an event-based camera measures a relative increase or decrease in brightness, it registers an event, mapping its pixel location to the timestamp and sign of the change. This timestamp is obtained with microsecond resolution and latencies in the range of 3 to 15 μs. In addition, event-based pixel architectures enable intrascene dynamic ranges exceeding 120 dB [12]. These characteristics are highly desirable in robotic visual navigation. Experiments using event-based cameras demonstrated high performance of visual control systems through low-latency state updates, efficient data processing, and operation over a wide range of illumination conditions [13], [14].

This novel approach to visual sensing is in general incompatible with state-of-the-art computer vision algorithms for estimating optical flow, due to the lack of absolute brightness measurements. Therefore, several event-based methods for optical flow estimation [15]–[19] as well as benchmarking datasets [20], [21] have been developed. Of the existing techniques, the local plane fitting approach by [16] is the most promising based on its application to estimating time-to-contact (the reciprocal of flow field divergence) in simple scenes [22] and good results in [21]. However, until now, no study has discussed an implementation of event-based optical flow into an optical flow based control loop of an MAV.

This paper contains *three main contributions*. First, a novel method for estimating event-based optical flow inspired by [16] is introduced. Its applicability is extended to a wider range of velocities, while improving computational efficiency. Second, a method for incorporating event-based optical flow into visual estimation of divergence is proposed, which accounts for the aperture problem that occurs in most existing event-based optical flow methods. Third, the proposed algorithms for event-based optical flow divergence estimation are incorporated in a constant divergence landing controller on-board of a quadrotor. To the best of the authors' knowledge, this paper presents the first functional event-based visual controller on-board of an MAV.

We begin by discussing related work concerning landing using optical flow, event-based vision, and optical flow estimation in Section II. Then, Section III introduces the mathematical models describing the relations between the ego-motion of the MAV, optical flow, and visual observables used in this work. In Section IV the estimation method for event-based optical flow is described and evaluated. Subsequently, Section V introduces the approach to estimating visual observables from event-based optical flow, followed by an assessment of its performance in combination with the optical flow method. Afterwards, Section VI presents flight test results of the full estimation pipeline during constant divergence landing maneuvers. Lastly, the main conclusions and recommendations for future work are presented in Section VII.

II. Related Work

This section introduces the fundamental concepts and previous contributions relevant for this work. First, Section II-A discusses bio-inspired landing strategies involving optical flow and associated research involving MAVs. Second, the concept of event-based cameras is described in Section II-B. Third, an overview of existing approaches to optical flow estimation is provided in Section II-C, including both frame-based camera applications and recently developed event-based techniques.

A. Landing Using Optical Flow

Although optical flow does not by itself provide metric scale to motion, information from optical flow fields is useful for several navigation tasks, including landing. Simple bio-inspired strategies were proposed in the past decades that utilize the visual observables in the optical flow field perceived from the ground. Such strategies form a lightweight alternative for visual estimation of three-dimensional structure, ego-motion, and relative pose, which can be performed through visual Simultaneous Localization And Mapping (SLAM) [23] or visual odometry [24]. These techniques have become increasingly efficient over the last few years [25], [26], yet still require processing and maintaining large amounts of measurement data. This strongly contrasts with optical flow based techniques, in which all information required for navigation is contained in a small number of visual observables.

The visual observables related to horizontal motion above ground are the ventral flows ω_x, ω_y, referring to the average flows along the x and y image axes. In several experiments with a tethered MAV [10], [27], the authors mimicked navigation strategies seen in insects, which have been observed to follow terrain and land using ventral flow [28]. By maintaining a constant ratio between forward motion and height and at the same time slowing down, they perform smooth landings. Ventral flows may also be used for hover stabilization to augment visual vertical control [29].

In recent aerial robotic applications, mainly visual observables based on vertical motion were applied, allowing control of vertical dynamics independent of horizontal motion. One of these observables is flow field divergence D, i.e. the ratio of vertical velocity to height. Its reciprocal is the time-to-contact to the ground τ. Similar to ventral flows, divergence was seen to guide docking and landing motion in biology. In [4] honeybees were seen to keep D constant. Hence, velocity is decreased exponentially, ensuring a smooth touchdown. Other strategies for vertical landing exist based on τ, such as the constantly decreasing τ strategy observed in braking human drivers [30]. This strategy provides more control over the landing trajectory [29], though it involves more parameters.

In practical control systems, a constant divergence approach suffers from instability as height decreases, due to self-induced oscillations. In [8] it was shown that a relation exists between the employed divergence controller gain and the height at which oscillations occur. A main insight then was that a drone could detect its own oscillations, and in this way determine its height. This strategy can be employed to trigger a separate final touchdown phase, or to continuously measure height by landing at a near-unstable control gain. The finding in [8] also contains an important key to high-performance optical flow divergence landings. This was used in [31] to develop an adaptive gain controller, which detects the height at the start of a landing maneuver, sets initial controller gains based on the height, and lands while exponentially reducing the gains. Although the landings performed were quite fast compared to landings in the literature ($D = 0.3$ compared to a typical $D = 0.05$ [6]), the speed of the landings in [31] are still quite limited by the standard cameras available on the used AR drone 2.0.

B. Event-Based Cameras

Inspired by the workings of biological retinas, event-based cameras rely on a sensing mechanism that fundamentally differs from their frame-based counterparts. In frame-based cameras the pixel values are measured at fixed time intervals to produce a sequence of images. In event-based cameras, on the other hand, pixel activity is driven by light intensity changes. Whenever a pixel measures a local change, it produces a *event*. Specifically, this occurs when the pixel's logarithmic intensity measurement $I(x, y, t)$ (at pixel location (x, y) and timestamp t) increases or decreases beyond a threshold C:

$$|\Delta\left(\log I\left(x,y,t\right)\right)| > C \tag{1}$$

Events are encoded according to an Address-Event Representation (AER) [32], which consists of event information encoded by an address and the timestamp of detection. Typically, an event encodes the pixel position (x, y), the timestamp t, and the polarity $P \in \{-1, 1\}$, which indicates the sign of the intensity change. A visualization of a basic stream of events, in comparison to an equivalent set of frames, is shown in Fig. 1.

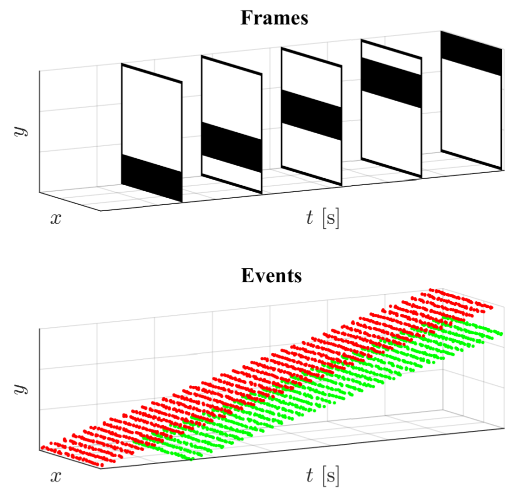


Fig. 1. Frame-based and event-based visual output generated from a simple synthetic scene, in which a black horizontal bar moves upward. The events are visualized as points in space-time, hence showing the trajectory of the leading and trailing edges of the black bar. Events with positive polarity are highlighted in green; those with negative polarity are marked in red.

The sensor used in this work is the Dynamic Vision Sensor (DVS) - specifically, the DVS128 - which is the first commercially available event-based camera [33]. It features a 128x128 pixel grid operating at an intrascene dynamic range of 120 dB, measuring events at 1 µs timing resolution with a latency of 15 µs [32]. A picture of the DVS is shown in Fig. 2. Since the availability of the DVS, other event-based cameras have been developed. Most notable are the Asynchronous Time-based Image Sensor (ATIS) [34], which measures absolute intensity as well as polarity for each event, and the Dynamic and Active pixel Vision Sensor (DAVIS) [35], whose pixels record events as well as full frames. Interesting in the context of this work is the 2.2 gram micro embedded DVS (meDVS) [36], which is highly suitable for on-board MAV applications.

Event-based cameras have several interesting applications for robotic navigation. Initial work has been performed on visual SLAM with event-based cameras [37], [38]. In [39] a pose estimation algorithm based on line tracking is applied to a quadrotor, enabling it to perform aggressive maneuvers . Some studies demonstrate the ability to simultaneously reconstruct intensity maps and relative pose [40] and, more recently, three-dimensional structure [41]. Others aim at combining the benefits of event-based and frame-based vision using the DAVIS. For example, the method presented in [42] uses frames to identify visual features and events to track their position in high-speed motion, in order to perform visual odometry.

Fig. 2. Picture of the event-based camera employed in this work, the DVS.

C. Optical Flow Estimation

In the following, we discuss available techniques for estimating optical flow. Many recent visual navigation experiments, in particular those with commercially available quadrotors, employ standard frame-based cameras, in combination with follow-up processing algorithms. Others employ off-the-shelf optical mouse sensors or specialized neuromorphic optical flow sensors, which directly yield translational optical flow output. For event-based cameras, several techniques have recently been developed, yet these have seen limited applications in robotic navigation.

1) Estimation from Frame Sequences: At present, a wide range of optical flow estimation techniques is available for frame-based cameras. Most of these algorithms derive from the brightness constancy assumption, which states that, when a pixel flows from one frame to another, its intensity I is conserved [43]. This assumption leads to the well-known optical flow constraint:

$$I_x(x,y)u + I_y(x,y)v = -I_t(x,y) \tag{2}$$

where x and y are the pixel position and u and v denote the unknown optical flow components in pixels per second. The partial derivatives I_x, I_y, and I_t are obtained from two sequential frames. Since this equation provides two unknown components, a second constraint is necessary to obtain optical flow. Many recent methods aim at providing a dense optical flow field estimate, where optical flow is estimated for any pixel for the frame. In this case, a global cost function minimization is performed, in which a second constraint is provided by prior knowledge. An example of such a constraint is the requirement of smoothness in the flow field in the well-known Horn-Schunck technique [44]. Recent dense optical flow algorithms provide accurate results for complex scenes, but at the cost of high computation times [43].

In recent real-time robotic applications, the most popular frame-based algorithm is the Lucas-Kanade algorithm [45]. This algorithm is originally developed for estimating optical flow in the local neighborhood of a pixel. In order to solve (2), the second assumption is that u and v are constant across neighboring pixels. Therefore, a least-squares system can be composed based on (2), using I_x, I_y, and I_t from neighboring pixels. This system can be solved for u and v. This technique is mainly applied to sparse estimation, where motion is only computed locally at visual features of interest.

Local optical flow estimation techniques are subjected to the aperture problem, which occurs when motion ambiguity is present due to a limited field of view [46]. This occurs along object contours which lack clearly distinguishable corner points. The result is that only *normal flow* can be estimated, which is the motion component normal to the contour's orientation. At corner locations, this ambiguity is not present. Only at these points, optical flow is estimated using Lucas-Kanade. Therefore, a corner detection algorithm (e.g. [47]) can be applied to first obtain points of interest in the frame. This strategy is applied in many recent optical flow based landing experiments [7], [8], [29], [48].

Alternatively, in [6] a 'pyramidal' variant of Lucas-Kanade is applied [49] to account for large displacements. This is a coarse-to-fine approach: optical flow is first computed for a highly downsampled frame. Then, the frame is iteratively refined, computing more detailed optical flow at each refinement level, using the estimate at the previous level to initialize the estimate.

In all these approaches, it is necessary to process full frames, either to find features of interest such as corners, or to obtain sufficiently detailed dense optical flow. While it is possible to use low resolution frames for faster processing, this comes at the cost of reduced detail and hence lower accuracy. Event-based cameras are much less subject to this trade-off, since their output directly highlights locations of interest for estimating optical flow.

2) Optical Flow Sensors: Hardware-based solutions for estimating optical flow have also been applied. Some researchers employ off-the-shelf optical mouse sensors for measuring translational optical flow e.g. [50]. In addition, the visual motion processing in insects inspired researchers to develop highly simplified optical flow sensors, such as the 2-photodetector elementary motion detector was developed for the tethered MAV research in [10], [51]. Optical flow sensors achieve relatively high sampling rates due to their simplicity. However, their operating principle is generally limited to measuring translational flow. For measuring patterns of optical flow, such as divergence, multiple separate sensors need to be applied and integrated.

3) Event-Based Methods: Since the introduction of the DVS and subsequently developed sensors, several different approaches to event-based optical flow estimation have been developed. Most of these techniques operate on each newly detected event and its spatiotemporal neighborhood, providing sparse optical flow estimates. However, in most cases, the algorithms do not distinguish between corner points and other visual features. Thus, they primarily estimate normal flow. In the following, a brief review of recent approaches is presented.

An adaptation of the frame-based Lucas-Kanade tracker is introduced in [15]. Similar to the original algorithm, it solves the optical flow constraint by including the local neighborhood of a pixel. Since absolute measurements of I are not available, the authors replaced the intensity I by the sum of event polarities at a pixel location, obtained over a fixed time window. The reconstructed 'relative intensity' is used to numerically estimate I_x, I_y, and I_t. However, the number of events is generally too low for this approach to provide accurate gradient estimates, in particular for the temporal gradient I_t.

In [16] an algorithm is presented that operates on the spatiotemporal representation of events as a point cloud (as shown in Fig. 1). When representing a sequence of events by three-dimensional points of (x, y, t), they form surface-like structures. The gradient of such a surface relates to the motion of the object that triggered the events. By computing a local tangent plane to an event and its neighbor events, normal flow for that event is estimated. A follow-up study employs this algorithm for detecting and tracking corners from neighboring normal flow vectors, hence obtaining fully observable optical flow [52]. However, real-time results are not yet demonstrated with a non-parallelized implementation.

In [17] a technique is introduced that estimates optical flow on object contours, based on both events and absolute intensity measurements. Input events are used to locate motion boundaries on contours. Along each boundary, motion is estimated using the width of the contour, which is computed from the local event distribution and the absolute intensity. The latter can be reconstructed from events, but having separate intensity measurements (e.g. from a DAVIS or ATIS sensor) simplifies the process.

A bio-inspired approach is proposed in [18]. In this approach, optical flow is estimated using direction- and speed-selective filters based on the first stages of visual processing in humans. A bank of spatiotemporal filters is employed, each of which is maximally selective for a certain direction and speed of optical flow. For each new event, the neighboring event cloud is convolved with the filters to obtain a confidence measure for each filter. Optical flow for that event is then obtained from the sum of the confidence measures weighted by direction.

More complex event-based algorithms have also been developed, which have not demonstrated real-time performance, but show promising results. In [53] a phase-based optical flow method is discussed, which is developed for high-frequency textures. The algorithm is compared to other event-based methods [15]–[17], indeed showing significant accuracy improvements. Also, an approach was presented for simultaneous estimation of dense optical flow and absolute intensity [19]. This is the only available approach aimed towards dense optical flow estimation. Visual results of this method are encouraging, yet a quantitative evaluation is not performed.

Recently, several datasets for event-based visual navigation have been published. The set in [20] provides both frame and event measurements from a DAVIS sensor accompanied by odometry measurements. This facilitates comparison between frame-based and event-based techniques for optical flow estimation or visual odometry. However, to the best of the authors' knowledge, an actual comparison of existing techniques has not yet been published for this set. In this respect, the work in [21] is more relevant for this work, as it features both an event-based dataset and a comparison of various optical flow algorithms. These are variants of the techniques in [15] and [16], as well as a basic direction selective algorithm.

We select the local plane fitting algorithm in [16] as the basis of the approach in our work. It has shown the most promising results in [21] and has recently been incorporated into follow-up experiments [22], [52]. In addition, its implementations yielded real-time operation for high event measurement rates.

III. Relations between Optical Flow, Ego-Motion, and Visual Observables

This section defines the optical flow model, which relates the ego-motion of an MAV equipped with a downward facing camera to the perceived optical flow, and the visual observables presented in Section II-A. These relations form the basis for our evaluation methods applied in subsequent sections, in which ground truth values for optical flow and visual observables are computed.

In the derivation, use is made of three reference frames: $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{W}$, which describe the body, camera and inertial world reference frames respectively. Their definitions are illustrated in Fig. 3. In each reference frame, a position is denoted through the coordinates (X, Y, Z), with corresponding velocity components (U, V, W). The body frame is centered at the center of gravity of the MAV. The rotation from $\mathcal{W}$ to $\mathcal{B}$ is described by standard Euler angles φ, θ, ψ, denoting roll, pitch, and yaw respectively. Similarly, p, q, and r describe the roll, pitch, and yaw rotational rates.

The camera reference frame $\mathcal{C}$ is centered at the focal point of the DVS. The camera is assumed to be located directly below the MAV's center of gravity, with $X_{\mathcal{C}} = Y_{\mathcal{B}}$ and $Y_{\mathcal{C}} = -X_{\mathcal{B}}$. However, we account for an offset ΔZ between $Z_{\mathcal{C}}$ and $Z_{\mathcal{B}}$, i.e. $Z_{\mathcal{C}} = Z_{\mathcal{B}} + \Delta Z$.

The relations between optical flow and ego-motion are based on the pinhole camera model formulation in [54]. In this formulation, pixel locations (x, y) in the sensor's pixel grid and optical flow components (u, v) in pixels per second are represented by their metric, real-world equivalents $(\hat{x}, \hat{y})$ and $(\hat{u}, \hat{v})$. The mapping between the two representations is composed of two parts: correction for lens distortion and transformation through the camera's intrinsics matrix. A two-parameter version of the commonly used Brown-Conrady model [55] is applied to model the relation between (x, y) and their undistorted equivalent (x_u, y_u):

$$\begin{bmatrix} x - x_p \\ y - y_p \end{bmatrix} = \begin{bmatrix} x_u - x_p \\ y_u - y_p \end{bmatrix} \left(1 + k_1 r_u^2 + k_2 r_u^4\right) \tag{3}$$

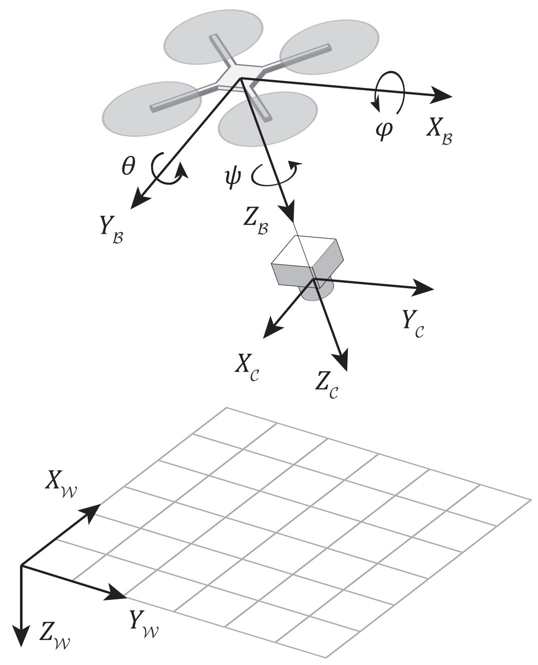


Fig. 3. Definitions of the world ($\mathcal{W}$), body ($\mathcal{B}$), and camera ($\mathcal{C}$) reference frames. The Euler angle definitions and their signs are also shown.

The point (x_p, y_p) in Eq. (3) represents the camera's principal point, and $r_u = \sqrt{(x_u - x_p)^2 + (y_u - y_p)^2}$ is the radial distance to the principal point. Second, the undistorted pixels and corresponding optical flow estimates relate to their metric equivalent through scaling with the focal length f:

$$\hat{x} = \frac{x_u - x_p}{f}, \quad \hat{y} = \frac{y_u - y_p}{f}, \quad \hat{u} = \frac{u}{f}, \quad \hat{v} = \frac{v}{f} \tag{4}$$

In order to obtain the parameters k_1, k_2, x_p, y_p, and f, the DVS is calibrated using the Camera Calibration Toolbox in MATLAB [56]. Since the DVS does not record absolute intensity, artificial images are generated by recording events from a flashing checkerboard pattern on an LCD screen, similar to [39].

The optical flow components $(\hat{u}, \hat{v})$ due to an arbitrary point moving in $\mathcal{C}$ with motion $(U_{\mathcal{C}}, V_{\mathcal{C}}, W_{\mathcal{C}})$ at depth $Z_{\mathcal{C}}$ are obtained as follows:

$$\begin{aligned} \hat{u} &= -\frac{U_{\mathcal{C}}}{Z_{\mathcal{C}}} + \hat{x}\frac{W_{\mathcal{C}}}{Z_{\mathcal{C}}} - p + r\hat{y} + q\hat{x}\hat{y} - p\hat{x}^2 \\ \hat{v} &= -\frac{V_{\mathcal{C}}}{Z_{\mathcal{C}}} + \hat{y}\frac{W_{\mathcal{C}}}{Z_{\mathcal{C}}} + q - r\hat{x} + q\hat{y}^2 - p\hat{x}\hat{y} \end{aligned} \tag{5}$$

When separate measurements of the rotational rates p, q, and r are available (for instance, from rate gyro measurements), the optical flow in Eq. (5) can be corrected for the ego-rotation of the camera. This derotation leaves only the translational optical flows $\hat{u}_T$, $\hat{v}_T$. Further, if all visible points are part of a single plane, their coordinates $Z_{\mathcal{C}}$ are interrelated. In most indoor applications, floor surfaces can be assumed planar and horizontal. However, MAVs perform fast horizontal maneuvers through rolling and pitching, such that the ground plane may have a slight inclination in $\mathcal{C}$. In this case, $Z_{\mathcal{C}}$ is expressed through three parameters: the distance Z_0 to the plane at $(\hat{x}, \hat{y}) = (0, 0)$, and the plane slopes Z_X, Z_Y.

$$Z_{\mathcal{C}} = Z_0 + Z_X X_{\mathcal{C}} + Z_Y Y_{\mathcal{C}} \tag{6}$$

This can be rewritten into:

$$\frac{Z_{\mathcal{C}} - Z_0}{Z_{\mathcal{C}}} = Z_X \hat{x} + Z_Y \hat{y} \tag{7}$$

When the surface is flat, the slopes are the tangents of the roll and pitch angles, and can therefore be obtained from separate sensors in e.g. an IMU:

$$Z_X = -\tan\varphi, \quad Z_Y = \tan\theta \tag{8}$$

Now we define the *scaled velocities* ϑ_x, ϑ_y, ϑ_z as follows:

$$\vartheta_x = \frac{U_{\mathcal{C}}}{Z_0}, \quad \vartheta_y = \frac{V_{\mathcal{C}}}{Z_0}, \quad \vartheta_z = \frac{W_{\mathcal{C}}}{Z_0} \tag{9}$$

Following the derivation in [48], substituting Eq. (7) and Eq. (9) into Eq. (5) leads to the following expression:

$$\begin{aligned} \hat{u}_T &= (-\vartheta_x + \hat{x}\vartheta_z)(1 - Z_X\hat{x} - Z_Y\hat{y}) \\ \hat{v}_T &= (-\vartheta_y + \hat{y}\vartheta_z)(1 - Z_X\hat{x} - Z_Y\hat{y}) \end{aligned} \tag{10}$$

The scaled velocities represent the visual observables presented in Section II-A. ϑ_x and ϑ_y are the opposites of the ventral flows, i.e. $\omega_x = -\vartheta_x$, $\omega_y = -\vartheta_y$. The vertical component ϑ_z is proportional to the flow field divergence, since $D = \nabla \cdot \mathbf{V} = 2\vartheta_z$. It is also the inverse of time-to-contact $\tau = Z_{\mathcal{C}}/W_{\mathcal{C}} = 1/\vartheta_z$.

Note that, in the presence of the vertical camera offset ΔZ, the rotational rates p and q induce additional translational velocity components into $U_{\mathcal{C}}$ and $V_{\mathcal{C}}$. These propagate to ϑ_x and ϑ_y, such that:

$$\vartheta_x = \frac{V_{\mathcal{B}}}{Z_{\mathcal{C}}} - p\frac{\Delta Z}{Z_{\mathcal{C}}}, \quad \vartheta_y = \frac{U_{\mathcal{B}}}{Z_{\mathcal{C}}} + q\frac{\Delta Z}{Z_{\mathcal{C}}} \tag{11}$$

These corrections are accounted for in the computation of ground truth optical flow and values of ϑ_x and ϑ_y.

IV. Event-Based Optical Flow Estimation

This section describes our optical flow estimation approach. Since it is based on the work in [16], this baseline approach is explained first in Section IV-A. Then, the proposed modifications for achieving higher efficiency (Section IV-B) and timestamp-based selection (Section IV-C) are discussed. In Section IV-D the result of our improvements is evaluated in comparison to the baseline algorithm.

A. The Baseline Plane Fitting Algorithm

The main working principle of the baseline algorithm is based on the space-time representation of events as a point cloud. In the following, an event is denoted as a space-time point according to $\mathbf{e}_n = (x, y, t)$, where x and y represent the undistorted pixel locations. Note that the polarity P is not considered; we group positive and negative polarity events and process them separately.

Let $\Sigma_e(x, y) = t$ be a mapping describing the surface along which events are positioned. The shape of Σ_e is a result of the feature geometry and, in particular, its motion. In the case of a locally linear feature (such as an edge) and constant motion, this surface reduces to a plane. This is clearly visible in the example scene in Fig. 1. With these assumptions, Σ_e can be approximated by a tangent plane within a limited range of x, y, and t.

For each newly detected event $\mathbf{e}_n$, a plane $\mathbf{\Pi}$ is computed that fits best to all neighboring events $\mathbf{e}_i$ for which $x_i \in \left[x_n - \frac{1}{2}\Delta x,\, x_n + \frac{1}{2}\Delta x\right]$, $y_i \in \left[y_n - \frac{1}{2}\Delta y,\, y_n + \frac{1}{2}\Delta y\right]$, and $t_i \in [t_n - \Delta t,\, t_n]$, where $\Delta x, \Delta y, \Delta t$ indicate spatial and temporal windows. The spatial windows are generally small and are both set to 5 pixels. The temporal window setting has a large influence on the detectable speed and interference of multiple features, which is discussed further in Section IV-C.

The plane $\mathbf{\Pi} = [p_x, p_y, p_t, p_0]^T$ is computed through an iterative process of linear least squares regression and outlier rejection. It is represented in homogeneous coordinates, such that the following hold for any event $\mathbf{e}_i$ that intersects with $\mathbf{\Pi}$:

$$p_x x_i + p_y y_i + p_t t_i + p_0 = 0 \tag{12}$$

Extending Eq. (12) with at least four neighboring events, an overdetermined system of equations is obtained, which is solved through linear least-squares. After an initial fit, the Euclidean distance of each event to the plane is computed. All events for which the distance exceeds a threshold d_{max}, are rejected from this fit. Using the remaining events, a new least-squares plane fit is computed. In [16], this process is repeated until the change in $\mathbf{\Pi}$ is no longer significant. This is the case if the norm of the change in all components in $\mathbf{\Pi}$ is smaller than a second threshold k_d, i.e. $\|\mathbf{\Pi}(i) - \mathbf{\Pi}(i-1)\| < k_d$. In practice, the latter often occurs already after one or two iterations. In this work, the values for d_{max} and k_d specified in [21] are applied, which are 0.01 and 1e-5 respectively.

The final plane is preserved and used to compute the local gradients of Σ_e:

$$\nabla\Sigma_e(x, y) = \left[\frac{\partial\Sigma_e}{\partial x}, \frac{\partial\Sigma_e}{\partial y}\right]^T = \left[-\frac{p_x}{p_t}, -\frac{p_y}{p_t}\right]^T \tag{13}$$

In [16] the gradient components of Σ_e are assumed to be inversely related to the optical flow components (u, v):

$$\nabla\Sigma_e(x, y) = \left[\frac{1}{u(x, y)}, \frac{1}{v(x, y)}\right]^T \tag{14}$$

However, as is also noted in [18], [21], Eq. (14) is subject to singularities when computing u and v. If either component of $\nabla\Sigma_e$ tends to zero, the corresponding optical flow component grows to infinity, which is incorrect. For example, consider a horizontally moving vertical line. Along the y-direction, temporal differences between the resulting events are in this case very small. Therefore, $\frac{\partial\Sigma_e}{\partial y}$ is also small, which leads to a high value of the vertical component v, even though the line is moving horizontally.

In recent work, two modifications to the previously discussed methodology have been proposed. First, in [18] an approach is presented that is robust to singularities in u and v, which led to significant accuracy improvements in the comparison in [21]. In this approach, an orthogonality constraint is imposed on the plane's normal vector $[p_x,\, p_y,\, p_t]^t$, the optical flow vector $[u,\, v,\, 1]$ and the orientation $[l_x,\, l_y,\, 0]$ of the edge in homogeneous coordinates. This constraint leads to a new expression of the optical flow components u and v in terms of the plane's normal vector:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\|\nabla\Sigma_e\|^2}\nabla\Sigma_e = -\frac{p_t}{{p_x}^2 + {p_y}^2}\begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{15}$$

Second, in the implementation in [21] not all events within the space-time window are considered. For each pixel location, only the most recent event is used for computing optical flow. However, high contrast edges tend to produce multiple events in quick succession at a single pixel. Hence, the most recent event at a pixel occurs slightly later than the first event caused by such an edge, which leads to over-estimation of its speed. It may also lead to optical flow estimates in the opposite direction of the edge motion. To prevent this, a refractory period Δt_R (typically 0.1 s) is applied. Events that occur within Δt_R are neither processed nor preserved to support future events.

The discussed algorithm with the previously proposed modifications forms our baseline algorithm. In the following, methods are proposed to increase its efficiency and range of application.

B. *Efficiency Improvements*

In order to enable faster computation and scale the algorithm towards low-end processing hardware, we propose two modifications.

The first modification is to reduce the number of parameters of the local plane. We reduce Eq. (12) by introducing the new parameters p_x^*, p_y^*, p_0^*:

$$p_x^* = \frac{p_x}{p_t},\, p_y^* = \frac{p_y}{p_t},\, p_0^* = \frac{p_0}{p_t} \tag{16}$$

hence obtaining the nonhomogeneous, three-parameter form of Eq. (12):

$$p_x^* x_i + p_y^* y_i + p_0^* = -t_i \tag{17}$$

A second reduction is performed by assuming that the new event $\mathbf{e}_n$ intersects with the plane, which enables the definition of relative coordinates for neighbor events as follows. Given $\mathbf{e}_n$ and a previously identified neighbor event $\mathbf{e}_i$, the relative coordinates of the neighbor event are defined as $\delta x_i = x_i - x_n,\ \delta y_i = y_i - y_n,\ \delta t_i = t_i - t_n$. Substituting these coordinates into Eq. (17) and rearranging terms, the following relation is obtained:

$$p_x^* \delta x_i + p_y^* \delta y_i + \delta t_i = -p_x^* x_n - p_y^* y_n - p_0^* - t_n \tag{18}$$

By substituting Eq. (17) (for which we set $i = n$ to enforce that e_n intersects with the plane) into Eq. (18), the right-hand side of the latter equation reduces to zero. Thus, the final plane $\mathbf{\Pi}^* = [p_x^*,\, p_y^*]$ is described by two parameters, the slopes:

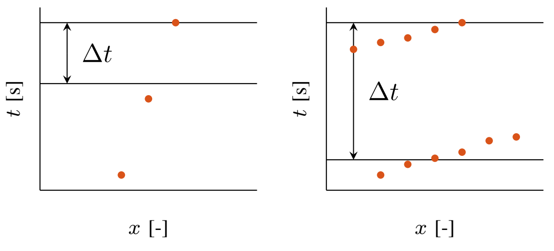
$$p_x^* \delta x_i + p_y^* \delta y_i = -\delta t_i \tag{19}$$

This reduced approach requires significantly less computational effort than the baseline. While solving a homogeneous least squares system is generally performed using a Singular Value Decomposition (SVD), more efficient solvers such as the commonly used QR-decomposition are applicable to nonhomogeneous problems. With a system of M events and N parameters, the computational complexity of the SVD scales with $O(MN^2 + N^3)$. In comparison, the complexity of the QR decomposition scales with $O(MN^2 - N^3/3)$ [57]. Hence, a four-parameter SVD solution has a cost that is approximately proportional to $16M + 64$, which compares to $4M - 8/3$ for a two-parameter QR-decomposition. Note that this is only a rough indication of the true complexity, but it suffices for illustrating the efficiency gain of the parameter reduction. Only a slight reduction in accuracy is introduced with this simplification.

The second modification consists of capping the rate at which optical flow vectors are identified, denoted as the output rate ρ_F. Depending on the computational resources available, input events can be processed at a limited rate to maintain real-time performance. In addition, since the approach assumes that for each individual event, motion needs to be estimated, neighboring events produce highly similar optical flow vectors, making the information increasingly redundant with increasing ρ_F. Therefore, capping this value also prevents unnecessary computational load on follow-up processes. To achieve this, we keep track of the timestamp t_f of the event for which the latest optical flow vector was identified. If a new event $\mathbf{e}_n$ occurs, optical flow is only estimated if $t_n - t_f > 1/\rho_{F_{max}}$, where $\rho_{F_{max}}$ denotes the output rate limit. The event is, however, still stored to support future events, taking into account the refractory period. Hence, accuracy of the estimates that are still performed, is unaffected. The resulting effect of the value of $\rho_{F_{max}}$ on computational performance is explored in Section IV-D3.

C. *Timestamp-Based Clustering of Recent Events*

The baseline algorithm incorporates a fixed setting for the time window Δt to collect recent events. There are two main drawbacks of using a fixed time window, which are illustrated in Fig. 4 for two simple one-dimensional cases. First, Δt defines the lower limit for the magnitude of observable optical flow. For slower motion, the time difference between neighboring events increases. If this difference is too large, all neighboring events fall outside the time window (as illustrated in Fig. 4a), such that the motion cannot be observed. Second, a larger time window can result in the inclusion of unrelated events. For example, in Fig. 4b events are shown which clearly belong to separate features. Still, a part of the outdated features falls within the time window, which leads to an inaccurate fit. In some cases outlier rejection may prevent this, but with tightly packed features this may still cause a failed estimate.



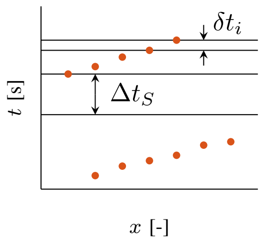


Fig. 4. Examples with one-dimensional ($x - t$) event structures representing motion, in which a fixed time window for collecting events leads to failed motion estimates. In (a), the time window is too small for being able to perceive the slow motion that triggers the events. On the other hand, in (b) events from two sequential fast-moving features enter the same time window. The bottom image (c) illustrates the proposed clustering approach, in which the time difference between the current event and the next most recent event δt_i defines the maximum time difference $\Delta t_S = k_S \delta t_i$. From the leftmost event, the time difference to the next most recent event exceeds Δt_S, such that all bottom events are rejected.

A fixed time window therefore imposes a fundamental trade-off between minimal observable speed and feature density. Since MAVs tend to move at a wide range of velocities, from hovering to fast maneuvers, the capability of observing both fast and slow motion is desirable.

To accomplish this, we propose a very simple clustering method based on the time order of events, which is illustrated in Fig. 4c for a one-dimensional motion case. First, the minimum number of most recent events $\mathbf{e}_i$ for observing velocity is found. In the one-dimensional case in Fig. 4c, only one point is necessary for constructing a line; for two-dimensional image motion, two linearly independent points $(\delta x_i, \delta y_i, \delta t_i)$ are required in order to construct a plane. From the point with the largest δt_i, the relative timestamp defines a maximum time increment Δt_S between the timestamps of consecutive events. To provide a margin for noise, δt_i is scaled with a factor k_S (which has a value of 3 in our experiments), such that $\Delta t_S = -\delta t_i k_S$ (since δt_i should be negative). Second, we iterate through the remaining recent events, ordered by decreasing value of δt_i. If the time difference between two consecutive events $\mathbf{e}_i$ and $\mathbf{e}_{i-1}$ exceeds Δt_S, $\mathbf{e}_{i-1}$ and all events that occurred before it are assumed to belong to different features, and are rejected.

Note that this approach does not take spatial location into account, except for finding the first linearly independent events. Therefore, a variation on the baseline process of outlier rejection is still applied, which is independent of time-scale. Instead of rejection based on a distance threshold, the overall fit quality is assessed through the Normalized Root Mean Square Error ($NMRSE$), defined here as follows:

$$NRMSE = \frac{n}{\sum_{i=1}^{n} \delta t_i} \sqrt{\frac{\sum_{i=1}^{n} \left(\delta t_i - p_x^* \delta x_i - p_y^* \delta y_i\right)^2}{n}} \tag{20}$$

Then, while $NRMSE > NRMSE_{max}$, only the event having the maximum distance to $\mathbf{\Pi}^*$ is rejected. This is repeated until a maximum number of n_R events are rejected. If n_R is exceeded, the estimated plane is rejected and no optical flow is computed. Suitable values for attaining a high number of successful estimates, without sacrificing significant quality, are empirically set at $NRMSE_{max} = 0.3$ and $n_R = 2$.

Still, incorrect optical flow estimates may be detected, either due to noise in the event stream or due to undesired inclusion of outlier events. Depending on the application, certain optical flow magnitudes can be deemed unrealistic in advance. Optical flow estimates are therefore rejected if their magnitude exceeds a threshold V_{max}, which is set to 1000 pixels/s. In addition, a minimum number of events n_{min} must be found through the clustering mechanism in order to have sufficient support for a reliable fit. This number is set to 8 events. Further, a maximal time window of $\Delta t = 2$ s is maintained such that unnecessary event checking is prevented. Note, however, that this time window can now be much larger than in the baseline approach.

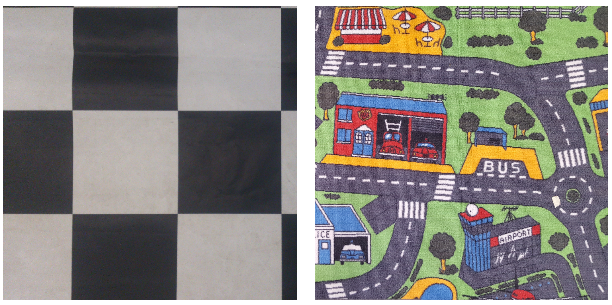
D. Evaluation

To evaluate optical flow estimation performance, several datasets were recorded in which the DVS was moved by hand, facing towards a ground surface covered with a textured mat. The recordings are performed indoors, using an Optitrack motion tracking system to measure ground truth position and orientation of the DVS. From these measurements, ground truth optical flow vectors are obtained using the relations derived in Section III. This is performed for each event for which optical flow is identified, hence providing the means for quantitative accuracy evaluation. Although currently datasets are already available [20], [21], the recorded set specifically represents motion above a flat surface in indoor lighting conditions, i.e. the environment in which flight tests are performed in Section VI.

Images of the recorded ground surfaces are shown in Fig. 5. The checkerboard in Fig. 5a provides high contrast and clear edges and is hence relatively simple for optical flow estimation. The roadmap texture in Fig. 5b has largely unstructured features and lower contrast. It is used to show that our approach extends to more general situations as well. Eight short sequences were selected to evaluate the performance of the proposed method. Each sequence is approximately 1.0 s long and consists of event and pose measurements in which one primary motion type is present. Five sets are selected for the checkerboard; one for vertical translational image motion ($\vartheta_y \approx 1.0$), one for rotational motion ($r \approx -1.3$) rad/s, and three sets with diverging motion of different speeds ($\vartheta_z \approx$

$\{0.2,\ 0.5,\ 2.0\}$). For the roadmap texture, three sets with diverging motion were selected as well ($\vartheta_z \approx \{0.1,\ 0.5,\ 1.0\}$).

In the following analysis, the cap on ρ_F is not applied (i.e. $\rho_{F_{max}} = \infty$), except for the assessment of computational complexity in Section IV-D3.



(a) Checkerboard (b) Roadmap

Fig. 5. Ground surface textures used during the experiments.

1) Qualitative evaluation: Fig. 6 shows optical flow vectors (yellow arrows) estimated using the improved algorithm during three of the selected sequences, along with ground truth flow vectors (blue arrows). Note that, for clarity, the time window for visualizing events is larger than for visualizing optical flow, which is why for some event locations, it appears that no optical flow estimates are found. Accurate normal flow estimates are visible for the checkerboard datasets. In Fig. 6a optical flow is generally constant along the checkerboard edges, matching well to the normal component of the ground truth vectors. The rotating checkerboard sequence (Fig. 6b) also provides accurate optical flow estimates. Clear variation of the normal flow magnitude along the lines is seen.

Last, in Fig. 6c it is clearly visible that the roadmap texture is more challenging. Event structures are less coherent and the visible features are more noisy. Optical flow vectors are sparsely present, yet the available estimates are sufficient for observing the global motion. At some location with noisy features the motion tends to be underestimated, but the majority of the estimates is very similar to the normal direction of the ground truth motion.

2) Quantitative evaluation: For quantitative evaluation, a comparison was made of the proposed optical flow algorithm and the baseline algorithm by [16] (as detailed in Section IV-A). In this comparison the fixed time window Δt for the original approach is set to 100 ms, and the rejection distance d_{max} is set to 0.001 to obtain a similar event density in both algorithms. For a consistent comparison, the baseline algorithm incorporate the same refractory period Δt_R, maximum speed limit V_{max}, and minimum number of events n_{min} as the proposed algorithm.

For benchmarking optical flow accuracy, several error metrics have been introduced such as the endpoint error and angular error [43]. These metrics have been incorporated into recent event-based optical flow benchmarks as well [21]. However, they are defined for optical flow that is fully determinable and is not subject to the aperture problem. The algorithms in this section both estimate normal flow. In [17] a version of the endpoint error is applied that indicates the magnitude error of the normal flow with respect to the *projection* of ground truth optical flow along the normal flow vector. This metric is employed here as well. For each optical flow vector, we compute the Projection Endpoint Error (PEE), which is defined as follows:

$$\mathrm{PEE} = \left| \|\mathbf{V}\| - \frac{\mathbf{V}}{\|\mathbf{V}\|} \cdot \mathbf{V}_{GT} \right| \tag{21}$$

where $\mathbf{V} = [u,\ v]^T$ is the normal flow estimate and $\mathbf{V}_{GT}$ denotes the ground truth optical flow vector.

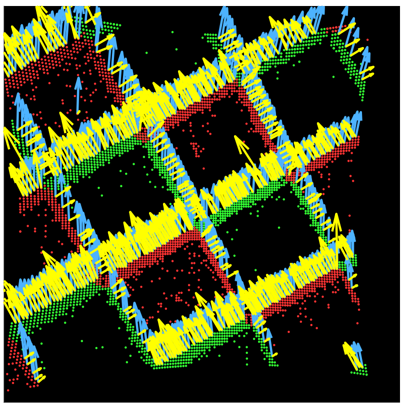
A comparison of the resulting mean absolute PEE values, and their standard deviations, is presented in Table I. The optical flow density is also shown (abbreviated here as η) which indicates the percentage of events for which an optical flow estimate was found. A high value of η indicates that more motion information can be obtained with a given event input.

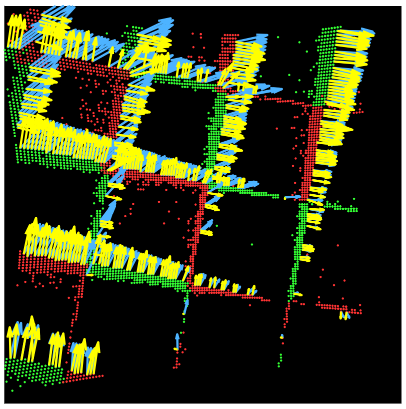
TABLE I
PROJECTION ENDPOINT ERROR (MEAN ABSOLUTE ERROR AND STANDARD DEVIATION) AND DENSITY RESULTS OF THE BASELINE PLANE FITTING ALGORITHM, AND THE NEW ALGORITHM PROPOSED IN THIS WORK. VALUES HIGHLIGHTED IN BOLD ARE THE LOWEST PEE OR THE HIGHEST DENSITY RESULT OF BOTH ALGORITHMS.

	Baseline [16]		This work	
	PEE [pix/s]	η [%]	PEE [pix/s]	η [%]
Checkerboard, $\vartheta_y = 1.0$	18.6 ± 22.9	**50.6**	**17.7** ± 18.7	45
Checkerboard, $r = -1.3$	26.8 ± 28.1	**50.7**	**26** ± 24.7	48.2
Checkerboard, $\vartheta_z = 0.2$	**7.78** ± 12.1	12.6	7.81 ± 8.84	**18.8**
Checkerboard, $\vartheta_z = 0.5$	13 ± 17.9	29.5	**12.7** ± 13.9	**30.7**
Checkerboard, $\vartheta_z = 2.0$	42.4 ± 58.4	**57.5**	**36.3** ± 32.6	56.3
Roadmap, $\vartheta_z = 0.1$	**7.85** ± 6.91	3.54	9.73 ± 8.41	**8.93**
Roadmap, $\vartheta_z = 0.5$	**13.5** ± 13.7	8.44	13.6 ± 12.9	**14.3**
Roadmap, $\vartheta_z = 1.0$	26.4 ± 30.3	13.3	**19.6** ± 19.6	**16.8**

Overall, the results are very similar. Both algorithms reach good scores on the checkerboard sets with translation, rotation, and medium divergence ($\vartheta_z = 0.5$). However, some differences are observable. The proposed algorithm tends to reach a higher optical flow density in the slow divergence ($\vartheta_z = 0.2$) checkerboard scene and in all roadmap scenes, since the baseline algorithm fails to perceive slowly moving features. Still, this does not degrade the estimate accuracy with respect to the baseline algorithm, or only to a limited extent. Note also that in both fast diverging sequences (Checkerboard, $\vartheta_z = 2.0$, and Roadmap, $\vartheta_z = 1.0$) a lower mean absolute PEE is achieved with our approach.

3) Computational Performance Evaluation: An assessment of computational complexity is made using two datasets, one for both texture types. Both sets have a duration of 12 s and contain approximately 40k events per second. This enables quantifying the potential of $\rho_{F_{max}}$ to regulate processing time, as well as the effect of texture. The algorithm is implemented in C and interfaced with MATLAB through MEX, running single-threaded on a Windows 10 64bit laptop with an Intel


(a) Translating checkerboard


(b) Rotating checkerboard

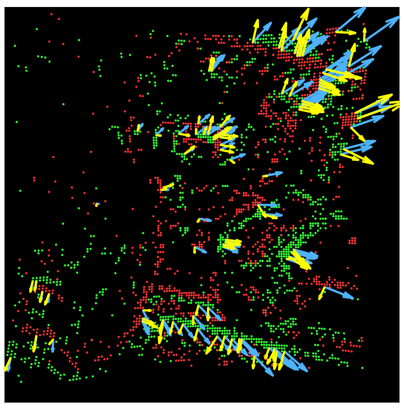

(c) Diverging roadmap ($\vartheta_z = 1.0$)

Fig. 6. Optical flow estimated in several sequences, shown as yellow arrows. The accompanying blue arrows show the ground truth optical flow. Events are shown as green dots (positive polarity) or red dots (negative polarity). The time window for displaying optical flow in each sequence is 10 ms. To better visualize the event input, a larger window of 50 ms is applied for the events.

Core i7 Q720 quadcore CPU. Each dataset and setting of $\rho_{F_{max}}$ is processed ten times for consistent results. The CPU usage of MATLAB during the test was around 12%.

The resulting computation time per event for several settings of $\rho_{F_{max}}$ is shown in Fig. 7, in comparison with the maximal computation time with no control of $\rho_{F_{max}}$. For both textures it is clearly possible to regulate processing time by $\rho_{F_{max}}$. A lower limit appears to be present, which is due to the remaining overhead related to event timestamp copying. Interestingly, there is a clear influence of texture. This difference is due to higher contrast edges in the checkerboard texture, at which several successive events are generated per pixel. Therefore, the refractory period filter rejects these duplicate events before optical flow computation.

Without the refractory period, computational effort is similar for both textures. In this case, the maximal computation time per event, i.e. without control of ρ_F, is 2.11 µs. This is equivalent to processing 470k events per second in real-time, which is easily sufficient for processing realistic scenes on the test machine. Event sequences recorded for post-processing contained event peaks below 150k events per second. Nevertheless, if hardware capabilities are more restricted (e.g. in on-board applications), control of ρ_F can be applied to scale the computational complexity of the algorithm down if necessary.

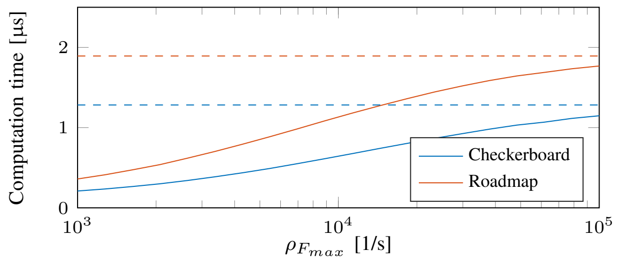


Fig. 7. Processing time per event for checkerboard and roadmap datasets, for different settings of $\rho_{F_{max}}$. The dashed lines indicate the computation times when no limit is applied to $\rho_{F_{max}}$.

V. Estimation of Visual Observables from Event-Based Optical Flow

This section describes our approach for estimating visual observables from event-based optical flow. While optic flow estimation is performed asynchronously, most existing control systems still operate on a periodic basis. Similarly, the proposed algorithm aims to update the estimates of visual observables at a fixed rate. For each periodic iteration, all newly detected optical flow vectors between the current iteration and the previous one form a planar optical flow field, of which the parameters are estimated.

The algorithm is based on two components. First, newly detected optical flow vectors are grouped per direction and incorporated into a weighted least-squares estimator for the visual observables, as discussed in Section V-A. To enable preservation of flow field information over subsequent periodic iterations, a recursive update technique is introduced in Section V-B. In addition, a confidence value is computed and applied to filter the visual observable estimates, as is described in Section V-C. The estimator is evaluated in combination with our event-based optical flow algorithm in Section V-D.

A. Directional Flow Field Parameter Estimation

The presented approach is based on techniques introduced in [48] and used in [7], [29], in which fully defined optical flow estimates are available. Since our optical flow algorithm provides normal flow output, a regular optical flow field representation as in Eq. (10) leads to inaccurate parameter estimates. However, in planar flow fields, normal flow may already provide sufficient information for computing the visual observables. Along the direction of the flow vector, normal flow does provide accurate information.

An example diverging flow field with both optical flow and normal flow is sketched in Fig. 8. Note that the normal flow in some cases deviates significantly from the optical flow equivalent, which leads to significant errors when computing the flow field parameters. However, when grouped by direction (which is done in Fig. 8 through the arrow colors), the normal flow vectors indeed show the original pattern of divergence. This idea is central to the proposed directional flow fields approach.

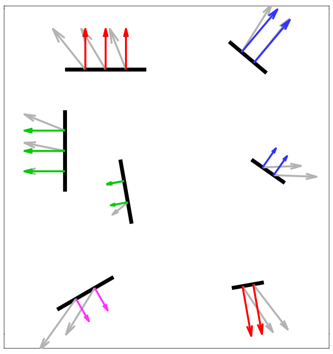


Fig. 8. Example of a diverging flow field resulting from several randomly oriented moving edges. The grey vectors indicate the true flow field, while the colored vectors show the normal flow along the edge orientation. Each color indicates a group of normal flow vectors with similar direction.

In order to observe flow field divergence along a normal flow direction, at least two separate normal flow vectors are required, whose positions are sufficiently apart. For example, in Fig. 8 the purple group of normal flow vectors does not, by itself, provide sufficient information for perceiving divergence. Also, if the flow vectors are located in close proximity, errors in normal flow magnitude have a larger influence. In Fig. 8 the green group is more sensitive to these errors than the red group, since the edges are located closely together. Grouping per direction enables assessment of the reliability of the flow field in each direction, taking the previous issues into account.

A set of m directions $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$ is defined, where $\alpha_1 = 0$ and $\alpha_i - \alpha_{i-1} = \pi/m$. In this work, $m = 6$ directions are used. For each newly available flow vector, we first determine the closest match of α_i to the flow direction α_f. Each direction α_i accommodates both flow in similar and opposite direction, i.e. when $-\pi < \alpha_f < 0$, a match is computed for $\alpha_f + \pi$.

Along the selected direction α_i, the projected normal flow position S and magnitude V are computed, hence obtaining a one-dimensional representation of the flow along α_i:

$$\begin{bmatrix} S \\ V \end{bmatrix} = \begin{bmatrix} \hat{x} & \hat{y} \\ \hat{u} & \hat{v} \end{bmatrix} \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \end{bmatrix} \tag{22}$$

Subsequently, it is corrected for rotational motion by subtracting the normal component of the rotational flow:

$$\begin{aligned} V_T = V &- \cos\alpha_i \left(p - \hat{y}r - q\hat{x}\hat{y} + p\hat{x}^2\right) \\ &+ \sin\alpha_i \left(q - \hat{x}r - p\hat{x}\hat{y} + q\hat{y}^2\right) \end{aligned} \tag{23}$$

For each direction, a one-dimensional flow field is maintained. From Eq. (10) and Eq. (22), the flow field in a single direction is expressed as:

$$V_T = -\vartheta_x \cos\alpha_i - \vartheta_y \sin\alpha_i + \vartheta_z S \tag{24}$$

To solve Eq. (24) for the visual observables, a weighted least-squares solution is computed using the flow vectors from all directions. Let $\mathrm{c}_\alpha = \cos\alpha$ and $\mathrm{s}_\alpha = \sin\alpha$. The overdetermined system to be solved is composed as follows:

$$\begin{bmatrix} -\mathrm{c}_{\alpha_1} & -\mathrm{s}_{\alpha_1} & S_{1,1} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_1} & -\mathrm{s}_{\alpha_1} & S_{1,n_1} \\ -\mathrm{c}_{\alpha_2} & -\mathrm{s}_{\alpha_2} & S_{2,1} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_2} & -\mathrm{s}_{\alpha_2} & S_{1,n_2} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_m} & -\mathrm{s}_{\alpha_m} & S_{m,n_m} \end{bmatrix} \begin{bmatrix} \vartheta_x \\ \vartheta_y \\ \vartheta_z \end{bmatrix} \approx \begin{bmatrix} V_{1,1} \\ \vdots \\ V_{1,n_1} \\ V_{2,1} \\ \vdots \\ V_{2,n_2} \\ \vdots \\ V_{m,n_m} \end{bmatrix} \tag{25}$$

which has the form $\mathbf{A\Theta} \approx \mathbf{y}$. The weighted least-squares solution is then obtained from the normal equations:

$$\mathbf{A}^T\mathbf{WA\Theta} = \mathbf{A}^T\mathbf{Wy} \tag{26}$$

in which $\mathbf{W}$ a diagonal matrix composed of the weights per direction:

$$\mathbf{W} = \mathrm{diag}\Big(W_1, \cdots, W_1, W_2, \cdots, W_2, \cdots, W_m\Big) \tag{27}$$

The weight W_i is used to represent the reliability of normal flow along a direction i based on the spread of S_i along that direction. Its value is determined by the variance $\mathrm{Var}\{S_i\}$. We let W_i scale linearly with $\mathrm{Var}\{S_i\}$, up to a maximum of $\mathrm{Var}\{S\}_{min}$:

$$W_i = \begin{cases} 0 & \mathrm{Var}\{S_i\} = 0 \\ \frac{\mathrm{Var}\{S_i\}}{\mathrm{Var}\{S\}_{min}} & 0 < \mathrm{Var}\{S_i\} \leq \mathrm{Var}\{S\}_{min} \\ 1 & \mathrm{Var}\{S_i\} > \mathrm{Var}\{S\}_{min} \end{cases} \tag{28}$$

The minimum variance $\mathrm{Var}\{S\}_{min}$ is set to 600 pixels^2.

Note also that, through the formulation of Eq. (25), directions with more normal flow estimates have a larger influence on $\mathbf{\Theta}$. Hence, directions for which more information is available, contribute more to the solution.

B. Recursive Updating of the Flow Field

The solution to Eq. (26) for $\mathbf{\Theta}$ provides the estimate for the visual observables. However, depending on the sampling rate of the estimator, it is possible that, during a single periodic iteration, too few normal flow estimates are available for an accurate fit. This leads to noise peaks in the measurement of $\mathbf{\Theta}$, especially during low speed motion. To limit this effect, the matrices $\mathbf{A}$ and $\mathbf{y}$ are not completely renewed at each iteration. Instead, rows from previous iterations are retained and assigned an exponentially decreasing weight, similar to an exponential moving average filter.

For an efficient implementation of the former, $\mathbf{A}$ and $\mathbf{y}$ are not explicitly composed as shown in Eq. (25). Instead, our approach operates on the normal equations in Eq. (26). For each direction independently, we recursively update parts of the matrices $\mathbf{B} = \mathbf{A}^T\mathbf{WA}$ and $\mathbf{C} = \mathbf{A}^T\mathbf{Wy}$. These matrices are composed by the following elements:

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{21} & b_{22} & b_{32} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \tag{29}$$

From Eq. (25), it can be shown that the elements of $\mathbf{B}$ are expressed as:

$$\begin{aligned} b_{11} &= \sum_{i=1}^{m} W_i n_i (\mathrm{c}_{\alpha_i})^2, & b_{21} &= \sum_{i=1}^{m} W_i n_i \mathrm{c}_{\alpha_i} \mathrm{s}_{\alpha_i} \\ b_{22} &= \sum_{i=1}^{m} W_i n_i (\mathrm{s}_{\alpha_i})^2, & b_{31} &= \sum_{i=1}^{m} W_i \mathrm{c}_{\alpha_i} \sum_{j=1}^{n_i} S_{i,j} \\ b_{33} &= \sum_{i=1}^{m} W_i \sum_{j=1}^{n_i} S_{i,j}^2, & b_{32} &= \sum_{i=1}^{m} W_i \mathrm{s}_{\alpha_i} \sum_{j=1}^{n_i} S_{i,j} \end{aligned} \tag{30}$$

and those of $\mathbf{C}$ are expressed as:

$$\begin{aligned} c_1 &= \sum_{i=1}^{m} W_i \mathrm{c}_{\alpha_i} \sum_{j=1}^{n_i} V_{i,j} \\ c_2 &= \sum_{i=1}^{m} W_i \mathrm{s}_{\alpha_i} \sum_{j=1}^{n_i} V_{i,j} \\ c_3 &= \sum_{i=1}^{m} W_i \sum_{j=1}^{n_i} S_{i,j} V_{i,j} \end{aligned} \tag{31}$$

We introduce a shorthand notation $\Sigma_S^i = \sum_{j=1}^{n_i} S_{i,j}$ to represent the sums, cross-product sums, and sums of squares of S and V for direction i. The unweighted contribution of the associated flow vectors is then contained in n_i and the sums Σ_S^i, $\Sigma_{S^2}^i$, Σ_V^i, and Σ_{SV}^i. These values are further referred to as the *flow field statistics*. Hence, a newly detected flow vector is included in the flow field estimate by incrementing these quantities according to the values S and V of the new vector.

What makes this decomposition interesting, is that the flow field statistics form a compact summary of the flow field, independent of the actual number of flow vectors. Thus, flow field information from a previous iteration can be efficiently included in subsequent ones, without increasing the size of the system in Eq. (25). Now, at the start of each iteration, it is possible to include information from the flow field of the previous iteration, simply by preserving a fraction F of the previous flow field statistics. Hence, the estimator accuracy is less dependent on the sampling rate of the algorithm.

The preservation process is illustrated using the statistic Σ_V^i. At the start of iteration k, Σ_V^i is initialized as $\Sigma_V^i(k) = F\Sigma_V^i(k-1)$. During iteration k, Σ_V^i is then updated using newly available normal flow vectors that are allocated to direction i. Hence, the complete update for Σ_V^i is performed as follows:

$$\Sigma_V^i(k) = F\Sigma_V^i(k-1) + \sum_{j=1}^{n_i} V_{i,j} \tag{32}$$

The value of F is computed as:

$$F = 1 - \frac{t(k) - t(k-1)}{k_f} \tag{33}$$

where the time constant k_f is assigned a value of 0.02 s. This step is similar for all statistics. When all newly available vectors are categorized and processed, the flow field is recomputed using Eq. (26).

C. Confidence Estimation and Filtering

In visual sensing, the reliability of motion estimates varies greatly depending on the environment. Factors such as visible texture and scene illumination have an effect on the estimate. With event-based sensing, motion in the scene is another key factor.

Therefore, a confidence value is computed based on several characteristics of the flow field, in order to quantify the reliability of the estimate. This confidence value is defined as a product of three individual confidence metrics based on the following statistical quantities:

- The flow estimation rate ρ_F.
- The maximal variance $\mathrm{Var}\{S\}$ of all flow directions.
- The coefficient of determination R^2 of the solution to Eq. (26), applied here as a nondimensional measure of the fit quality.

R^2 is generally computed through the following [58]:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{34}$$

In this work, the Residual Sum of Squares (RSS) and Total Sum of Squares (TSS) are computed in weighted form as follows:

$$\begin{aligned} RSS &= \mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{\Theta}^T\mathbf{A}^T\mathbf{W}\mathbf{y} \\ TSS &= \mathbf{y}^T\mathbf{W}\mathbf{y} - \frac{\left(\sum_{i=1}^{m} W\Sigma_V^i\right)^2}{\sum_{i=1}^{m} W n_i} \end{aligned} \tag{35}$$

For each indicator, a confidence value k is computed ranging from 0 to 1 (higher is better), similar to the variance weight in Eq. (28). The individual confidence values are thus dependent on settings for R^2_{min}, $\mathrm{Var}\{S\}_{min}$, and $\rho_{F_{min}}$ (not to be confused with $\rho_{F_{max}}$). The values of R^2_{min} and $\rho_{F_{min}}$ are set to 1.0 and 1500 respectively. Note that, since Eq. (28) already provides individual confidence values per direction in the form of W, we simply let $k_{\mathrm{Var}\{S\}} = \max(W_i : i = 1, \dots, m)$.

The total confidence value K is then the product of k_{ρ_F}, $k_{\mathrm{Var}\{S\}}$, and k_{R^2}. Hence, each individual confidence factor needs to be close to 1 in order to obtain a high K. For example, when ρ_F and R^2 are very large, but the flow is very localized (the maximal value for $\mathrm{Var}\{S\}$ is small), the estimate is still not reliable. In this case, it is likely that a single visual feature causes the normal flow, which is insufficient for computing the visual observables.

The confidence K is useful to monitor the estimate quality of the visual observables during flight. In addition, it is the main component of a *confidence filter* for $\mathbf{\Theta}$. This filter is based on a conventional infinite impulse response low-pass filter, in which K is multiplied with the filter's update constant. The final estimate for the visual observables $\hat{\mathbf{\Theta}}$ is determined through the following update equation at iteration k:

$$\hat{\mathbf{\Theta}}(k) = \hat{\mathbf{\Theta}}(k-1) + \left(\mathbf{\Theta}(k) - \hat{\mathbf{\Theta}}(k-1)\right) K \frac{t(k) - t(k-1)}{k_t} \tag{36}$$

where k_t is the time constant of the low-pass filter, which is set to 0.02 s. Lastly, a saturation limit is applied that caps the magnitude of the update of each individual value in $\mathbf{\Theta}$ to $\Delta\vartheta_{max}$ in order to reject significant outliers. The value for $\Delta\vartheta_{max}$ is set to 0.3.

D. Results

For evaluating the accuracy of the presented visual observable estimator, we use the measurements generated for evaluating optical flow performance in Section IV-D, which are generated through handheld motion. Optitrack position measurements provide the ground truth estimates for ϑ_x, ϑ_y, and ϑ_z. For each set, normal flow estimates are computed using the C-based implementation discussed in Section IV-D. The flow detection rate cap $\rho_{F_{max}}$ is set to 2500 flow vectors per second and the periodic estimator samples the visual observables at 100 Hz, similar to the on-board implementation in Section VI.

In our experiments the main variable of interest is ϑ_z, as it forms the basis for the constant divergence controller. Therefore, this variable is investigated over a wide range of velocities. However, the estimates of ϑ_x and ϑ_y are also interesting to assess, since a more elaborate optical flow based controller may also include the horizontal components for hover stabilization. The latter process does require the MAV to perform rolling and pitching motion, inducing rotational normal flow. Therefore, the effectiveness of derotation is evaluated as well.

1) Vertical Motion: For assessment of ϑ_z estimates, vertical oscillating motion was performed above both texture types. The vertical speed of these oscillations was gradually increased, hence covering a wide range of divergence values. This enables a first-order characterization of the estimator behavior.

Fig. 9 shows the resulting estimates compared to ground truth measurements, accompanied by height measurements $h = -Z_{\mathcal{W}}$. Detail sections are shown for low and high divergence motion, for which also the confidence value is shown.

The detail plots show that the estimator is relatively sensitive to local outliers in normal flow at low speeds. In addition, the confidence K is generally low due to lower detection rates of optical flow and low value of R^2. At higher speeds, the errors are relatively smaller. Note that K is also generally higher there. Somewhat lower confidence values are seen for the roadmap texture.

Around sign changes, brief moments are present where the confidence value K is low. The result of this is that, due to the confidence filter, the update of $\hat{\vartheta}_z$ at these points is limited, which leads to a local delay with respect to the ground truth. However, when higher confidence estimates are available, the estimate quickly converges back to the ground truth value.

Based on the estimator results in Fig. 9 we can assess how the error varies with the ground truth divergence. Fig. 10 shows the variation of the absolute error $\varepsilon_{\vartheta_z} = |\hat{\vartheta}_z - \vartheta_z|$ with the magnitude of ϑ_z. A quadratic model $\varepsilon = p_0 + p_1\vartheta_z + p_2\vartheta_z^2$ is fitted to the points, which is represented by the blue line. The values of p_0, p_1, and p_2 are shown in Table II. The errors of both the checkerboard set and the roadmap set are combined, since the estimator shows roughly the same error distribution for both cases. Interestingly, the largest individual errors appear to be present at low divergence. This results from the local delay occurring around zero-crossings in Fig. 9. Note, however, that the error increase with the magnitude of $|\vartheta_z|$ is limited, which enables application of the presented pipeline to a wide range of velocities.

In [31] an extensive characterization of two frame-based visual estimators for ϑ_z is performed, which includes an assessment of their absolute error distribution up to $\vartheta_z \approx 1.3$ (Fig. 10 in the paper). For a first-order comparison, Fig. 9 also shows the quadratic error fit obtained for the frame-based 'size divergence' estimator, which performed best in [31]. Compared to the presented event-based estimator, the size divergence estimator achieves slightly lower errors in the region of $\vartheta_z < 0.5$. However, for faster motion, the error is lower for our event-based estimator. Note that our quadratic model is based on relatively little measurements, and does not yet provide a full characterization.

TABLE II
PARAMETERS OF THE QUADRATIC FIT MODELS IN FIG. 10.

	This work	Size divergence [31]
p_0	0.0761	0.0455
p_1	-0.0016	-0.0043
p_2	0.0288	0.1841

2) Horizontal Motion: Estimation performance for the components ϑ_x and ϑ_y is assessed through a dataset consisting of primarily horizontal motion. In the following set, the DVS is moved in a circular pattern above a checkerboard surface at approximately 0.8 m height.

The resulting visual observable estimates are shown in Fig. 11. For completeness, the values of ϑ_z are displayed as well. Overall, the horizontal movement is captured well in the estimates, although some disturbances are still clearly present, for example around $t = 23$ s. The deviations are comparable to those seen in the vertical motion dataset. A summary of the error values is presented in Table III.

TABLE III
MEAN AND STANDARD DEVIATION OF ABSOLUTE ERRORS FOR THE ESTIMATES DURING HORIZONTAL MOTION, SHOWN IN FIG. 11.

	Mean abs. error [1/s]	Standard deviation [1/s]
ϑ_x	0.094224	0.074771
ϑ_y	0.081472	0.068432
ϑ_z	0.059899	0.047099

3) Effect of Derotation: In order to assess how well normal flow can be derotated with the current setup, measurements

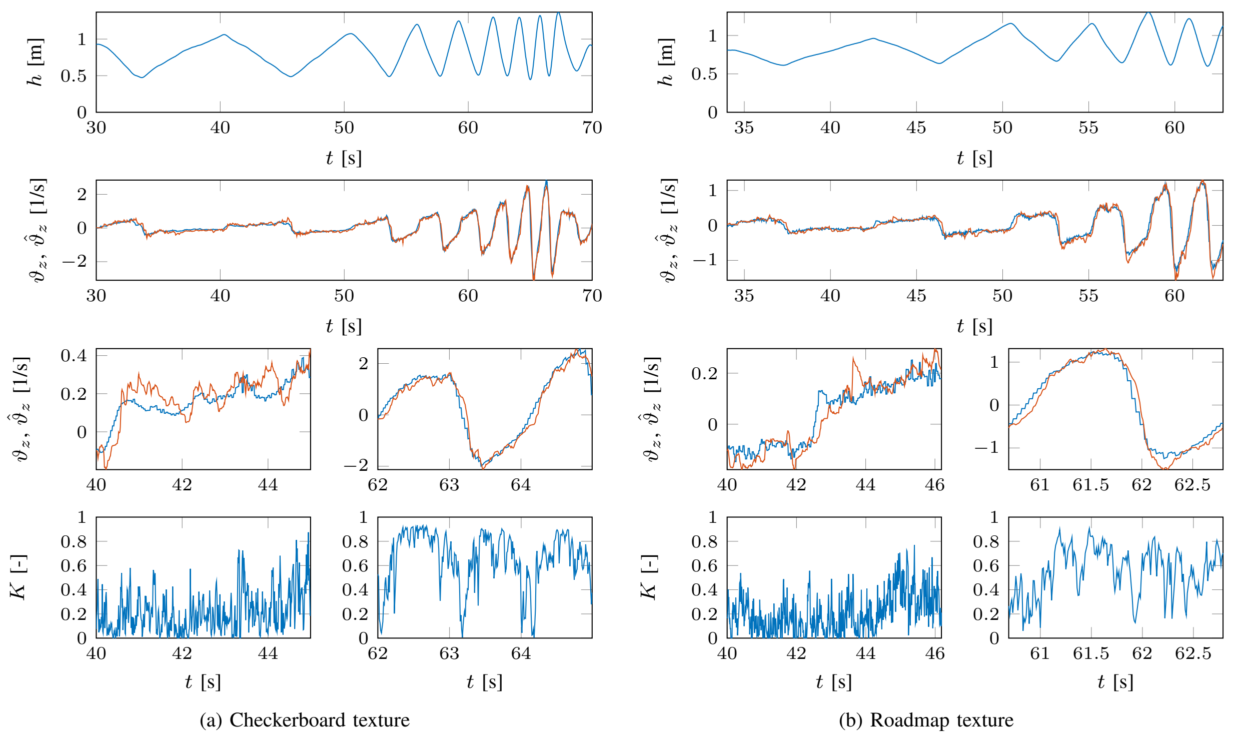


Fig. 9. From top to bottom: Height measurements (top row) and estimates of ϑ_z (second row, red line) in comparison to ground truth measurements (blue line). The two bottom rows show detail sections of ϑ_z estimates (third row) at low speed and high speed, as well as the accompanying estimate confidence value K (bottom row). Measurements are shown for (a) checkerboard and (b) roadmap textures separately.

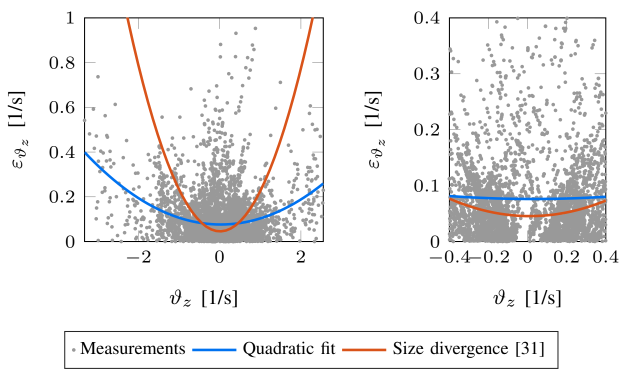


Fig. 10. Left: absolute error distribution for the estimates of ϑ_z shown in Fig. 9. The measurements of the checkerboard and roadmap datasets are combined here. The red line shows a quadratic fit of the error. For comparison, the model obtained for the frame-based size divergence estimator [31] is shown as well. Right: detail view of the error distribution around $\vartheta_z = 0$.

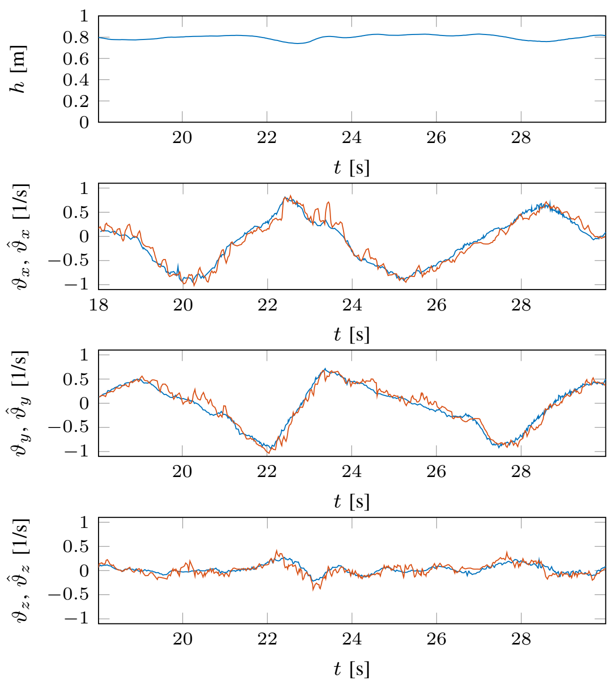


Fig. 11. Height and visual observable measurements (blue) and estimates (red) during horizontal motion above checkerboard texture.

were conducted in which the DVS performed pure rotation along all axes. We compare body rate measurements, ground truth values for ϑ_x and ϑ_y, and estimates with and without derotation. Fig. 12 shows these quantities obtained in three separate sequences, in which each body rate is varied independently.

The most relevant influences are those of p on ϑ_x and q on ϑ_y, which are shown in the top and middle graphs respectively. The influence of r is less profound. For conciseness, only

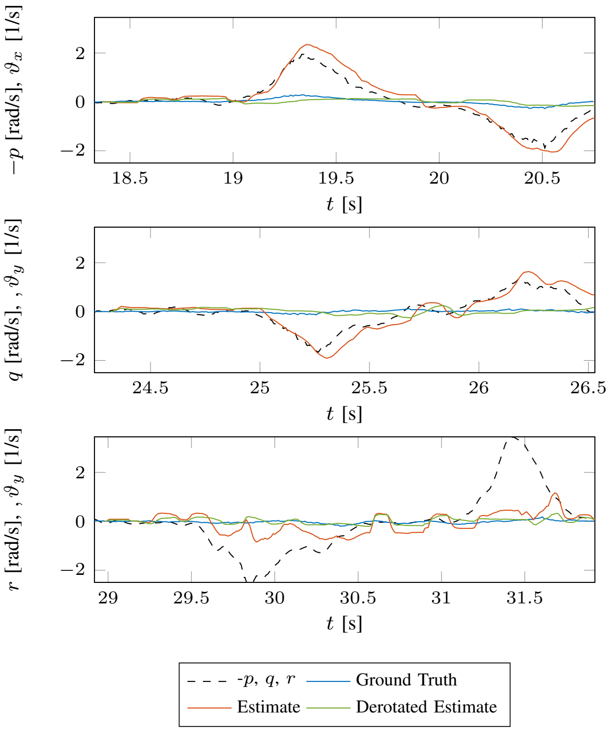


Fig. 12. Baseline and derotated estimates of ϑ_x, ϑ_y compared to ground truth measurements and body rates p, q, and r. Note that the sign of p is inverted to match ϑ_x.

the effect of r on ϑ_y is shown, which provided the clearest result. Some residual motion in p and q is present in the latter case, though it does not fully account for the deviation seen in Fig. 12. Note that the derotation process generally performs well; the largest part of the rotational flow is successfully corrected in the derotated estimate.

VI. Constant Divergence Landing Experiments

This section presents experimental results of constant divergence landings with the presented algorithms in the control loop. In Section VI-A the divergence control law is defined, after which the experimental setup is detailed in Section VI-B. Results from the experiments are presented and discussed in Section VI-C.

A. Divergence Controller

The control law regulates ϑ_z through the vertical thrust T. The controller applies a thrust difference ΔT with respect to a nominal hover thrust T_0, such that $T = T_0 + \Delta T$. A simple proportional control law is applied to ΔT based on ϑ_z, similar to [8]:

$$\Delta T = k_P \left(\vartheta_{z_r} - \vartheta_z\right) \tag{37}$$

The nominal hover thrust T_0 counteracts the weight of the test vehicle. Its value is adapted in-flight in the height control loop of the test vehicle's autopilot software. Before the start of each landing, the vehicle first performs automatic hover to obtain a stable estimate for T_0. During the subsequent landing maneuver its value is kept constant.

B. Experimental Setup

The flying platform used in this work is a customized quadrotor referred to as the MavTec. Its main component is a Lisa/M board, which features a 72MHz 32bit ARM microprocessor as well as a pressure sensor and 3-axis rate gyros, accelerometers, and magnetometers. The Lisa/M runs the open-source autopilot software Paparazzi[1], which handles the control of the drone. The DVS is mounted at the bottom of the MavTec facing downwards, aligned according to the reference frame definitions of $\mathcal{C}$ and $\mathcal{B}$ in Section III. Experiments are performed indoors, using an Optitrack motion tracking system to measure ground truth position and attitude.

In addition, an Odroid XU4 board is mounted on the quadrotor, which processes the event output of the DVS. It features a Samsung Exynos 5422 octacore CPU (four cores at 2.1 GHz and four at 1.5 GHz). The Odroid receives the events from the DVS through a USB 2.0 connection and processes these through the C-based open-source software cAER [59].

An overview of the experimental setup is shown in Fig. 13, including an overview of the on-board processing workflow in Fig. 13c. The estimation pipeline is subdivided in two stages. First, raw events are transmitted from the DVS to the Odroid through a USB interface. In cAER, optical flow is computed from the events using an implementation of our optical flow algorithm. Any event for which flow is estimated, is transmitted to the Lisa/M board through a serial UART interface. This process is completely event-based and is performed in a single thread. Separate threads handle event reception and transmission through the USB and UART interfaces.

Second, in Paparazzi, a periodic follow-up processing thread runs at 100 Hz. At each iteration, all newly received optical flow events are collected and corrected for the quadrotor's attitude and rotational motion. When all new events are processed, new estimates of the scaled velocities are computed with accompanying confidence values. A separate thread running at 512 Hz performs divergence control using the new update for ϑ_z, as well as horizontal position control and stabilization.

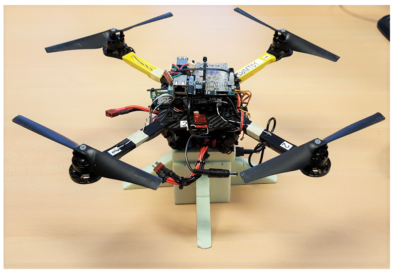
The source code for our versions of cAER and Paparazzi are publicly available online[2].

C. Results

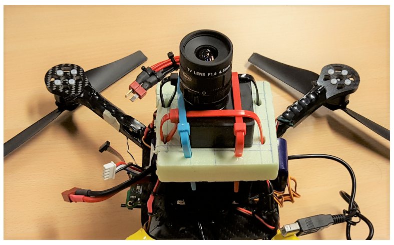
Constant divergence landing maneuvers were performed for several values of the setpoint ϑ_{z_r}. During the tests, the target ground location was covered with the roadmap textured mat shown in Fig. 5b. Currently, no mechanism is implemented to account for instability of constant divergence landings at low height, as described in [8]. Therefore, when significant self-induced oscillations are observed, the landing maneuver is manually terminated.

Resulting flight profiles (height, vertical speed, and divergence) are shown in Fig. 14 for setpoints of $\vartheta_{z_r} = \{0.5, 0.7, 1.0\}$. Note that these values are much higher than the setpoints in comparable frame-based experiments [6], [7]. The estimates for ϑ_z are shown in comparison to the ground truth

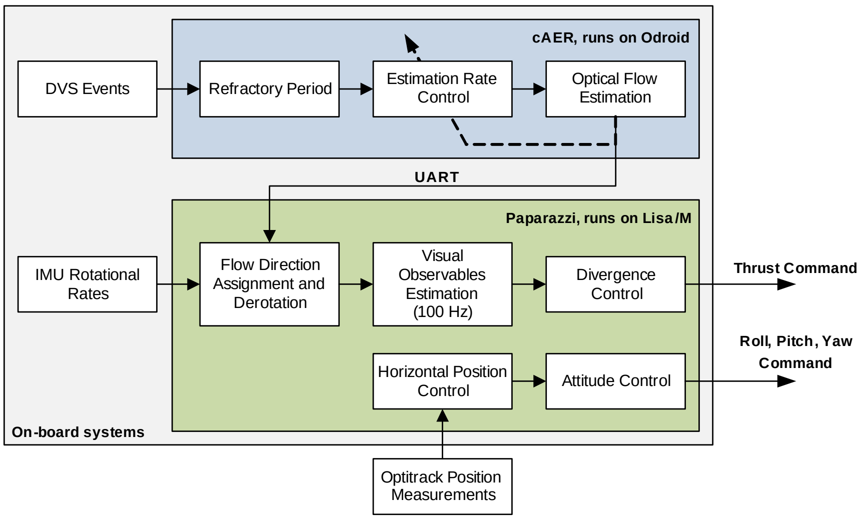
[1]Paparazzi UAV, http://wiki.paparazziuav.org/
[2]cAER: https://github.com/baspijhor/caer/tree/flow_adaptive_final
Paparazzi: https://github.com/baspijhor/paparazzi/tree/event_based_flow



(a) Top view



(b) Bottom view showing the DVS



(c) Overview of the implementation

Fig. 13. Overview of the experimental setup, including pictures of the MavTec. In (a) a top view of the vehicle is shown. The DVS is located at the bottom, protected by a foam cover. In (b) the cover is removed to expose the DVS. In (c) an overview of the processing workflow is shown, indicating the distribution of processes over the Odroid and the Lisa/M processors.

estimate and the corresponding setpoint. For these maneuvers, the proportional gain k_P is set to 0.2. This gain ensures that the descent remains stable during the first part. Decent tracking performance is seen for the lower two setpoints, while at $\vartheta_{z_r} = 1.0$ some overshoot is observed. Still, a faster response may be obtained with an adaptive gain, such as in [31].

The expected instability is also clearly visible. For each setpoint, oscillations with diverging amplitude start to appear when the height is around 0.6 m above the ground, requiring the maneuver to be aborted manually at the moment. Also, a time delay is observed, whose magnitude differs between datasets. By examining the cross-correlation functions of the estimate and ground truth signals, average time delays of 0.05 s, 0.04 s, and 0.10 s are observed for the respective signals. A possible cause for this is the latency in the UART interface between the Odroid and the Lisa/M. Also, part of the delay results from the confidence filter that delays visual observable updates around zero-crossings.

In practice, the visual observable estimator thread running on the Lisa/M microprocessor does not maintain its target frequency of 100 Hz with an optical flow measurement rate $\rho_{F_{max}}$ of 2500 events per second. Instead, it drops to around 75 Hz during the landing maneuvers. However, given the limited processing power of the microprocessor, this is still a decent result. It well exceeds sampling rates seen in recent frame-based optical flow estimation pipelines, which are in the order of 15 to 25 Hz [6], [8], [29], [31]. Also, with a lower setting of $\rho_{F_{max}}$ (around 2000 optical flow events per second), the target frequency of 100 Hz is well attainable. The Odroid can transmit up to approximately 8500 optical flow events per second over the UART connection, limited by the baud rate of 921.6 kilobytes per second.

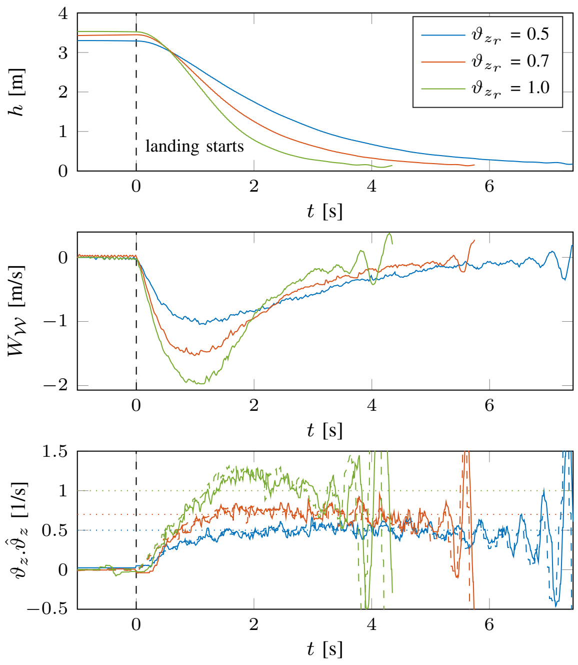


Fig. 14. Height above ground, vertical speed, and divergence measurements with ground truth during a constant divergence landings performed at three different divergence setpoints. In the bottom graph, the dotted, dashed, and solid lines represent the setpoint, ground truth, and estimate for ϑ_z respectively.

For the largest part, the maneuvers are executed successfully, even for high divergence values setpoints. With $\vartheta_{z_r} = 1.0$, the MAV performs a rapid maneuver, descending from a height of 3.5 m to 1 m within 1.79 s. In comparable recent experiments with frame-based cameras for divergence measurement [31], landings were performed up to $\vartheta_{z_r} = 0.3$. Since higher values have not been attempted in these experiments, we cannot know for certain that frame-based optical flow is not applicable to such high speeds.

VII. Conclusion

In this paper we present a successful implementation of event-based optical flow estimation into a constant divergence landing controller for flying robots. Three main contributions lead to this result.

First, a novel algorithm for computing event-based optical flow is derived from an existing local plane fitting technique. The algorithm is capable of estimating normal optical flow with a wide range of magnitudes through timestamp-based clustering of the event cloud. Its performance is evaluated in ground texture scenes recorded by a DVS. Accurate estimates are seen in real event scenes with sparse, high contrast edges, as well as in scenes with densely packed, lower contrast features. Compared to the existing technique, optical flow accuracy is slightly improved for fast motion, while a larger number of succesful optical flow estimates is obtained during slow motion. In addition, it is shown that the optical flow detection rate can be capped to limit computational effort for the algorithm, which enables implementation on low-end platforms without sacrificing accuracy.

Second, we introduce an algorithm for estimating optical flow based visual observables from normal optical flow measurements. By grouping flow vectors by their direction, the aperture problem is circumvented for estimating the parameters of a planar optical flow field. The estimator assesses the reliability of its output through a confidence metric based on the flow estimation rate, the variance of optical flow positions, and the coefficient of determination of the flow field. When coupled to the optical flow algorithm, it is capable of estimating the visual observables accurately over a wide range of speeds. Also, the influence of fast rotational motion on the visual observables is adequately corrected through separate rotational rate measurements.

Third, using the developed pipeline, fast constant divergence landing maneuvers are demonstrated using a quadrotor equipped with a downward facing DVS. Decent tracking performance is achieved for the majority of the descent using a simple proportional controller. The final touchdown of the landing maneuver is not yet performed due to self-induced oscillations close to the ground. However, stability-based control methods have already been demonstrated that can resolve this issue. A future controller based on these methods can, for example, autonomously detect the oscillations and switch to a final touchdown phase based on constant thrust, or perform a complete landing maneuver using an adaptive gains. In addition, our controller does not yet incorporate the visual observables for horizontal stabilization, but relies on an external position tracking system. However, with the estimate accuracy and rotational motion correction presented in this work, this appears feasible.

In a first-order comparison to recent work on landing using frame-based cameras for estimating optical flow, the presented event-based pipeline demonstrates more accurate measurements at high speed and a higher sampling rate, which enable faster maneuvers. However, for a more solid conclusion regarding real-time computational benefits of event-based vision, a comparison should be performed where both frame-based and event-based cameras are incorporated in the same hardware configuration.

While miniature frame-based bottom cameras are readily embedded into several commercially available quadrotors, the on-board hardware configuration in this work is still relatively bulky and inefficient. The implementation in this work is performed on a relatively large MAV in order to carry the weight of the DVS with separate computers for processing events and estimation of visual observables. However, with the availability of smaller and lighter event-based cameras, such as the 2.2 mg meDVS, embedded implementations on smaller vehicles are possible, enabling the full potential of the sensor's low latency.

References

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[2] C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. De Croon, "Autonomous flight of a 20-gram Flapping Wing MAV with a 4-gram onboard stereo vision system," in *Proceedings - 2014 IEEE International Conference on Robotics and Automation*, 2014, pp. 4982–4987.

[3] J. J. Gibson, *The ecological approach to visual perception*. Boston: Houghton Mifflin, 1979.

[4] E. Baird, N. Boeddeker, M. R. Ibbotson, and M. V. Srinivasan, "A universal strategy for visually guided landing," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, no. 46, pp. 18 686–18 691, 2013.

[5] B. Herisse, F. X. Russotto, T. Hamel, and R. Mahony, "Hovering flight and vertical landing control of a VTOL Unmanned Aerial Vehicle using optical flow," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 801–806.

[6] B. Herissé, T. Hamel, R. Mahony, and F.-X. Russotto, "Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 77–89, 2012.

[7] H. W. Ho and G. C. H. E. De Croon, "Characterization of Flow Field Divergence for MAVs Vertical Control Landing," in *AIAA Guidance, Navigation, and Control Conference*, 2016, pp. 1–13.

[8] G. C. H. E. De Croon, "Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy," *Bioinspiration & Biomimetics*, vol. 11, no. 1, pp. 1–18, 2016.

[9] C. Posch, T. Serrano-Gotarredona, B. Linares-barranco, and T. Delbrück, "Retinomorphic Event-Based Vision Sensors : Bioinspired Cameras With Spiking Output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014.

[10] F. Ruffier and N. Franceschini, "Optic Flow Regulation in Unsteady Environments: A Tethered MAV Achieves Terrain Following and Targeted Landing Over a Moving Platform," *Journal of Intelligent & Robotic Systems*, vol. 79, pp. 275–293, 2014.

[11] D. Floreano, R. Pericet-Camara, S. Viollet, F. Ruffier, A. Brückner, R. Leitel, W. Buss, M. Menouni, F. Expert, R. Juston, M. K. Dobrzynski, G. L'Eplattenier, F. Recktenwald, H. a. Mallot, and N. Franceschini, "Miniature curved artificial compound eyes." in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, 2013, pp. 9267–72.

[12] M. Yang, S.-C. Liu, and T. Delbruck, "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 9, pp. 2149–2160, 2015.

[13] J. Conradt, R. Berner, M. Cook, and T. Delbruck, "An embedded AER dynamic vision sensor for low-latency pole balancing," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 2009, pp. 780–785.

[14] T. Delbruck and M. Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Frontiers in Neuroscience*, vol. 7, no. 223, pp. 1–7, 2013.

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

[16] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow." *IEEE transactions on neural networks and learning systems*, vol. 25, no. 2, pp. 407–17, 2014.

[17] F. Barranco, C. Fermuller, and Y. Aloimonos, "Contour Motion Estimation for Asynchronous Event-Driven Cameras," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1537–1556, 2014.

[18] T. Brosch, S. Tschechne, and H. Neumann, "On event-based optical flow detection," *Frontiers in Neuroscience*, vol. 9, no. 137, pp. 1–15, 2015.

[19] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous Optical Flow and Intensity Estimation from an Event Camera," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 884–892.

[20] F. Barranco, C. Fermuller, Y. Aloimonos, and T. Delbruck, "A Dataset for Visual Navigation with Neuromorphic Methods," *Frontiers in Neuroscience*, vol. 10, no. February, pp. 1–9, 2016.

[21] B. Ruckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in Neuroscience*, vol. 10, no. 176, 2016.

[22] X. Clady, C. Clercq, S.-H. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, and R. Benosman, "Asynchronous visual event-based time-to-contact." *Frontiers in neuroscience*, vol. 8, no. 9, 2014.

[23] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[24] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004, pp. I—-652.

[25] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO : Fast Semi-Direct Monocular Visual Odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 15–22.

[26] J. Engel, T. Schöps, and D. Cremers, "{LSD-SLAM}: Large-scale direct monocular {SLAM}," in *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2014, pp. 834–849.

[27] F. Expert and F. Ruffier, "Flying over uneven moving terrain based on optic-flow cues without any need for reference frames or accelerometers." *Bioinspiration & biomimetics*, vol. 10, 2015.

[28] M. Srinivasan, S. Zhang, M. Lehrer, and T. Collett, "Honeybee navigation en route to the goal: visual flight control and odometry," *Journal of Experimental Biology*, vol. 199, no. 1, pp. 237–244, 1996.

[29] M. T. Alkowatly, V. M. Becerra, and W. Holderbaum, "Bioinspired Autonomous Visual Vertical Control of a Quadrotor Unmanned Aerial Vehicle," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 2, pp. 249–262, 2015.

[30] D. N. Lee, "A theory of visual control of braking based on information about time-to-collision." *Perception*, vol. 5, no. 4, pp. 437–459, 1976.

[31] H. W. Ho, G. C. H. E. de Croon, E. van Kampen, Q. P. Chu, and M. Mulder, "Adaptive Control Strategy for Constant Optical Flow Divergence Landing," pp. 1–14, 2016. [Online]. Available: http://arxiv.org/abs/1609.06767

[32] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[33] D.-i. D. Cho and T.-j. Lee, "A Review of Bioinspired Vision Sensors and Their Applications," *Sensors and Materials*, vol. 27, no. 6, pp. 447–463, 2015.

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

[35] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240x180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[36] J. Conradt, "On-Board Real-Time Optic-Flow for Miniature Event-Based Vision Sensors," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Zhuhai, China, 2015.

[37] D. Weikersdorfer, R. Hoffmann, and J. Conradt, "Simultaneous localization and mapping for event-based vision systems," in *International Conference on Computer Vision Systems*. Springer-Verlag Berlin Heidelberg, 2013, pp. 133–142.

[38] D. Weikersdorfer, D. B. Adrian, and D. Cremers, "Event-based 3D SLAM with a depth-augmented dynamic vision sensor," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 359–364, 2014.

[39] 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*, 2014, pp. 2761–2768.

[40] H. Kim, "Simultaneous Mosaicing and Tracking with an Event Camera," in *Proceedings of the British Machine Vision Conference 2014*, 2014.

[41] H. Kim, S. Leutenegger, and A. J. Davison, "Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera," in *European Conference on Computer Vision*. Amsterdam: Springer International Publishing, 2016, pp. 349–364.

[42] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, "Low-Latency Visual Odometry using Event-based Feature Tracks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[43] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, 2011.

[44] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[45] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *International Joint Conference on Artificial Intelligence*, vol. 81, 1981, pp. 674–679.

[46] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM Computing Surveys*, vol. 27, no. 3, pp. 433–466, 1995.

[47] E. Rosten, R. Porter, and T. Drummond, "Faster and better: a machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–19, 2008. [Online]. Available: http://arxiv.org/pdf/0810.2434

[48] G. C. H. E. De Croon, H. W. Ho, C. De Wagter, E. Van Kampen, B. Remes, and Q. P. Chu, "Optic-flow based slope estimation for autonomous landing," *International Journal of Micro Air Vehicles*, vol. 5, no. 4, pp. 287–297, 2013.

[49] J.-Y. Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker - Description of the Algorithm," *Intel Corporation, Microprocessor Research Labs*, vol. 5, 2001. [Online]. Available: http://robots.stanford.edu/cs223b04/algo_affine_tracking.pdf

[50] J.-C. Zufferey, A. Beyeler, and D. Floreano, "Autonomous flight at low altitude using light sensors and little computational power," *International Journal of Micro Air Vehicles*, vol. 2, no. 2, pp. 107–117, 2010.

[51] F. Ruffier and N. Franceschini, "Optic flow regulation: The key to aircraft automatic guidance," *Robotics and Autonomous Systems*, vol. 50, no. 4, pp. 177–194, 2005.

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

[53] F. Barranco, C. Fermuller, and Y. Aloimonos, "Bio-inspired Motion Estimation with Event-Driven Sensors," in *Advances in Computational Intelligence*. Springer International Publishing, 2015, pp. 309–321.

[54] H. C. Longuet-Higgins and K. Prazdny, "The interpretation of a moving retinal image." *Proceedings of the Royal Society of London, B: Biological Sciences*, vol. 208, no. 1173, pp. 385–397, 1980.

[55] D. Brown, "Decentering Distortion of Lenses," *Photometric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[56] J.-Y. Bouguet, "Complete Camera Calibration Toolbox for Matlab," 1999. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[57] M. T. Heath, *Scientific computing: an introductory survey*, 2nd ed. New York: McGraw-Hill, 2002.

[58] S. Weisberg, *Applied linear regression*. John Wiley & Sons, 2005.

[59] L. Longinotti, "cAER: A framework for event-based processing on embedded systems," BSc Thesis, University of Zürich, 2014. [Online]. Available: http://sourceforge.net/projects/jaer/files/cAER/

Part II

Literature Review

Chapter 2

Landing Strategies for Micro Air Vehicles using Optical Flow

The concept of optical flow is a means to quantify motion in the images perceived by cameras and biological retinas. Several animals use implicit features of perceived optical flow to control ego-motion, which result in surprisingly simple models of their behavior. These models formed inspiration for the development of basic control systems for MAVs. First, we introduce optical flow concepts through a mathematical model in Section 2-1. Second, we describe the estimation process of optical flow in these applications through conventional cameras and through special sensors for optical flow perception in Section 2-2. Next, we discuss landing strategies seen in biology in Section 2-3 and their applications involving MAVs in Section 2-4.

2-1 Modeling optical flow

To mathematically represent the perception of motion in monocular vision, a model is introduced that relates the motion of points on an image to the corresponding motion of these points in the world. A common approach is to use a perspective projection model of motion, often referred to as the *pinhole camera model*. This model simplifies images appearing on spherical retinas through the assumption that the retina is planar. Hence, the relation between points in the world and their projection on the image plane become simple equations. In applications involving wide-angle viewing, for example using omnidirectional or fisheye cameras, it may be necessary to apply a more advanced projection model, such as a spherical model (Geyer & Daniilidis, 2000), as this assumption will not be valid. However, for the camera used in this work, the pinhole camera model is sufficiently accurate.

2-1-1 Optical flow in the pinhole camera model

The formulation of Longuet-Higgins and Prazdny (1980) is frequently employed (e.g. De Croon et al., 2013; Ho & De Croon, 2016) and is applied in this work to model optical flow.

Figure 2-1 shows a sketch of a point A with coordinates (X, Y, Z) in an observer-fixed reference frame $OXYZ$. The origin corresponds to the nodal point of the observer. The Z-axis is the *optical axis*. A plane normal to the Z-axis is located in front of the observer, representing the retina or focal plane of the observer. The intersection of this plane and the optical axis, point $o = (x_p, y_p)$ is the *principal point*.

In arbitrary ego-motion, the observer reference frame is subject to translational velocities (U, V, W) along, and rotational velocities (p, q, r) around the X-, Y-, and Z-axes respectively. In this situation, the derivatives of the coordinates of point A with respect to the observer are:

$$\begin{aligned} \dot{X} &= -U - qZ + rY \\ \dot{Y} &= -V - rX + pZ \\ \dot{Z} &= -W - pY - qX \end{aligned} \tag{2-1}$$

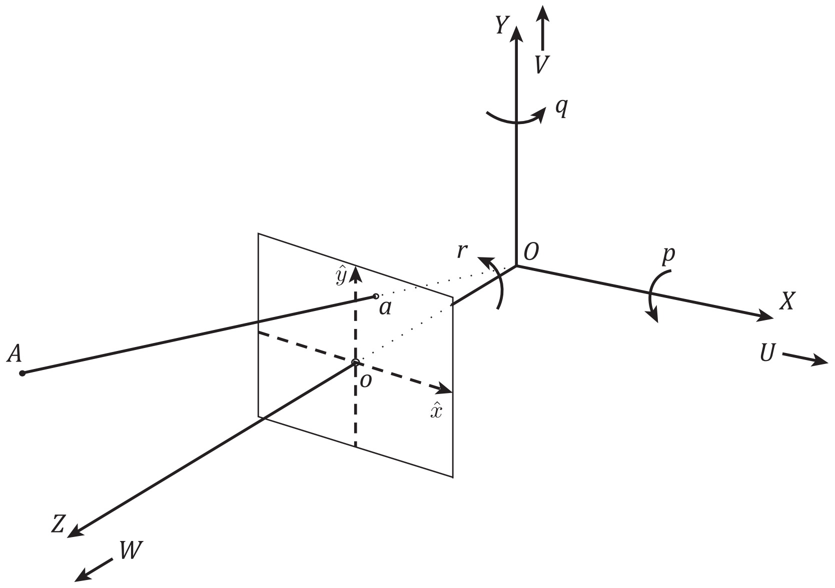


Figure 2-1: Projection of a world point on the image plane in a pinhole camera model. Adapted from Longuet-Higgins and Prazdny (1980).

Let $(\hat{x}, \hat{y}) = (X/Z, Y/Z)$ represent the normalized coordinates of the point projected at an image plane at a distance of 1 m from the origin This point produces observable optical flow with velocity components $(\hat{u}, \hat{v}) = \left(\dot{\hat{x}}, \dot{\hat{y}}\right)$. Now $\hat{u}$ and $\hat{v}$ can be represented in terms of the ego-motion of the observer, and the depth of the point:

$$\begin{aligned} \hat{u} &= \dot{X}/Z - X\dot{Z}/Z^2 &= (-U/Z - q + r\hat{y}) - \hat{x}\left(-W/Z - p\hat{y} + q\hat{x}\right) \\ \hat{v} &= \dot{Y}/Z - Y\dot{Z}/Z^2 &= (-V/Z - r\hat{x} + p) - \hat{y}\left(-W/Z - p\hat{y} + q\hat{x}\right) \end{aligned} \tag{2-2}$$

These equations show that the motion of a point on the image plane consists of one part due to translational motion, and another due to rotations:

$$\begin{aligned} \hat{u}^T &= (-U + \hat{x}W)/Z, \quad & \hat{u}^R &= -q + r\hat{y} + p\hat{x}\hat{y} - q\hat{x}^2 \\ \hat{v}^T &= (-V + \hat{y}W)/Z, \quad & \hat{v}^R &= -r\hat{x} + p + p\hat{y}^2 - q\hat{x}\hat{y} \end{aligned} \tag{2-3}$$

2-1-2 Simplified visual observables derived from optical flow

Assuming that the observer's environment is static, the observer's ego-motion states (p, q, r, U ,V,W) are equal for all world points. The point depth Z varies per point. Hence, if multiple points and their associated optical flow, one could combine optical flow observations for multiple points in order to solve for depths and ego-motion. This is a complex calculation which forms a basis for e.g. structure-from-motion (estimating the world point depths) (Adiv, 1985) and monocular visual odometry (estimating observer motion and position) problems (Nistér, Naroditsky, & Bergen, 2004). Solutions to these problems are computationally demanding and result into low-rate control systems (e.g. Kendoul, Fantoni, & Nonami, 2009).

However, other quantities regarding the observer's motion can be extracted using a simpler approach. These *visual observables* provide non-metric information regarding the relative motion of the observer, that are still useful for navigation tasks. In the case of landing maneuvers, Eq. (2-3) can be simplified through two assumptions, which we discuss in the following.

Derotation

First, if the observer can, through other sensors, obtain information on rotational velocity, the rotational flow components can be corrected for, i.e. the flow can be *derotated.* This correction is often seen in MAV-related applications, since MAVs are commonly equipped with rotational rate sensors as part of an Inertial Measurement Unit (IMU) (e.g. De Croon et al., 2013; Grabe, Bulthoff, Scaramuzza, & Giordano, 2015; Herissé et al., 2012).

With the rotational components left out, some basic motion cues are provided by Eq. (2-3). It is shown by Longuet-Higgins and Prazdny (1980) that, if we define the coordinates $\hat{x}_f = U/W$ and $\hat{y}_f = V/W$, the translational velocities become:

$$\hat{u}^T = (\hat{x} - \hat{x}_f)\, W/Z, \quad \hat{v}^T = (\hat{y} - \hat{y}_f)\, W/Z \tag{2-4}$$

Hence, if there is a translational motion W, a point on the image plane $(\hat{x}_f, \hat{y}_f)$ exists where $\hat{u}$ and $\hat{v}$ are zero independent of the point depth, while their magnitude increases further away from this point. This point is therefore referred to as the *FoE*, or in the case of negative W, *Focus of Contraction (FoC).* This visual cue provides the observer with a sense of motion direction. Further, if this point is known, it is possible to compute for each point the *time-to-contact* $\tau = Z/W$ to the corresponding world point. This quantity provides a non-metric measure of how fast the observer approaches this world point.

Planar flow

Second, if the camera is viewing a perfectly planar surface, a constraint can be imposed on the depth of points, based on their projected position and the orientation of the plane. With this assumption, a deterministic expression for the components of a *planar flow field* can be obtained. De Croon et al. (2013) proposed an expression that directly expresses the flow field in terms of the plane slopes and the observer's normalized velocities. Let Z_0 represent the distance to the plane *along the optical axis of the observer*, and (Z_X, Z_Y) the plane slopes along the observer's X- and Y-axes respectively. Further, let $\vartheta_x = U/Z_0$, $\vartheta_y = V/Z_0$, and $\vartheta_z = W/Z_0$ represent the velocities of the observer, normalized by Z_0. As shown by De Croon et al. (2013), this results in the following expression:

$$\begin{aligned} \hat{u} &= -\vartheta_x + (\vartheta_x Z_X + \vartheta_z)\hat{x} + \vartheta_x Z_Y \hat{y} - Z_X \vartheta_z \hat{x}^2 - Z_Y \vartheta_z \hat{x}\hat{y} \\ \hat{v} &= -\vartheta_y + \vartheta_y Z_X \hat{x} + (\vartheta_y Z_Y + \vartheta_z)\hat{y} - Z_Y \vartheta_z \hat{y}^2 - Z_X \vartheta_z \hat{x}\hat{y} \end{aligned} \tag{2-5}$$

A further simplification can be made if the slopes are negligible. Then Eq. (2-5) reduces to a very simple expression relating normalized velocities to the flow field:

$$\begin{aligned} \hat{u} &= -\vartheta_x + \vartheta_z \hat{x} \\ \hat{v} &= -\vartheta_y + \vartheta_z \hat{y} \end{aligned} \tag{2-6}$$

The normalized velocities $\vartheta_x, \vartheta_y, \vartheta_z$ are valuable sources of information in the visual perception of motion usable for basic navigation. They are the main *visual observables* in this thesis. Similar to the general case, τ can be obtained from ϑ_z, in this case through $\tau = Z_0/W = 1/\vartheta_z$. Alternatively, ϑ_z can be related to the flow field *divergence* D, which is defined as (McCarthy, Barnes, & Mahony, 2008):

$$D(x, y) = \frac{\partial \hat{u}}{\partial \hat{x}}(\hat{x}, \hat{y}) + \frac{\partial \hat{v}}{\partial \hat{y}}(\hat{x}, \hat{y}) \tag{2-7}$$

From Eq. (2-6) is then derived that, in translational motion facing perpendicular to a planar surface, $D = 2\vartheta_z = 2/\tau$.

The remaining normalized velocities ϑ_x, ϑ_y are the opposites of the *ventral flows* ω_x, ω_y of the ground plane, i.e. $\omega_x = -\vartheta_x$, $\omega_y = -\vartheta_y$. The ventral flows quantify the projected velocities of the ground plane on the retina.

2-2 Optical flow measurement

As optical flow is perceived through changes in light perception, measurement of optical flow is possible from sensors that measure brightness. The main type of sensor used for this purpose is a regular frame-based camera that outputs a stream of images. In addition, specialized sensors for optical flow measurement have been developed and used for navigation tasks. We briefly discuss here how these are used and what the limitations of both approaches are.

2-2-1 Frame-based cameras

Charge-Coupled Device (CCD) or Complementary Metal Oxide Semiconductor (CMOS) imaging sensors, i.e. frame-based camera sensors, are widely used and highly versatile. Besides navigation, frame-based cameras enable intelligent visual perception capabilities for MAVs. Computer vision techniques can be employed for object detection, feature tracking, classification, scene reconstruction, and mapping. In autonomous applications for MAVs such capabilities are essential. Also for remote piloted applications, operators usually require visual information from the MAV. For surveillance, exploration, aerial video, and inspection applications, the camera is the main payload of the MAV. Extending their usage to visual navigation is therefore a straightforward option.

On-board cameras for MAVs typically enable optical flow update rates in the range of 15 to 25 Hz (De Croon, 2016; Grabe et al., 2015; Herissé et al., 2012). From these images, optical flow estimation is performed using computer algorithms. The main drawback of optical flow estimation from frame-based cameras is that processing in real-time is computationally intensive, as is seen in real-time implementations. In the following we discuss techniques and issues that are frequently seen in MAV and real-time robotic applications.

Frame-based optical flow techniques

In this work we primarily distinguish between local and global approaches to computing optical flow. In a *local* approach, optical flow is estimated at certain pixel locations incorporating information from neighboring pixels. In contrast, *global* approaches compute dense image flow over a full image.

The local approach of Lucas and Kanade (1981) (further referred to as 'Lucas-Kanade') is the technique mostly used in real-time optical flow applications, where it outperforms other classical methods (McCarthy & Barnes, 2004). Despite its long existence, it is still a leading approach in recent real-time MAV applications (e.g. De Croon, 2016; Grabe et al., 2015; Ho & De Croon, 2016). It result from the *brightness constancy constraint*, which states that total brightness is conserved across sequential frames. This constraint can be expressed in the form of the following partial differential equation:

$$I_x u + I_y v = -I_t \tag{2-8}$$

where I_x, I_y and I_t are the partial derivatives of I, which can be computed from two sequential frames, and (u, v) are the velocity components of an optical flow at the image level, i.e. in pixels per second. As such, however, Eq. (2-8) is insufficient to determine u and v. To solve this, Lucas and Kanade (1981) proposed the assumption that, within the direct neighborhood of a point in the image, u and v are constant. Hence, with $n > 2$ neighboring pixels, an overdetermined system of equations is obtained:

$$\begin{bmatrix} (I_x)_1 & (I_y)_1 \\ \vdots & \vdots \\ (I_x)_n & (I_y)_n \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} (I_t)_1 \\ \vdots \\ (I_t)_n \end{bmatrix} \tag{2-9}$$

From Eq. (2-9), estimates for u and v are obtained through ordinary least-squares.

A global estimation technique seen in early MAV applications (Chahl, Srinivasan, & Zhang, 2004) is based on image interpolation (M. V. Srinivasan, 1994). This technique computes reference images to which predefined transformations are applied, e.g. translation or rotation. Then, through interpolation the most likely transformation is estimated. It is well-suited for computing ventral flows, though not for more complex motion patterns, such as divergence from an unknown FoE.

Aperture problem, normal flow, and corners

A common issue in optical flow estimation is the *aperture problem.* It occurs when there is ambiguity in the observed motion due to a limited field of view (Beauchemin & Barron, 1995). An example is illustrated by Figure 2-2 where a long line moves through a rectangular aperture. Since the endpoints of the line are not observable, only the line motion component normal to the line (the bold vector) can be observed with certainty. It is not possible to observe whether any motion occurs in tangential direction. The bold vector indicates therefore the *normal flow* observable from the line.

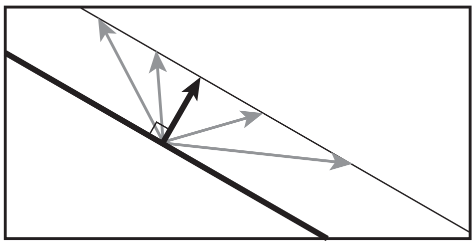


Figure 2-2: Illustration of the aperture problem occurring with an edge moving through the field of view. Only the black flow component can be identified with certainty, while the actual flow may be any of the gray vectors.

This problem is seen frequently in local optical flow algorithms where a limited pixel neighborhood is used, such as the Lucas-Kanade algorithm. Lines or object contours are observed through features with a brightness gradient dominant in one spatial direction, and are referred to as *edges.* Along an edge, only normal flow can be identified. Therefore, most implementations of Lucas-Kanade operate on *corners* (e.g. De Croon, Alazard, & Izzo, 2015; Ho & De Croon, 2016), i.e. image features where a significant brightness gradient is observable in two linearly independent spatial directions. The Lucas-Kanade technique is in this case preceded by a corner detector such as the popular FAST algorithm (Rosten, Porter, & Drummond, 2008).

Evaluating optical flow algorithm performance

In order to evaluate the performance of optical flow methods, benchmarking techniques have been developed. The most notable benchmarks are the framework set up by Barron, Fleet, and Beauchemin (1994) and, more recently developed, the Middlebury benchmark (Baker et al., 2011). In these frameworks optical flow techniques are evaluated based on predefined image sequences for which ground truth flow fields are available.

The Middlebury benchmark is set up such that researchers can download the image datasets and publish their results online[1]. This facilitated ongoing development of new optical flow techniques, and hence, results from over 100 techniques have been published so far (Baker et al., 2011).

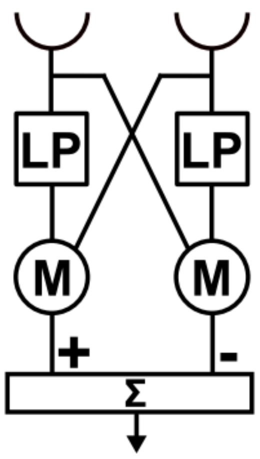
[1]Available at `http://vision.middlebury.edu/flow/eval/`

2-2-2 Insect-inspired sensors

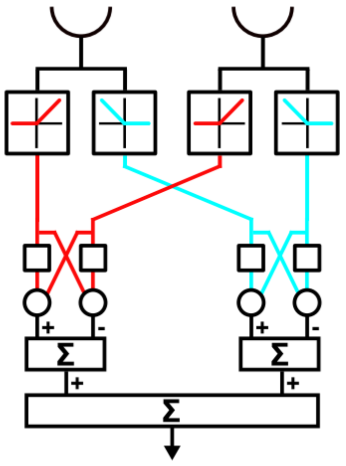
Inspiration for different sensor types was formed by the compound eyes of insects, which are built up from several small photoreceptors called *ommatidia*. As opposed to vertebrate eyes using lenses, compound eyes offer the advantages of a very wide field of view and high temporal resolution (Floreano et al., 2013). Both these advantages are desirable for robotic applications involving motion detection. Efforts have therefore been made to create artificial compound eyes.

Floreano et al. (2013) developed a miniature panoramic curved sensor showing close similarities to these compound eyes. This 1.75 g sensor, the CurvACE, features a 180°×60° field of view, covered with 42×15 separate ommatidia, and a signal acquisition bandwidth of 300 Hz. Its ommatidia adapt their local brightness independently, allowing for a large intrascene dynamic range. However, despite its small size, a relatively high power consumption is required (0.9 W)

The motion detection mechanism used by insects is also interesting for sensor design through its simplicity. It is referred to as an *EMD*, and it detects motion through correlation of the signals of neighboring photoreceptors. A classical model for the EMD is the Reichard detector (Hassenstein & Reichardt, 1956). This model involves multiplication of a receptor signal with the time-delayed signal of a neighbor receptor, see Figure 2-3a. Since its introduction, other variants of this mechanism have been introduced as well. In recent research involving flies (Eichner et al., 2011), most evidence was seen for the mechanism by Franceschini, Riehle, and Le Nestour (1989). The improvement of the Franceschini model is that there is no interaction between signals of opposite sign.



(a) The Reichardt detector model.



(b) The Franceschini detector model.

Figure 2-3: EMD models investigated by Eichner et al. (2011). On top of the images are two photoreceptor cells, from which the signals flow downward. The square blocks (on the left with LP) indicate time delays applied through a low-pass filter. The circular cells (on the left with M) indicate multiplication of two signals. Adapted from Eichner et al. (2011).

EMDs formed the basis for extremely simplified optical flow sensors. In the work by Ruffier and Franceschini (2005) and Ruffier and Franceschini (2014) a microcontroller-based sensor using only two photoreceptors was employed. Despite its simplicity, this sensor was capable of measuring flow at a 1000 Hz sampling rate with decent accuracy, though only in a single spatial direction. A drawback of EMDs-based sensors is that they typically cannot, by themselves, measure patterns of optical flow such as divergence or rotation. However, this can be overcome through integration of data from multiple sensors (Expert & Ruffier, 2015).

2-2-3 Optical mouse sensors

Off-the-shelf optical mouse sensors provide a second simple alternative. They have smaller pixel grids than camera image sensors (typically 32×32 pixels) but can achieve higher update rates (in the order of 1000 Hz) (Chao, Gu, & Napolitano, 2014). These sensors measure two-dimensional translational optical flow. They were seen frequently in MAV applications where translational flow measurements were sufficient, or in arrays of several sensors (Zufferey, Beyeler, & Floreano, 2010; Zufferey & Floreano, 2006). In comparison to EMDs they work well in highly illuminated scenes such as in outdoor conditions, but perform less in indoor applications (Expert, Viollet, & Ruffier, 2011).

2-3 Bio-inspired landing strategies using optical flow

Extensive research has been performed on how animals exploit optical flow for navigation. In this research, the visual observables identified in Section 2-1-2 were related to visually guided maneuvers performed in biology.

Honeybees were seen performing grazing landings based on the velocity of the ground plane as seen on the retina (Chahl et al., 2004; M. Srinivasan, Zhang, Lehrer, & Collett, 1996). Therefore, in a grazing landing, the honeybee lands by keeping *ventral flow constant*. Hence, when the honeybee slows down, height above the surface is decreased accordingly through $\omega_x = -U/Z$. According to Chahl et al. (2004), a linear relationship can also be observed between forward speed and descend speed, i.e.

$$\dot{Z} = cU = -c\omega_x Z \tag{2-10}$$

with c a proportionality constant. In this case, the height during a landing maneuver decays exponentially over time, hence ensuring a smooth landing:

$$Z(t) = Z(t_0)e^{-c\omega_x(t-t_0)} \tag{2-11}$$

Recent research has also shown that honeybees land on vertical surfaces with constant image divergence (Baird et al., 2013), which is equivalent to keeping τ constant. This results in a similar landing behavior seen in grazing landings. However, only the velocity perpendicular to the landing surface is regulated. In this case, $\dot{Z} = Z/\tau$, which is equivalent to Eq. (2-10), and similarly results in exponentially decaying height.

A different strategy was observed in human drivers performing braking maneuvers (Lee, 1976) and pigeons performing landings (Lee, Davies, Green, & Van der Weel, 1993). In these experiments behavior was observed where not τ was kept constant, but its *rate of change* $\dot{\tau}$. The result is that a slightly different landing trajectory is obtained. Let $k = -\dot{\tau}$ be the rate of change setpoint. Then, as derived by Izzo and De Croon (2012), the height decreases as follows:

$$Z(t) = Z(t_0)\left(k\frac{t}{\tau(t_0)} + 1\right)^{1/k} \tag{2-12}$$

Eq. (2-12) shows that, for this strategy, a deterministic point exists where $Z = 0$, depending on the choice of k. The higher its value, the faster is the landing executed. Also note that, when $k = 1$, the trajectory for Z becomes linear with t. For the human driver experiments by Lee (1976) a value for k around 0.5 was observed. Interestingly, this makes Eq. (2-12) quadratic, which is equivalent to applying *constant deceleration* (Alkowatly, Becerra, & Holderbaum, 2015). The ability to tune k make the constant $\dot{\tau}$ a versatile approach where the landing profile can be tailored to the desired application.

2-4 Applications in Micro Air Vehicles

The strategies outlined in Section 2-3 inspired researchers to develop simple reactive control systems for landing. In this section the main applications seen in MAVs are discussed. These include landing control laws as well as related applications such as slope estimation and deriving metric scale.

2-4-1 Navigating using ventral flow

The observations of grazing landings performed by honeybees inspired researchers to apply a constant ventral flow technique to landing tasks. Chahl et al. (2004) implemented this strategy on-board of a small fixed-wing MAV to explore applications in this field. A small camera was used for ventral flow estimation through image interpolation. Optical flow was controlled by the elevator of the MAV, while forward thrust was reduced. Similarly, Green, Oh, and Barrows (2004) applied an optical flow sensor to perform landing and obstacle avoidance on-board of a fixed-wing MAV. During these experiments, a properly tuned ventral flow controller proved feasible for landing these type of aircraft, when tuning the controller properly to stay above stall speed.

The constant ventral flow strategy for MAVs was extensively investigated by Ruffier and Franceschini (2005), with a specific focus on mimicking navigation in insects. The authors demonstrated the constant ventral flow strategy using a tethered rotorcraft equipped with an optical flow sensor moving along a circular trajectory above a textured pattern. Similar experiments were performed for navigating in roofed environments by incorporating dorsal flow (Expert & Ruffier, 2012) and navigating in unsteady environments (Ruffier & Franceschini, 2014). An example test setup is shown in Figure 2-4. In the first work, the rotorcraft pitch angle was used to control the flight condition. A downward pitch angle setpoint resulted in steady forward flight. Landing was performed by slowly pitching up, such that forward speed decreased, and the height was decreased accordingly through the ventral flow controller. Hence, the controller was capable of terrain following and landing by solely keeping ventral flow constant.

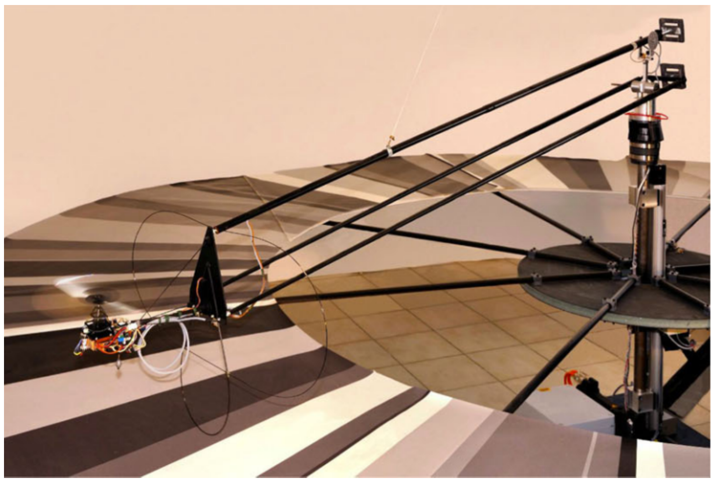


Figure 2-4: The test setup used by Ruffier and Franceschini (2014), with a tethered rotorcraft flying above a circular surface. In this experiment, the surface was moved vertically to simulate an unsteady environment. Adapted from Ruffier and Franceschini (2014).

While during most experiments using this setup the rotorcraft attitude was controlled externally through the tether, Expert and Ruffier (2015) performed similar tethered experiments with a twin-rotor MAV that could regulate its own pitch angle, height, and speed, without the need for inertial sensors or external guidance. The MAV was equipped with two CurvACE sensors mounted facing forward and backward, giving it a full 360° field of view around its pitch axis. Through an active reorientation system, the sensors were rotated independently from the MAV pitch angle to align parallel with the nearest surface. This was done based on differences in ventral or dorsal flow perceived by the front and rear CurvACE sensors.

2-4-2 Constant divergence landing

The limitations of navigating using ventral flow only, are that this requires forward motion, and that no control is possible over vertical dynamics independent from forward motion. While this is not necessarily problematic for landing fixed-wing MAVs, for rotorcraft MAVs pure vertical motion and hovering are commonly desired navigation maneuvers. Divergence cues provide a solution in this case. The simplest approach would be to keep D constant. Ventral flow control can in this case be used to control horizontal motion independently.

Herissé et al. (2012) applied a constant divergence strategy to hovering above and landing on a moving platform. In addition, the authors implemented a control law to force ventral flows to zero, enforcing horizontal control. Control performance was demonstrated successfully in a maneuver consisting of horizontal stabilization above a landing pad, followed by vertical landing. However, response speed was limited due to the latency and low sampling frequency of the outer control loop (15 Hz). A landing maneuver took 25-40 s, and there was significant 'hesitation' before the actual touchdown.

Very recent work by Ho and De Croon (2016) focused on experimentally characterizing constant divergence landing systems in terms of parameters affecting control performance, i.e. noise and delay. Here a typical latency of 0.1 s was obtained using a regular camera, with an algorithm running at 25 Hz. It was seen that noise and mainly time delay introduce significant oscillations and destabilize the control algorithm close to the landing surface. This is seen in Figure 2-5, which shows a simulated landing using the noise and delay model. This is due to the higher sensitivity of divergence measurements to erroneous motion: small errors give rise to the same motion, but since this motion is scaled by Z, disturbances have much more influence at low height.

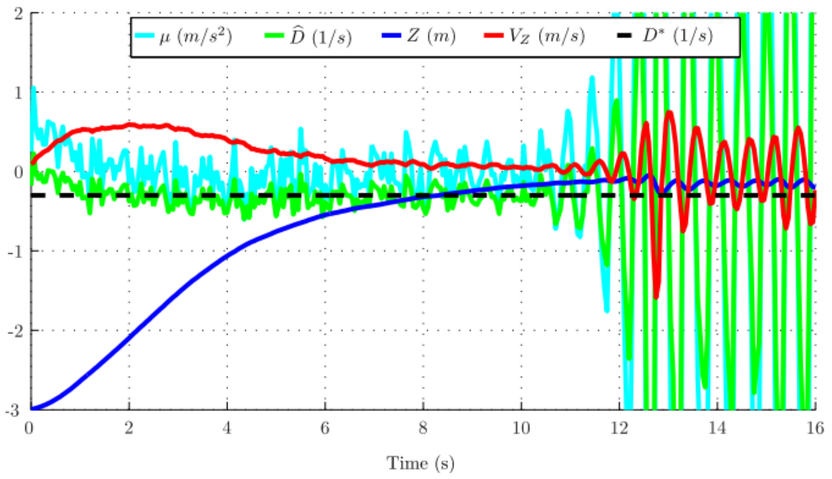


Figure 2-5: Simulated constant divergence landing using the noise and delay model. From Ho and De Croon (2016).

2-4-3 Constant rate-of-change in time-to-contact

Research was also performed into using different τ-based laws. Izzo and De Croon (2012) investigated the concepts of using constantly decreasing τ and exponentially decreasing τ within the context of spacecraft landing. In particular, exponentially decreasing τ is found to provide a mass-optimal landing. Further work included the application of these control laws to a simulated spacecraft descent, estimating optical flow from scaled images (De Croon et al., 2015).

Kendoul (2014) designed and implemented a purely τ-based autopilot, using estimates for τ in three dimensions for control. In this study τ was calculated from fused GPS, IMU, and pressure sensor measurements; future work is proposed for applying vision sensors for this. However, the autopilot performed maneuvers only on the basis of the τ values. Reference τ trajectories in three dimensions were calculated at the start of a maneuver such that it is completed within a given time. Two control laws are proposed for τ tracking: a hybrid linear control law and a nonlinear ratio control law. Extensive testing of this setup was performed on-board of an Ascending Technologies Pelican quadcopter, showing good performance of the controller.

The constant $\dot{\tau}$ approach was implemented on-board of a MAV by Alkowatly et al. (2015). τ and ventral flow were computed by assuming a planar optical flow field structure, and directly estimating its parameters using least squares. Derotation of the optical flow field is performed here using an unscented Kalman filter for sensor fusion.

2-4-4 Estimating the slopes of a landing surface

With Eq. (2-5), a relation was established that defines a planar flow field for surfaces under an arbitrary inclination with respect to the observer. De Croon et al. (2013) employed this relation to estimate the slopes of a landing surface, before deciding to initiate a landing maneuver. An experiment was performed where a quadrotor MAV descended a staircase and landed when the estimated slope was low enough. The MAV successfully landed at the bottom of the staircase. Small errors were still seen in the slope, most likely caused by the pitch attitude of the MAV.

2-4-5 Deriving metric scale and distance during landing maneuvers

Growing evidence suggests that animals require some sense of metric scale for landing. For example, honeybees need it to timely extend their legs before touchdown (Evangelista, Kraft, Dacke, Reinhard, & Srinivasan, 2010). Since the discussed visual observables do not provide any indication of metric scale, additional information is needed. Therefore, recent work has been performed aimed at estimating the metric scale from basic visual observables.

A sensor fusion approach (e.g. Grabe et al., 2015) is interesting for most MAVs, since they are commonly fitted with several sensors that do rely on metric scale, such as an IMU. However, for the smallest MAVs that rely on pure visual sensing (e.g. De Croon, De Clercq, Ruijsink, Remes, & De Wagter, 2009), this approach is not an option. In addition, insects lack accelerometer-like inertial senses (Expert & Ruffier, 2015), yet they still appear capable of sensing a form of metric distance.

Van Breugel, Morgansen, and Dickinson (2014) demonstrated a method for estimating distance during a constant divergence docking maneuver, based on the control input required by the control law. It was shown that, for a constant divergence motion, a direct relation exists between divergence, control input, and distance. Such an approach by itself is highly noise-sensitive due to errors in divergence. However, through recursive least-squares estimation of the distance trajectory parameter (i.e. the initial distance, comparable to $Z(t_0)$ in Eq. (2-11)), accurate estimates for distance were obtained.

A novel approach based on the stability of a constant divergence control law was recently proposed by De Croon (2016). In Section 2-4-2 it was established that a constant divergence controller becomes increasingly unstable closer to the ground. In this work, it was established that the height where self-induced oscillations of the controller start, is in fact dependent on the control gain. Hence, through detecting these oscillations and the controller gain at which they occur, the height can be estimated. This technique has interesting applications, e.g. triggering when to turn off a constant divergence controller and finish a descent with low vertical speed, or to land with adaptive control gain and continuously estimate height.

Chapter 3

Event-based Vision

Applications discussed in the previous chapter involved mainly frame-based cameras and special optical flow sensors. In this chapter we introduce event-based vision by means of the sensor used in this research, the DVS, in Section 3-1. Next, a comparison of the DVS to other event-based sensors is made in Section 3-2. Finally, an overview of current research using the DVS and similar sensors is given in Section 3-3.

3-1 The Dynamic Vision Sensor

The DVS is an event-based vision sensor with a 128×128 pixel array, designed by Lichtsteiner et al. (2008). A picture of the device is shown in Figure 3-1. As opposed to pixels in conventional frame-based image sensors, the pixels of the DVS operate asynchronously. Each pixel individually responds to *changes* in perceived brightness by generating *events*. Therefore, the output of the sensor is a stream of events encoding local brightness changes in the sensor's field of view.

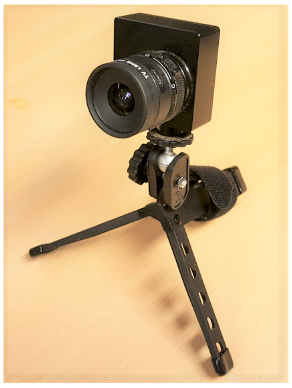


Figure 3-1: Picture of the DVS.

3-1-1 Working principle

DVS events correspond to an increase or decrease with respect to a reference brightness level set at the last occurring event. Specifically, the criterion for generating an event is that the *logarithmic change in brightness* with respect to the last event ΔI in a pixel exceeds a predefined threshold. The result is that brightness changes perceived are encoded in the time differences between consecutive events.

Figure 3-2 illustrates this behavior. The top graph shows a typical voltage output of the photoreceptor V_p, which represents the logarithmic level of brightness perceived. In the bottom graph, V_p is reset to a baseline value once it exceeds either the ON or the OFF threshold, such that the difference voltage V_{diff} is obtained. A reset generates a corresponding ON or OFF event.

Each event $\mathbf{e}$ encodes the timestamp at which a threshold is exceeded at a pixel (t), the pixel location on the array (x, y), and the polarity of the change (P). This polarity has a discrete value of +1 or -1 corresponding to an ON or OFF event respectively. This method of encoding is referred to as an Address-Event Representation (AER) (Lichtsteiner et al., 2008).

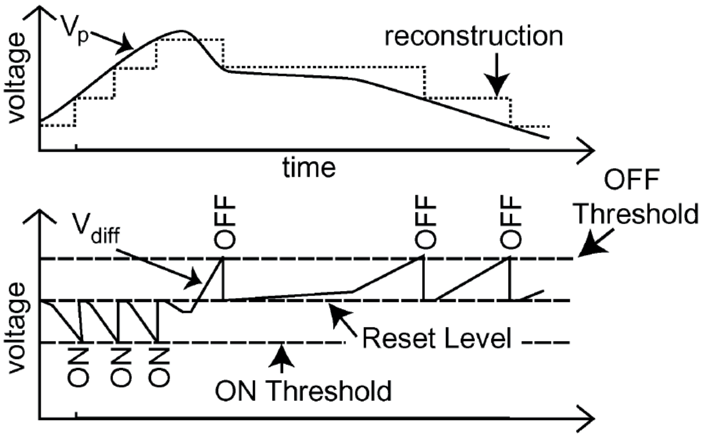


Figure 3-2: Working mechanism of a DVS pixel. From Lichtsteiner et al. (2008).

3-1-2 Sensor characteristics, advantages, and limitations

The DVS is currently available for purchase by the product name 'DVS128'. The sensor is housed in a casing with a standard CS lens. It interfaces with other hardware through a USB 2.0 connection, which is also used to power the device. Table 3-1 shows an overview of the specifications of the DVS as published online (IniLabs, n.d.). The values of latency and power consumption are slightly lower than the values published by Lichtsteiner et al. (2008), possibly due to improvements in the design.

Mainly due to asynchronous pixel readout, the DVS offers distinct advantages compared to frame-based cameras. First, information from the sensor is sparse and contains much less redundant data compared to sets of frames obtained from frame-based sensors. Second, the

[1]Available from Yang et al. (2015)

Table 3-1: DVS specifications (IniLabs, n.d.)

Array size	128×128 pixels
Pixel size	40×40 µm
Dimensions	5×5×2.5 cm (without lens)
Weight	120 g (including lens)
Connectivity	USB 2.0
Latency	12 µs (at 1 klux illumination)
Temporal contrast sensitivity [1]	17%
Power consumption	23 mW
Intrascene dynamic range	120 dB
Event bandwidth	1 million events per second

DVS measures changes at a very high temporal resolution (1 µs) and with small latency (15 µs). Third, the average power consumption is 24 mW. Last, due to the combination of event-based encoding and logarithmic intensity measurement, the sensor achieves a dynamic range of 120 dB.

Currently, the DVS's main limitations are the relatively small number of pixels, and the lack of absolute brightness levels. In addition, temporal contrast sensitivity is still limited. In essence, this quantity represents the minimum event threshold, and hence how much detail in scenes can be captured by the DVS. Also, experiments show that with the standard DVS, the stated latency of 15 µs cannot be achieved due to USB data transfer. In practice, latencies ranging from 125 µs to 4 ms were obtained, depending on the sensor activity (Conradt, Cook, et al., 2009; Delbruck & Lichtsteiner, 2007). Interestingly, higher activity leads to lower latencies. A possible reason is that USB data packets fill up faster when more events are transmitted, and are thus sent more often (Delbruck & Lang, 2013).

3-1-3 Processing software

The open-source tool jAER has been developed (Delbruck, 2007) in order to record and visualize measurements of the DVS. This Java-based program processes the output of sensors using AER encoding, such that it can be recorded and played back at lower speed. It is suitable for a monocular setup as well as for a stereo vision configuration.

Operation settings of the DVS, such as the pixel firing threshold, can be adjusted in order to obtain a desired response. Also, jAER offers the possibility to immediately process data in real-time using *filters*, allowing to use the tool for research purposes and testing. These filters process each event at the moment it is received by jAER. Results from previous work are represented in jAER through these filters, such as control algorithms for a robot goalkeeper (Delbruck & Lichtsteiner, 2007), a pencil balancer (Conradt, Cook, et al., 2009), and a slot car racer (Delbruck et al., 2015). Within the context of this research, it is also interesting to note that filters have been implemented for determining optical flow, as well as optical flow based visual servoing (Delbruck, 2007).

In addition, a version of jAER has been developed based on the C programming language, named cAER. This version is aimed at enhancing portability of the framework to embedded systems, providing more efficient processing, and integration into complex applications (Longinotti, 2014). A modular structure is used, allowing for the definition of filters similar to jAER. Its functionality was validated on multiple systems, including a Raspberry Pi model B, a PandaBoard, and a desktop PC. For interfacing the DVS with the processor on board of a MAV, this is a promising framework.

3-2 Related hardware developments

Several event-based vision sensors have been created before the DVS was available. Most were, however, not practical. Kramer (2002) developed a 48×48 pixel 'optical transient' sensor based on CMOS image sensor technology, where a filtering circuit asynchronously converts measured transients to ON/OFF events. This sensor had problems achieving a low temporal contrast sensitivity as well as leakage currents (Lichtsteiner et al., 2008). Zaghloul and Boahen (2004) developed a 96×60 pixel silicon retina to model spiking output cells of the human retina, which suffered from poor fixed-pattern noise.

The DVS was the first practical, commercially available event-based vision sensor, initiating research on applications of this type of sensor (Cho & Lee, 2015). Since then, several other event-based vision sensors have been designed. Some show various performance improvements with respect to the DVS. Others combine the event-based advantages of the DVS with the availability of absolute brightness values. In addition, smaller versions of the DVS have been developed for lightweight embedded applications.

3-2-1 Performance improvements

Serrano-Gotarredona, Leñero-Bardallo, and Linares-Barranco (2011) developed and tested a 128×128 pixels sensor featuring a smaller pixel size (35×35 µm), a finer temporal contrast sensitivity (10%) and a lower latency (3.6 µs). A pre-amplifying stage has been added to achieve this. However, its power consumption is therefore higher (reaching a maximal value of 231 mW) and dynamic range is limited to 100 dB. An improved design features a lower latency of 3 µs, while also reducing power consumption to 4 mW and temporal contrast sensitivity to 1.5%, maintaining an effective dynamic range of 120 dB. This improvement resulted mainly from a new pixel photo sensing and transimpedance pre-amplifying stage, and allows this sensor to capture motion with more detail (Serrano-Gotarredona & Linares-Barranco, 2013).

Recently, a 60×30 pixel sensor was published, which features a 1% temporal contrast sensitivity, 130 dB dynamic range and in-pixel asynchronous delta modulation to encode events. The latter resulted in better preservation of signal, reducing losses by a factor 3.5 (Yang et al., 2015).

3-2-2 Combination with absolute brightness measurements

A drawback of the DVS and its newer variants is the lack of absolute brightness levels. This mainly limits its application in areas such as loop closure in Visual Simultaneous Localization and Mapping (VSLAM), object recognition, and classification (Cho & Lee, 2015). Therefore, interest has gone out to sensors that combine the benefits of both frame-based and event-based vision.

An alternative that provides a solution to this problem is the Asynchronous Time-based Image Sensor (ATIS) (Posch, Matolin, & Wohlgenannt, 2011). It has a 304×240 pixel array operating asynchronously through the AER communication scheme. In this sensor, each pixel not only transmits events indicating the direction of relative brightness change, but also absolute brightness values. It achieves a slightly higher dynamic range of 125 dB, but requires a power consumption ranging from 50 mW (static) to 175 mW (high activity).

Further development of the DVS resulted into the Dynamic and Active pixel Vision Sensor (DAVIS) (Brandli, Berner, Yang, Liu, & Delbruck, 2014). Essentially, it combines an improved DVS with a conventional global shutter frame-based sensor. Each of the 240×180 pixels contains circuitry to read out individual, asynchronous brightness change events, as well as absolute brightness values in a synchronous manner. The asynchronous capabilities of the sensor have been improved to achieve a 130 dB dynamic range, a latency of 3 µs, and a lower power output of maximally 14 mW. The DAVIS is also equipped with an IMU. Compared to the ATIS, the DAVIS offers advantages in terms of low power consumption and the flexible availability of complete frames desired. On the other hand, the localized nature of absolute brightness measurements by the ATIS may be desirable when complete image data is not necessary, for example in local classification. Both the ATIS and the DAVIS are commercially available and are frequently used in recent research.

Recently, a new version of the DAVIS chip has been reported that is able to distinguish colors (Li et al., 2015). This sensor, nicknamed C-DAVIS, is currently under development and aims to achieve a pixel size slightly larger than those of the DAVIS. The array size is also increased to house 320×240 pixels available for monochrome asynchronous event-based vision. Also, each pixel is aimed to contain 4 subpixels for synchronous image capture, such that RGBA frames of 640×480 pixels can be captured.

3-2-3 Miniaturization

Work has also been done to design and build the DVS in a more compact form, suitable for lightweight, embedded applications. The result is a 23 g version of the camera that has no casing and a smaller lens: the Embedded Dynamic Vision Sensor (eDVS). The current version features a 32-bit 204 MHz microcontroller, 136 kB SRAM, and 1 MB storage to enable on-board processing, as well as an accelerometer (IniLabs, n.d.). These on-board computational resources allow for processing at minimum latency. Additionally, a further miniaturized version, the Miniature embedded Dynamic Vision Sensor (meDVS), has been reported, where the sensor size is narrowed down to just the size of the DVS128 chip (20×20 mm). This sensor has a custom made plastic lens and only a 4 Mbps UART port for communication, hence weighing only 2.2 g (Conradt, 2015). The sensor is shown in comparison to the eDVS and a regular DVS in Figure 3-3.

3-3 Applications of event-based cameras

The introduction of event-based vision opened up new research in the field of computer vision. Recently work was performed to achieve visual tracking, optical flow detection, object recognition and classification, stereo vision, and VSLAM (Cho & Lee, 2015). In this section we

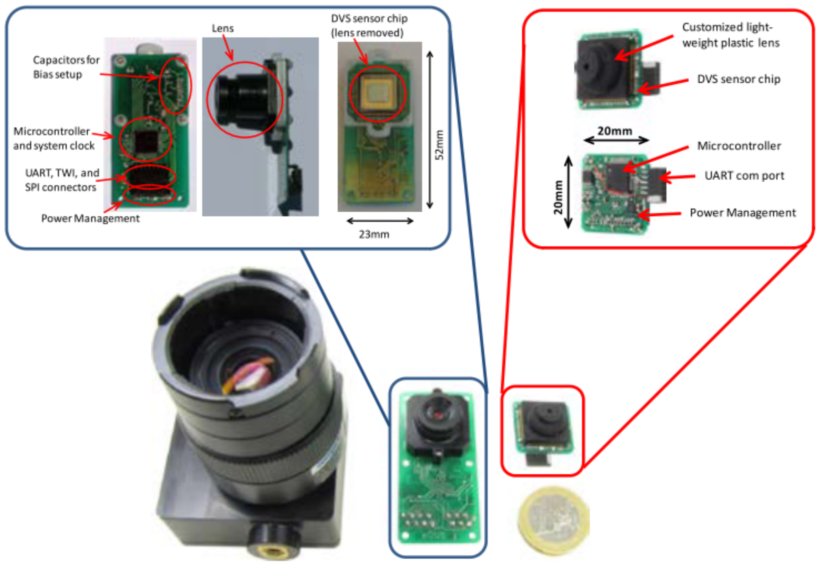

Capacitors for Bias setup
Microcontroller and system clock
UART, TWI, and SPI connectors
Power Management
Lens
DVS sensor chip (lens removed)
52mm
23mm
Customized light-weight plastic lens
DVS sensor chip
20mm
Microcontroller
UART com port
Power Management
20mm

Figure 3-3: Size comparison of the DVS, the eDVS, and the meDVS (left to right). From Conradt (2015).

summarize several past applications, showing the state-of-the-art of research using event-based sensors, the significance of this new sensor, the advantages, and the challenges encountered. We focus on the applications in visual tracking, control systems and visual odometry: fields in which the fast dynamics of the DVS have the biggest impact. Since work on event-based optical flow detection is particularly relevant in this research, this is discussed separately in Chapter 4.

3-3-1 Visual tracking

Litzenberger et al. (2006) developed a cluster tracking algorithm for AER data. In short, this algorithm tries to match incoming events to clusters of previous events, whose positions are tracked and updated using new event positions. It was applied to vehicle tracking on a two-lane highway and for tracking of walking people in a top-down view. This approach was expanded upon by Piatkowska, Belbachir, Schraml, and Gelautz (2012) to address occlusions of multiple clusters through Gaussian mixture models.

Drazen, Lichtsteiner, Häfliger, Delbrück, and Jensen (2011) performed a proof-of-concept study for applying the DVS to particle tracking velocimetry in fluid flows, using an event matching algorithm similar to the methodology used by Litzenberger et al. (2006). Several modifications were introduced to address issues of crossing particles. In this study tracking capabilities of the DVS were compared to those of a standard 1,024 $\times$ 1,024 pixel camera operating at 2,000 frames per second. The study showed that the sparsity of DVS output data is a significant advantage, as its recorded dataset was 840 kB compared to the high-speed camera video of 8 GB size. Also, the tracking algorithm was capable of running faster than real-time on a 3.33-GHz Intel Core i7 Windows 7 desktop computer. However, the resolution of the DVS currently limits the amount of particles that were successfully tracked.

3-3-2 Visual control systems

Delbruck and Lichtsteiner (2007) demonstrated high-speed tracking capabilities of the DVS using a test setup of a 'goalie robot'. A DVS was used to track balls moving towards a goal, using the cluster tracking algorithm by Litzenberger et al. (2006). A servo directed the robot arm in position to block the balls. Processing was performed on board of a 2 GHz Pentium M laptop. Using this setup a latency of 2.8±0.5 ms for determining the servo motor command was achieved. Also, a success rate in the range of 80-90% was estimated for blocking incoming balls with $>$ 150 ms time to impact. The main cause for the achieved latency appears to be the USB 2.0 interface. Later, Delbruck and Lang (2013) added a self-calibration feature.

Another visual control experiment was performed by Conradt, Cook, et al. (2009) where a pencil was balanced on top of an actuated table using a pair of DVSs (see Figure 3-4). The setup was later improved by using eDVSs and dedicated hardware (Conradt, Berner, et al., 2009). This balancing task requires fast state measurements in order to achieve good performance. Tracking of the pencil pose was performed using a Hough transform approach. Each eDVSs Both line estimates were combined to yield a full 3D estimate of the pencil position. A PD controller uses the pose information to control the servos such that the pencil is balanced. This setup achieves position update intervals ranging between 125 and 300 µs, though maximum values of 1.5 ms were also measured when the pencil was not moving rapidly or when a servo command was sent. Pencils and similar objects could be balanced for several minutes.

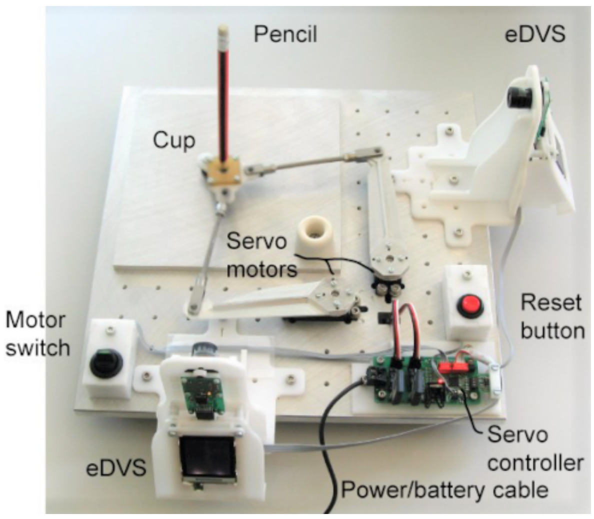


Figure 3-4: The pencil balancer setup used by Conradt, Berner, et al. (2009).

An event-based visual servoing controller was developed by Gil, García, Mateo, and Torres (2014). The authors adapted classical image-based visual servoing to work with events detected by a DVS, without the use of a pattern. Incoming events are clustered in real-time into features that can be tracked. An experiment is discussed with a stationary DVS mounted at a manipulator endpoint tracking a mobile robot moving on the ground. Good tracking performance was obtained, but no description of closed-loop performance (i.e. where the DVS is moved by the manipulator) is given.

3-3-3 Pose estimation

Censi, Strubel, Brandli, Delbruck, and Scaramuzza (2013) performed pose tracking of a MAV during high-speed maneuvers using LED markers blinking at high frequency (> 1 kHz). The DVS was placed on the floor to track infrared LED markers installed on the MAV. During the experiment, the MAV performed flips while facing a textured wall with the on-board camera. Pose estimates from the tracked LED markers were compared to ground truth from an Optitrack system and a standard VSLAM algorithm. While the former performed better on estimating translation, the Euler angles were estimated more accurately by the VSLAM algorithm. The authors mention the resolution of the DVS as a possible cause for this.

Censi and Scaramuzza (2014) also investigated the use of the DVS in combination with a frame-based camera, to perform visual odometry. In an experimental setting their algorithm achieved good performance at estimating rotational motion. However, translational motion estimates were noisy. The authors suggested to separate the use of event-based cameras for rotation estimation, while using conventional cameras for translation.

Mueggler, Huber, and Scaramuzza (2014b) performed a demonstration of a robust pose estimation method using a DVS on-board, by tracking a set of line features. A quadrotor MAV was facing a wall with a black square marker, performing flips and tracking the edges of the marker in order to estimate its own pose. To track the edges, line position estimates were maintained for the marker edges using the Hough transform, similar to the work by Conradt, Cook, et al. (2009). During the demonstration, events were streamed to a laptop that performed computations necessary for pose estimation. It was possible to achieve real-time visualization of the pose estimate with this setup. The MAV reached angular rates of 1,200 $^\circ$/s during the flips. Position was estimated with a mean error (10.8 cm) slightly less than the event-based algorithm of Censi et al. (2013), but a better performance was seen in estimating orientation (mean error of 5.1°). Also, in both position and orientation, the standard deviation of the results was much lower (7.8 cm for position, 2.4° for orientation). Again, the limited resolution of the DVS is given as the main reason for the perceived errors.

Chapter 4

Event-based Optical Flow

The field of optical flow estimation from event-based vision has gained interest since the availability of the DVS. Several methods have been developed over the past six years. This chapter provides first an overview of these methods and their characteristics in Section 4-1. Second, existing frameworks for evaluation of event-based optical flow performance are discussed in Section 4-2. Third, existing methods for computing visual observables from event-based optical flow are described in Section 4-3.

4-1 Optical flow estimation techniques

To make the best use of event information, event-based optical flow computation is generally performed with each newly detected event. By itself, an event provides no motion information, but by correlating events to recently detected events from neighboring pixels, optical flow is estimated. Hence, the methods presented in this section make use of the *event history* in the spatiotemporal neighborhood of each new event.

4-1-1 Pixel velocity from neighbor events

A very simple approach is to assume that neighbor events occurring within a short time window are caused by motion of a single feature. As such, a velocity vector may simply be estimated by computing the time difference between each new event, and the events of the neighbors. Very basically, such a method would then compute velocity using:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1/\left(t_{x,y} - t_{x-\Delta x,y}\right) - 1/\left(t_{x,y} - t_{x+\Delta x,y}\right) \\ 1/\left(t_{x,y} - t_{x,y-\Delta y}\right) - 1/\left(t_{x,y} - t_{x,y+\Delta y}\right) \end{bmatrix} \tag{4-1}$$

with $t_{x,y}$ indicating the time of the last event at pixel (x, y), and $\Delta x, \Delta y$ indicating the spacing between pixels in x and y directions. Note that in these equations, (x, y) indicate the pixel location in the grid, as opposed to $(\hat{x}, \hat{y})$.

This approach was taken as a starting point by Conradt (2015) in order to implement event-based optical flow estimation on-board of a small (7 cm by 6 cm) quadrotor equipped with a meDVS. From the local flow vectors, a global optical flow field based on angular body rates (i.e. no translation) is determined using flow collected within 10 ms time windows, and solving for the body rates. Manual control experiments with the quadrotor showed that the method runs in real-time on-board, although the estimates contained significant amounts of noise.

In the jAER software (Delbruck, 2007) a filter is present that estimates local optical flow, referred to as *DvsDirectionSelectiveFilter*. This algorithm is fundamentally based on the same assumption of pixel velocity, but with some extensions. First, only events that can be identified as part of an edge are used. This occurs for a new event when direct neighboring pixels fired events shortly before. The direction of the most recent neighbor event is used as edge orientation. Then, pixel velocity is computed by searching for events *normal* to the edge. In the experiments by Paz Gomes Verdugo (2015) this algorithm showed the best performance compared to the other candidates.

An inherent drawback of the simplicity of these two methods is that motion can only be computed in fixed directions. The first method is extremely simple and barely produces any computational load, but horizontal and vertical velocity components are computed independently of one another, meaning that any cross-coupling between the two is neglected. The second method does consider diagonal motion, but due to the orientation property, the direction of motion vectors is constrained to the amount of possible orientations. When only direct neighbors are considered, this means that velocity can only be identified in eight different directions: horizontal, vertical, and twice diagonal, in both positive and negative directions. This limits the accuracy that can be achieved with a single optical flow vector. However, integrating multiple flow vectors may still lead to accurate results.

4-1-2 Event-based Lucas-Kanade

Benosman et al. (2012) introduce an event-based adaptation of the commonly used frame-based technique by Lucas and Kanade (1981). It follows from the same derivation as the original algorithm, as discussed in Section 2-2-1. However, instead of absolute brightness values, relative brightness values are estimated by summing the polarities of events within a time window $t' \in [t - \Delta t, t]$. In this case, I_x, I_y, and I_t are computed through numerical differentiation. The original formulation by Benosman et al. (2012) is as follows:

$$\begin{aligned} I_x(x,y) &= \sum_{t' \in [t-\Delta t, t]} P(x,y,t') - \sum_{t' \in [t-\Delta t, t]} P(x-1,y,t') \\ I_y(x,y) &= \sum_{t' \in [t-\Delta t, t]} P(x,y,t') - \sum_{t' \in [t-\Delta t, t]} P(x,y-1,t') \\ I_t(x,y) &= \frac{1}{t-t_1} \sum_{t' \in [t_1, t]} P(x,y,t') \end{aligned} \tag{4-2}$$

Partial brightness derivatives are computed here using a numerical backward differences scheme. Note that the time window for computing I_t differs, since instead of the regular time window, $t' \in [t_1, t]$ is used, where $t_1 < t$.

In more recent work by Brosch et al. (2015) the problem and related equations were reformulated slightly. Since events by themselves result from temporal changes in brightness, an event is already a measure of I_t. This approach is therefore not based on first-order partial derivatives, but rather on *second-order partial derivatives* I_{tx}, I_{ty}, and I_{tt}. Nevertheless, the authors showed that Eq. (4-2) holds for second-order derivatives as well, and is still applicable to optical flow estimation. However, instead of using a backward numerical difference scheme, a central difference scheme was used for computing spatial derivatives.

In experiments presented by Benosman et al. (2012), the algorithm showed reasonable capacity to estimate orientation of motion and real-time capacity on a desktop computer. However, the tests performed used only very locally moving stimuli, and while optical flow vectors seen in results had logical direction, their magnitude was often constant, independent of motion speed. This is visible in an example result shown in Figure 4-1. Brosch et al. (2015) argued that the limited number of events that is usually available, leads to severe inaccuracies in the brightness derivatives.

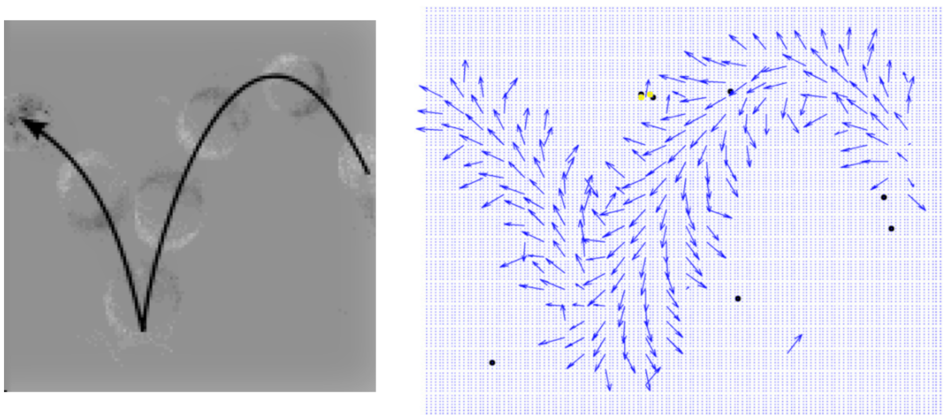


Figure 4-1: Optical flow estimated from a bouncing ball sequence using the event-based adaptation of Lucas-Kanade. From Benosman et al. (2012).

An extension of the algorithm was investigated by Paz Gomes Verdugo (2015). It was found that in its current form, motion is identified along the direction of edges rather than normal to that edge. The cause found for this issue was the presence of multiple events along an edge. To counteract this behavior, the property of orientation was determined to events, similar to the *DvsDirectionSelectiveFilter* implemented in jAER (Delbruck, 2007). After the computation of velocity using the method from Benosman et al. (2012), velocities identified along the edge orientation of events are set to zero. This modification slightly improved the algorithm performance.

4-1-3 Spatiotemporal plane fitting

Benosman et al. (2014) proposed an approach based on event representation in space-time as three-dimensional points. Events are assumed to be part of a 'surface of active events' Σ_e that maps the position of an event to its timestamp: $t = \Sigma_e(x, y)$. An illustration of this concept is shown in Figure 4-2. The inverse gradient components of Σ_e are then equivalent to optical flow. Further, it is assumed that velocity is locally constant or, equivalently, that Σ_e

is planar within a limited spatiotemporal window. Therefore, local velocities are estimated as the inverted gradients of a plane fitted to a set of events within this window. In Benosman et al. (2014), a robust plane fitting procedure is applied to a spatiotemporal window centered on each event. An initial plane is fitted to events within this window through linear least-squares, followed on an iterative process of outlier rejection and repeating the least-squares fit. ON and OFF events are processed separately. In experimental results the algorithm was able to identify motion with reasonable accuracy, although again only from simple clearly identifiable visual stimuli.

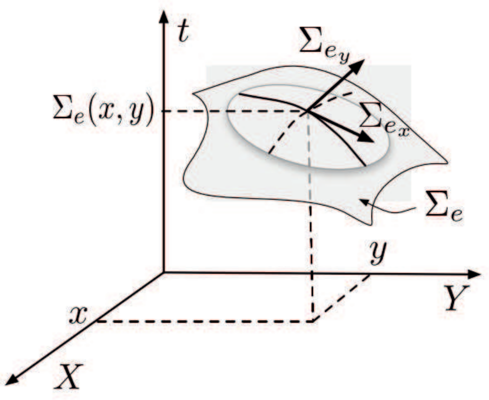


Figure 4-2: Illustration of Σ_e in space-time and its gradients. From Benosman et al. (2014).

In Benosman et al. (2014) the velocity components are computed as independent inverses of the plane slopes. Brosch et al. (2015) argued this is mathematically incorrect and also leads to practical errors. A new approach to velocity estimation is proposed that is fundamentally based on edge motion in space-time.

4-1-4 Flow-based corner detection

Clady et al. (2015) proposed an extension of the spatiotemporal plane fitting approach to solve the aperture problem by detecting corners. Similar to all other local methods, the plane fitting approach cannot be used to estimate true optical flow, but only normal flow. In this work the authors proposed to detect intersections of edges (i.e. corners), through geometrical constraints provided by the plane estimates, and combining the normal flow along these edges to obtain fully observable, non-normal flow. The approach is incremental and shows accurate results in simple scenes, as shown for example in Figure 4-3. However, due to the DVS's limited resolution, corners are difficult to detect for complex scenes with high event density. Further, while no indication of computational performance is provided, the full algorithm is relatively complex compared to those seen in other approaches.

4-1-5 Direction selective filtering

A biologically motivated approach is presented by Tschechne, Sailer, and Neumann (2014), based on the first stages of spatiotemporal processing in vision. In this approach filters are

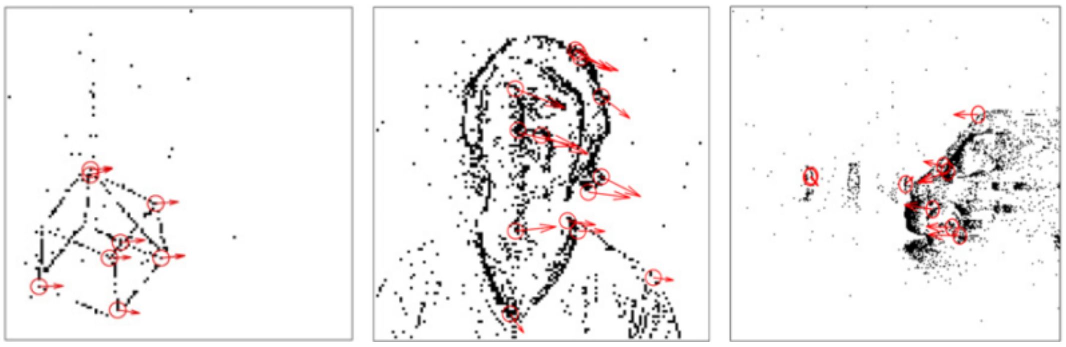


Figure 4-3: Corner points and optical flow estimated in three scenes: a 3D cube wireframe, a human face, and an outdoor scene with a moving car. Adapted from Clady et al. (2015).

constructed that mimic the direction and speed selectivity mechanisms of V1 cells in the visual cortex (De Valois, Cottaris, Mahon, Elfar, & Wilson, 2000).

A filter function is built up as a combination of two spatiotemporal filters, each built up as a combination of two spatial filters and two temporal filters. Figure 4-4 illustrates how the filters are built up in one spatial dimension and time. Of the spatial filters, one is even-symmetric and the other is odd-symmetric. The temporal filters are biphasic and monophasic respectively. These filters were tuned based on experimental data obtained from V1 cells by De Valois et al. (2000). In two dimensions, the spatial filters can be rotated to control in which direction the combined filter is maximally selective. Therefore, a filter bank is generated using different rotation settings for the spatial filter (e.g. 0°, 45°, 90° and 135°), to generate combined filters that are maximally selective in different directions.

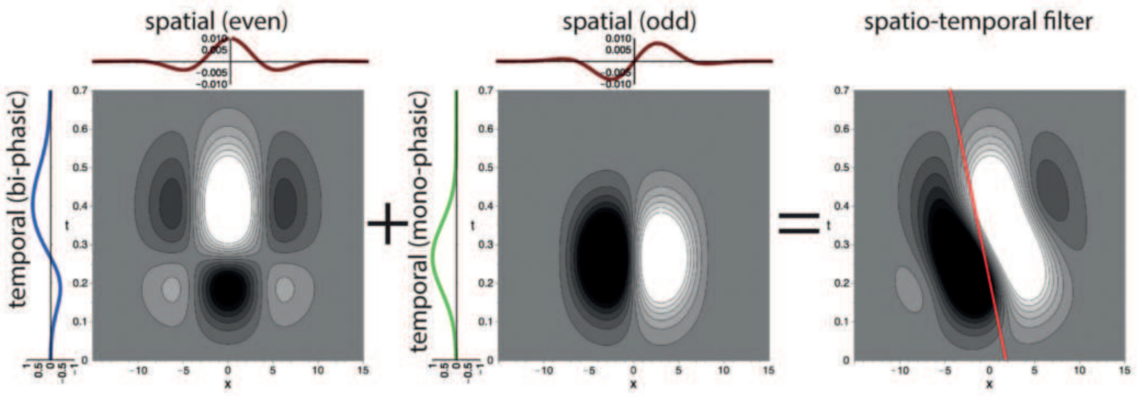


Figure 4-4: Composition of spatiotemporal filters in one spatial dimension and time. From Brosch et al. (2015).

To compute optical flow using these filters, events within a spatiotemporal neighborhood of a new event are convolved with each filter, i.e. the filter values at the events' relative position and timestamp are determined. The sum of these filter values results in a *confidence value* for the direction associated with this filter. In this work, the true flow values are not computed, but by summing the confidence values of all filters, the most likely direction of the flow is obtained. However, since each filter is also maximally selective for a certain speed, filters may also be defined that select different speeds. Then, a velocity value may also be computed for the flow.

Brosch et al. (2015) extend the spatiotemporal filtering method by Tschechne et al. (2014) and investigated the relation between the filter parameters and the 'preferred speed' of the filters through Fourier analysis. This facilitates setting up a filter bank accounting for various speed selectivities. In order to reduce the occurrence aperture problem, a response normalization filter based on a layer of spiking neurons is added. This filter reduces flow confidence values where ambiguity is present, i.e. along edges. However, this normalization mechanism still works with the filter confidence values and is not readily usable with optical flow speeds.

Note that, through direction and speed selectivity, this filter-based approach shows similarities with EMDs seen in insect eyes (see Section 2-2-2). However, the filter structure allows for simultaneous integration of multiple events.

4-1-6 Event-based Elementary Motion Detection

Camus (2010) investigated the use of a Reichardt detector (see Section 2-2-2) to estimate optical flow. Normally, a Reichardt detector combines two continuous-time signals from separate photodetectors. In the case of an event stream, it is necessary to convert the event stream data to a continuous-time signal. The author approached this issue by representing events by Gaussian probability distributions centered at the event. Several additional rules were necessary to resolve and integrate multiple EMD results.

A different approach to using a Reichard detector is proposed by Richter, Rohrbein, and Conradt (2014), which involves combining the detector with a spiking neural network. Pixels of the DVS provide spikes when events are generated. The Reichard detectors are represented by neurons that respond to spikes of a pixel and its neighbor, the latter including a fixed time delay. Thus, a detector neuron spikes when these two input spikes follow each other rapidly such that a threshold of the neuron state is exceeded. To avoid interference from flicker, the authors propose to use a filter where detections in opposite directions are subtracted from one another. This method is intended to be used in combination with a SpiNNaker, a massively parallel computing system optimized to simulate large spiking neural networks. As such, the algorithm can run in real-time consuming little power.

4-2 Evaluating event-based optical flow performance

The development of event-based optical flow techniques generates the need for a consistent approach for comparing the performance of these approaches. Clearly, existing computer vision benchmarks such as the Middlebury dataset (Baker et al., 2011) are by themselves not suitable. Two very recent publications address this problem and propose datasets for evaluating optical flow technique performance.

Barranco et al. (2016) published a dataset for visual navigation using event-based vision. In this dataset synthetic image and event sequences were generated using 3D graphics, generating artificial events at image locations where brightness increased or decreased. In addition, real datasets were generated by combining a DAVIS sensor with a Kinect RGB-D sensor and performing short rotational and translational movements using a robotic platform. Ground truth motion is provided from odometry measurements of the platform. This approach is useful since it provides both image data and events, facilitating comparisons between frame-based and event-based techniques.

Similarly, Ruckauer and Delbruck (2016) collected event sequences and images using a DAVIS, as well as artificial event sequences for simple motion types. For the DAVIS sequences, ground truth flow was obtained using rotational rate output of the IMU present in the sensor. Interestingly, the sequences in the dataset contain primarily *normal motion* of edges, diminishing the effects of the aperture problem in the evaluation process. This is useful given that most methods published so far are local.

Additionally, the authors compare several variants of three flow methods described in Section 4-1: the Lucas-Kanade method (Benosman et al., 2012), the plane fitting method (Benosman et al., 2014), and the *DvsDirectionSelectiveFilter* algorithm (Delbruck, 2007). All algorithms are implemented in jAER and achieve real-time performance for event rates lower than 1e5 events per second on a Core i5 2.4 GHz PC. In these results, the authors improve the Lucas-Kanade and plane-fitting approaches through Savitsky-Golay filtering, reducing computational effort and improving directional accuracy. Also, noise suppression was achieved through application of a refractory period on pixels. The most favorable results are seen for the plane-fitting approach variants.

4-3 Estimating visual observables

The event-based optical flow techniques discussed in Section 4-1 operate on events in the order of occurrence. Additionally, these methods are predominantly local and are subjected to the aperture problem discussed in Section 2-2-1. In contrast, frame-based techniques for estimating visual observables from optical flow rely on fully determined (non-normal) flow estimates and simultaneous availability of flow across the image. Therefore, new approaches were sought for event-based visual observable estimation.

4-3-1 Using sequential flow vectors

Paz Gomes Verdugo (2015) investigated and compared several techniques for estimating the FoE from sequentially detected flow vectors. One promising technique is the 'cross-product method'. It is based on the assumption that, at the FoE, all flow vectors intersect. This method performed the second best in the analysis by Paz Gomes Verdugo (2015).

The definition of the cross-product method is correct for non-normal optical flow. However, in the case of normal flow, this assumption is violated, making the method less suitable for combination with most existing event-based optical flow techniques.

4-3-2 Mapping techniques

An event-based FoE estimation method, which is robust to normal flow estimates, is introduced by Clady et al. (2014) to estimate time-to-contact τ. It employs a spatial probability map whose maximal value indicates the location of the FoE. Since any correct flow vector component, including normal flow, should point away from the FoE, the part of the image plane 'behind' the flow vector is likely to contain the FoE. Hence, for this plane part the probability value is increased. Superimposing sequential optical flow vectors bound the location of the FoE, as shown in Figure 4-5. To prevent over-fitting of the map to a certain pattern, an exponential decay is applied to the full map. Then τ is found for each individual optical flow estimate using the distance to the FoE (see Eq. (2-4)).

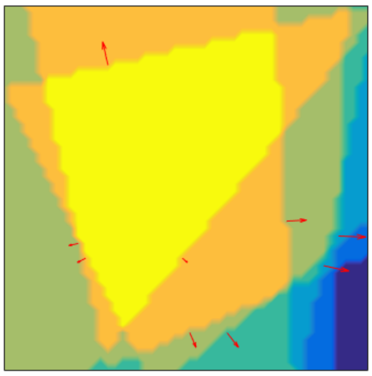


Figure 4-5: A probability map for locating the FoE bounded by several flow vectors. The yellow area indicates the map region with the highest probability for the FoE location.

In their work, the authors use an ATIS sensor on-board of a mobile robotic platform to compute τ and a laser range finder to obtain a ground truth estimate. Optical flow is estimated using the plane fitting algorithm by Benosman et al. (2014). During experiments with the robotic platform the optic-flow-based τ showed good correspondence to the ground truth estimate. Also, computational effort needed on a desktop PC was very low (on average 20 μs per event).

Paz Gomes Verdugo (2015) introduced a different mapping technique, separating the u- and v-components of the flow. Instead, two one-dimensional maps for the x- and y- coordinates of flow vectors are maintained. The maps are applied to identify separately the coordinates where u and v are minimal in magnitude, hence identifying the location of the FoE. During the experiments the method showed the best results compared to other methods for determining the FoE.

A limitation of both approaches is, however, that they limit themselves to the size of the retina. If the FoE lies outside of the mapping region, it cannot be identified with map-based approaches.

Chapter 5

Synthesis

An extensive literature study was performed at the start of this thesis, focusing on three main areas. Valuable insights were obtained in the application of optical flow for landing tasks, and the role of hardware in this form of navigation. Also, a broad overview of state-of-the-art event-based cameras and their applications was established. Most importantly, several existing approaches to event-based optical flow estimation were found, which forms a proper basis for the design of a pipeline for event-based estimation of visual observables. This chapter summarizes the main findings and addresses their relevance per topic.

5-1 Landing strategies

Optic flow based navigation strategies seen in biology have been widely applied in MAVs and other robotic systems. They provide computationally lightweight, simple approaches for controlling ego-motion during maneuvers. We saw that grazing landings can be performed through keeping ventral flow constant and reducing forward speed or height. This approach works only if the MAV has forward motion. Through controlling time-to-contact or flow divergence, pure vertical motion may also be controlled. This is highly desirable for MAVs that need to operate within confined locations. With augmentation by ventral flow control for horizontal motion, the MAV can stabilize its motion above a suitable landing site, and then perform vertical landing through divergence, or equivalently, time-to-contact.

As seen in landing experiments involving time-to-contact or divergence (Alkowatly et al., 2015; Herissé et al., 2012; Ho & De Croon, 2016), time delays introduced by frame-based optical flow estimation are significant and limit the speed at which purely vision-based maneuvers can be performed without sacrificing stability. This problem holds not only for landing or docking tasks, but also for other visual navigation tasks. Special optical flow sensors achieve higher sampling rates and are usable for ventral flow control, but have limited capabilities when optical flow patterns become more complex, making them less suitable for estimating divergence. Combinations of several sensors are possible, though this is undesirable in smaller MAV where space and weight constraints are strict. The CurvACE sensor (Floreano et al.,

2013) offers attractive characteristics for MAV experiments, due to its high sampling rate and wide field of view. However, its power consumption is relatively high, while its resolution is lower compared to the DVS or related sensors.

5-2 Event-based cameras

Event-based cameras are an attractive alternative due to their micro-second temporal accuracy and small latency. The sparse nature of their output may considerably simplify visual processing necessary to obtain local optical flow vectors. In a control system this is highly desirable for achieving a fast response without sacrificing stability. Theoretically, event-based cameras can therefore greatly enhance the response of existing landing control systems. Further practical benefits of event-based pixels are that the sensor's intrascene dynamic range is very high (>120 dB) and that power consumption is limited (<23 mW for DVS-based sensors).

The DVS is a frequently used event-based camera that is convenient in usage, due to its USB connectivity and interface with jAER. In experiments, its low resolution (128×128 pixels) is frequently reported as a limiting factor. Also, the use of USB communication increases the actually achievable latency to values in the order of milliseconds instead of microseconds. However, in more recent event-based cameras, such as the ATIS, DAVIS and C-DAVIS, resolution has already been improved significantly. Also, embedded and miniaturized versions of the DVS do not rely on USB and can achieve the sensor's 15 µs latency. The miniaturized meDVS is especially interesting for future usage of event-based vision on-board MAVs. As this sensor is functionally equivalent to a DVS, algorithms developed in this work are transferable to this smaller sensor.

5-3 Event-based optical flow

Several approaches to event-based optical flow estimation have been discussed, and several (Benosman et al., 2014; Conradt, 2015) have shown real-time capacity in experiments, obtaining good results using simple visual stimuli. Comparing the performance of different approaches based on the paper results is challenging since for each approach, different means of evaluation were used. Based on the criticism by Brosch et al. (2015), event-based Lucas-Kanade (Benosman et al., 2012) appears to be error-sensitive with little events, mainly in magnitude. The filtering approach is, as presented, not capable of estimating motion, except if it is possible to map confidence values to velocities, or by including speed selectivity settings.

By themselves, all approaches are local and can only estimate normal flow. This leads to errors when the flow estimates are to be integrated into visual observable estimates. Flow-based corner detection (Clady et al., 2015) may provide a solution here. This does add a layer of complexity to a sensing algorithm. The time-to-contact estimation approach by Clady et al. (2014) provides an alternative approach that is robust to normal flow. However, it is invalid when the FoE is outside of the field of view, limiting the application of this approach to pure vertical landing.

Interestingly, over the course of the preliminary work in this thesis, two datasets for evaluating event-based optical flow were published. These partially serve the same purpose as the preliminary analysis performed in this work. However, the fundamentals of the datasets are significantly different, especially the composition of ground truth data. Also, the events are recorded with a DAVIS instead of the DVS and illumination conditions are likely different than in the CyberZoo where our dataset was recorded. The event data is thus not representative of the conditions where the experimental phase of this thesis will take place. Nevertheless, when a correction is made for the different resolution of the DAVIS, the event datasets and ground truth are usable for evaluation of event-based optical flow algorithms. In particular, the evaluation results from Ruckauer and Delbruck (2016) are interesting, since three methods evaluated here are also considered in our preliminary analysis performed in Part III, and the ground truth flow corresponds to normal flow.

Part III

Preliminary Comparison of Event-Based Optical Flow Methods

Chapter 6

Methodology and Datasets

In this preliminary comparison we present a first analysis conducted on methods in literature for estimating event-based optical flow. In order to gain a better understanding of event-based vision and optical flow methods, a preliminary implementation of several techniques is performed, in which they estimate flow from realistic event data. In addition, to form a basis for further work conducted in this thesis, a framework is presented for evaluating the performance of optical flow methods and visual observable estimation methods.

In this chapter an overview of the steps in the analysis is provided. We demonstrate how pose information is used to compute ground truth visual observables and optical flow. Further, a brief description of the input datasets is given. The following chapters provide an in-depth description of the evaluation procedures and their results. Event-based optical flow estimation is discussed in Chapter 7, followed by the subsequent process of visual observable estimation in Chapter 8.

Over the course of the preliminary work it was considerefd to correct the DVS output for lens distortion. A procedure was developed for calibrating the DVS based on its event input. Corrections for event positions and event-based optical flow were developed and demonstrated, which are detailed in Chapter 10. For this preliminary comparison, the lens distortion is not yet incorporated due to an inaccurate result, but it does provide the input parameter values for the camera's principal point and focal length.

6-1 Outline of the analysis

In an optical flow based landing maneuver, the visual sensor used is mounted on a MAV while facing normal to the ground, such that it perceives ventral flow and divergence cues. In order to assess how well different techniques for estimating these cues perform in a landing task, we record event datasets using a DVS during simulated landing maneuvers. The DVS is moved by hand in translational motion above a flat floor.

In addition, ground truth position, velocity, and attitude data of the camera is obtained by simultaneously tracking the pose of the DVS. This is done using a Optitrack camera system, which is capable of tracking the pose of objects in the area with great accuracy. This system can track the position, speed, and orientation of objects that are fitted with several infrared markers. From this information, ground truth values for ventral flow and divergence can be estimated using the relations defined in Chapter 2.

The obtained event and pose measurement datasets are employed to assess the performance of event-based methods for local flow estimation and visual observable estimation. Each candidate algorithm is implemented and evaluated by computing estimates and comparing the output to the ground truth value. This process of implementation and evaluation of the methods is performed using MATLAB. This procedure is decoupled in a two-stage process: local optical flow estimation performance and visual observable estimation performance are assessed in two separate procedures.

6-2 Determining ground truth visual observables and optical flow

Pose and velocity information from Optitrack is used to compute ground truth values for local optical flow, ventral flow, and divergence. For this we apply the planar optical flow relations derived in Section 2-1. In the general case, the DVS will not be perfectly aligned to the ground. Therefore, the velocity information from Optitrack needs to be represented in the *camera reference frame* $\mathcal{C}$ through a transformation.

To this end, we define two additional reference frames. Optitrack position and velocity information is represented in the inertial *world reference frame* $\mathcal{W}$. With respect to this inertial frame, the DVS is represented by the *body-fixed reference frame* $\mathcal{B}$. The body-fixed frame results from three sequential rotational transformations, represented by Euler angles ϕ, θ, ψ. The sequence of rotation is (1) a yaw rotation ψ around the Z-axis, (2) pitch rotation θ around the Y-axis, and (3) roll rotation ϕ around the X-axis. A graphical representation of the reference frames and the Euler angle definitions is shown in Figure 6-1.

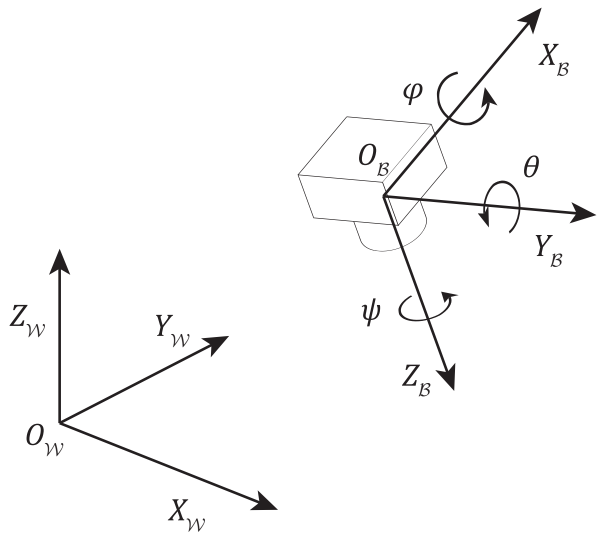


Figure 6-1: Optitrack-based world reference frame and the body-fixed reference frame, with Euler angle definitions.

The relations in Section 2-1 are expressed in $\mathcal{C}$, which is oriented depending on the camera lens orientation. During initialization the camera lens is pointed approximately perpendicular to the ground surface. Hence, the assumption is introduced that $Z_{\mathcal{C}} \approx Z_{\mathcal{B}}$. Further, the camera is aligned in heading with a geometrical marker in the CyberZoo, such that $X_{\mathcal{C}} \approx Y_{\mathcal{B}}$ and $Y_{\mathcal{C}} \approx X_{\mathcal{B}}$. Note that X and Y are interchanged in the observer reference frame with respect to the body reference frame; this is due to the definition of pixel coordinates in DVS events. Lastly, in this preliminary work it is assumed that the camera nodal point intersects with the body center of mass.

Based on the former, a linear transform $\mathbb{T}$, representing the yaw, pitch, and roll rotations, is applied to obtain the body velocity components $U_{\mathcal{B}}, V_{\mathcal{B}}, W_{\mathcal{B}}$ from the Optitrack velocity measurements $U_{\mathcal{W}}, V_{\mathcal{W}}, W_{\mathcal{W}}$:

$$\begin{bmatrix} U_{\mathcal{B}} \\ V_{\mathcal{B}} \\ W_{\mathcal{B}} \end{bmatrix} = \mathbb{T} \begin{bmatrix} U_{\mathcal{W}} \\ V_{\mathcal{W}} \\ W_{\mathcal{W}} \end{bmatrix} \tag{6-1}$$

Due to the assumed correspondence between the observer reference frame and the body-fixed reference frame, this can also be expressed as:

$$\begin{bmatrix} V_{\mathcal{C}} \\ U_{\mathcal{C}} \\ W_{\mathcal{C}} \end{bmatrix} = \mathbb{T} \begin{bmatrix} U_{\mathcal{W}} \\ V_{\mathcal{W}} \\ W_{\mathcal{W}} \end{bmatrix} \tag{6-2}$$

To compute the transformation $\mathbb{T}$, a quaternion attitude representation is applied (Diebel, 2006). The quaternions q_0, q_x, q_y, q_z are introduced, which are related to the Euler angles through Eq. (6-3). Here, a shorthand notation $c_x = \cos(x), s_x = \sin(x)$ is introduced.

$$\begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} c_{\phi/2}c_{\theta/2}c_{\psi/2} + s_{\phi/2}s_{\theta/2}s_{\psi/2} \\ s_{\phi/2}c_{\theta/2}c_{\psi/2} - c_{\phi/2}s_{\theta/2}s_{\psi/2} \\ c_{\phi/2}s_{\theta/2}c_{\psi/2} + s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}s_{\psi/2} - s_{\phi/2}s_{\theta/2}c_{\psi/2} \end{bmatrix} \tag{6-3}$$

The transformation matrix $\mathbb{T}$ is then composed by the quaternions as follows:

$$\mathbb{T} = \begin{bmatrix} {q_0}^2 + {q_x}^2 - q_y^2 - {q_z}^2 & 2\left(q_x q_y + q_0 q_z\right) & 2\left(q_x q_z - q_0 q_y\right) \\ 2\left(q_x q_y - q_0 q_z\right) & {q_0}^2 - {q_x}^2 + q_y^2 - {q_z}^2 & 2\left(q_y q_z + q_0 q_x\right) \\ 2\left(q_x q_z + q_0 q_y\right) & 2\left(q_y q_z - q_0 q_x\right) & {q_0}^2 - {q_x}^2 - q_y^2 + {q_z}^2 \end{bmatrix} \tag{6-4}$$

The distance to the ground along the optical axis Z_0 is computed through the following geometrical relation:

$$Z_0 = \frac{Z_{\mathcal{W}}}{\cos\phi\cos\theta} \tag{6-5}$$

which is used to compute the ground truth ventral flows and divergence as follows:

$$\omega_x = -\frac{U_{\mathcal{C}}}{Z_0}, \quad \omega_y = -\frac{V_{\mathcal{C}}}{Z_0}, \quad D = 2\frac{W_{\mathcal{C}}}{Z_0} \tag{6-6}$$

Using the obtained ventral flow and divergence, a ground truth flow vector can be computed for each event where a flow vector is estimated. Since we collect data above an approximately planar surface, Eq. (2-5) derived in Section 2-1-2 is applied to compute a planar flow field. The slopes Z_X, Z_Y in these equations are computed from the camera roll and pitch angles obtained from Optitrack:

$$Z_X = \tan\phi, \quad Z_Y = \tan\theta \tag{6-7}$$

As the data from Optitrack is generally available at a lower sampling rate than the event rate, all flow field parameters Z_X, Z_Y, ϑ_x, ϑ_y, and ϑ_z are calculated through linear interpolation based on the event timestamp. Then, using the event position a ground truth flow vector is calculated.

6-3 Dataset description

Three main datasets were used recorded by moving the DVS above a flat floor. Each dataset consists of a sequence of recorded events, as well as the Optitrack pose information. The pose information consists of position and speed in the Optitrack reference frame, and the attitudes of the camera.

Two datasets have been recorded above a checkerboard texture shown in Figure 6-2a. In the first set the camera moves such that expansion and contraction occur in its field of view, but with a focus of expansion left of the center of view. This leads to a combination of ventral flow and divergence. In the second set the camera moves parallel to the surface, such that mainly ventral flow is perceived.

The third dataset is recorded above a roadmap playmat (see Figure 6-2b), which leads to less structured features in the field of view, but a higher feature density. The motion performed for this dataset consists of a short part with contraction, followed by ventral flow, and completed with a part with expansion.

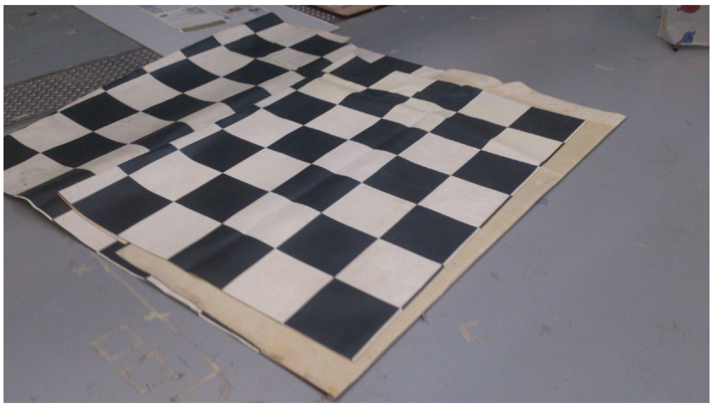
An overview of the datasets and their characteristics is provided in Table 6-1. Note that the event density is higher for the road map dataset due to the more complex texture.

Table 6-1: Summary of event dataset characteristics

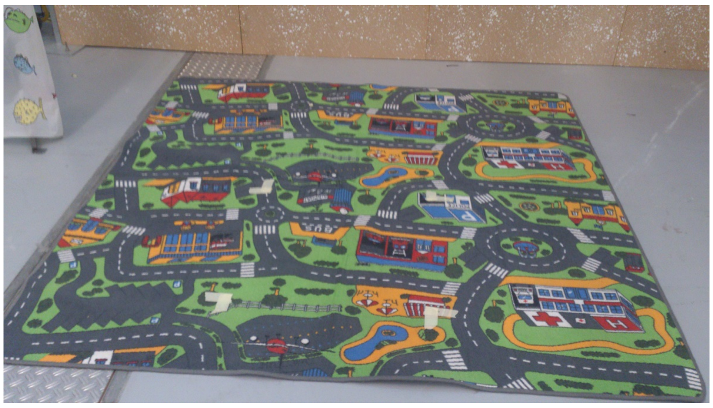
	Texture	Primary motion cue(s)	Duration [s]	Number of events
Set 1	Checkerboard	Off-center expansion and contraction	22.97	5.41e5
Set 2	Checkerboard	Ventral flow	16.99	2.37e5
Set 3	Road map	Centered contraction, ventral flow, and expansion	4.97	3.36e5

In Figure 6-3 the computed visual observables are shown for each dataset. Additionally, height and event rate are displayed. The event rate is computed at each event timestamp by summing all previous events in a 0.1 s window.

The motion types highlighted in Table 6-1 are clearly present. Some observations show that the current dataset may require improvement. First, the height measurements in set



(a) Checkerboard texture



(b) Roadmap texture

Figure 6-2: Textures used for data acquisition.

2 show that the spatial resolution was limited to 1 cm, and the sampling rate is limited to approximately 3 Hz. These are settings that can be adjusted in recording the pose data, and are therefore adjusted in upcoming datasets.

Second, in sets 1 and 2 there are sharp peaks in the event rates. These are due to 'flashes': distortions where a large part of the sensor triggers. An example of flash events is shown in Figure 6-4. Most likely these flashes are caused by incorrect settings in the DVS pixel thresholds imposed by jAER. In set 3 the DVS pixel firing threshold was increased slightly to remedy this, and as a result, no flash peaks are present in the event rate. Disregarding the peaks due to flashes, typical event rates during motion are in the range of 10k to 30k events per second for set 1 and 2, and in the range of 40k to 120k events per second for set 3.

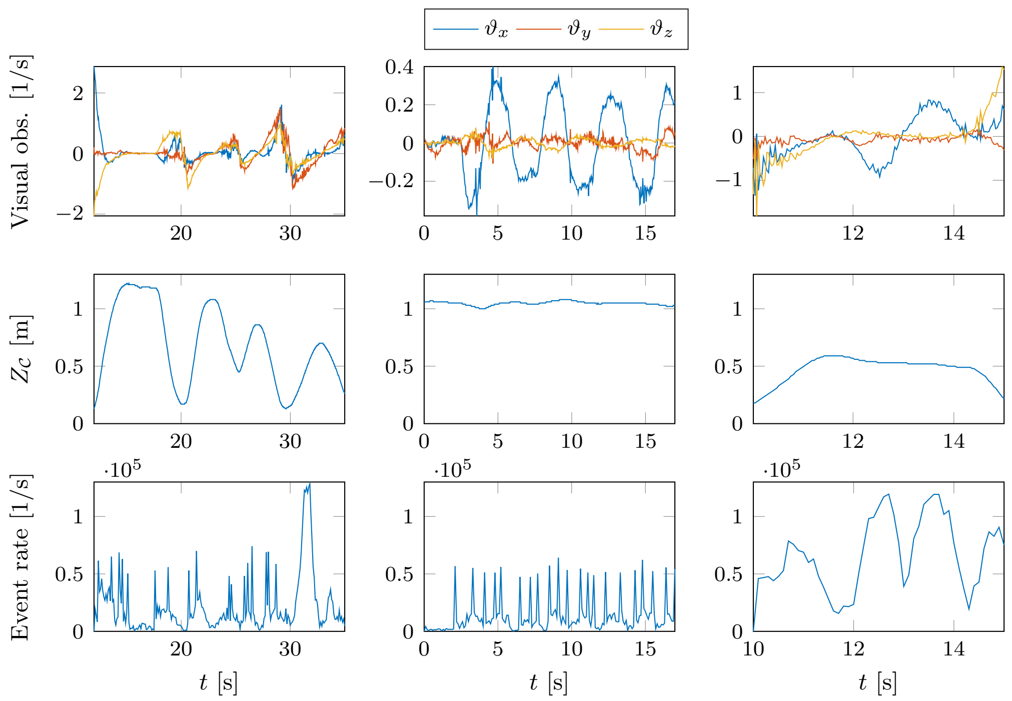


Figure 6-3: Ground truth visual observable, height above ground, and event rates of the input datasets.

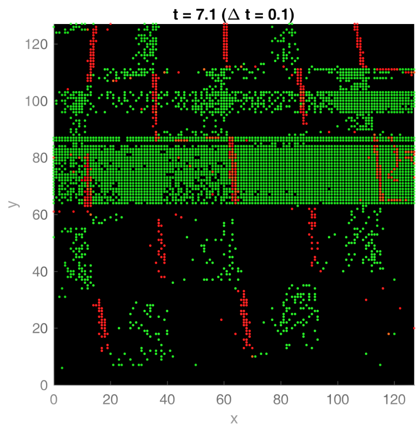


Figure 6-4: Visualization of events recorded during a flash in set 2. Green dots represent positive polarity (ON) events; red dots are negative polarity (OFF) events. A clear stripe of ON events is visible due to the flash.

Chapter 7

Analysis of Optical Flow Estimation Methods

In this comparison four different algorithms are included to estimate local optical flow. These are based on methods found in literature (see Chapter 4). We evaluate the accuracy, computation time, and flow density of the algorithms through MATLAB implementations. Additionally, we investigate the effect of two filters designed to enhance flow estimates.

7-1 Local flow estimation algorithms

This section describes the implementation of the algorithms. For convenience, abbreviations are introduced to refer to the algorithms in the remainder of this chapter. All algorithms are causal and asynchronous: they operate on each individual event at the moment when it is detected. They incorporate previous events within a spatial neighborhood of the new event, as well as within a limited time range, forming a *spatiotemporal search window* at the new event.

7-1-1 Event-based Lucas-Kanade

The event-based version of Lucas-Kanade (LK) is the first algorithm. As discussed in Section 4-1-2, brightness gradients are approximated by computing sums of event polarities instead of absolute brightness values. A version of the formulation by Brosch et al. (2015) is used, where spatial brightness derivatives are computed through numerical *central differences*, and temporal derivatives through *backward differences*:

$$\begin{aligned}
I_{tx}\left(x,y\right) &= \tfrac{1}{2}\left(\sum_{t'\in[t-\Delta t,t]} P\left(x+1,y,t'\right) - \sum_{t'\in[t-\Delta t,t]} P\left(x-1,y,t'\right)\right) \\
I_{ty}\left(x,y\right) &= \tfrac{1}{2}\left(\sum_{t'\in[t-\Delta t,t]} P\left(x,y+1,t'\right) - \sum_{t'\in[t-\Delta t,t]} P\left(x,y-1,t'\right)\right) \\
I_{tt}\left(x,y\right) &= \frac{1}{\Delta t}\sum_{t'\in[t-\Delta t,t]} P\left(x,y,t'\right)
\end{aligned} \tag{7-1}$$

With respect to the original formulation, one modification is introduced. During implementation, the flow magnitude was found to depend directly on I_{tt} estimates. All estimates of I_{tt} scale proportional to $1/\Delta t$. In the case that a larger Δt setting is used but no extra events are obtained, this means that the magnitude *scales with the time window setting.* To counter this, we set Δt flexible. Events are first collected within a predefined setting for Δt. Then, Δt is adjusted such that the difference to the *earliest occurring event in the time window* is used.

Having brightness derivatives available for all pixels within a spatial neighborhood of the event, a least-squares system of equations is solved:

$$\begin{bmatrix} I_{tx}^1 & I_{ty}^1 \\ \vdots & \vdots \\ I_{tx}^n & I_{ty}^n \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_{tt}^1 \\ \vdots \\ I_{tt}^n \end{bmatrix} \tag{7-2}$$

This system of the form $\mathbf{AV} = \mathbf{y}$ is solved through the linear least-squares solution for the optical flow vector $\mathbf{V}$:

$$\mathbf{A}^T\mathbf{AV} = \mathbf{A}^T\mathbf{y} \tag{7-3}$$

7-1-2 Normal-to-Edge Search

The *DvsDirectionSelectiveFilter* in the jAER software (see Section 4-1-1) was implemented as the Normal-to-Edge Search (NES) algorithm. This approach is based on computing pixel velocity after identification of local edge orientation. Specifically, the algorithm establishes if an event is part of an edge with a discrete *orientation.* For simplicity, only four edge orientations are identified (0°, 45°, 90°, 135°).

Figure 7-1 illustrates the working principle of the algorithm. First, the algorithm establishes evidence for each of the possible edge orientations. This is done based on the orientation along which on average the most recent events occurred. With four possible orientations, this algorithm considers the most recent horizontal, vertical, and diagonal neighbor events. For robustness to missing events, also second and third closest neighbors are included.

Then, if an orientation can be identified, the algorithm searches for events normal to the edge in two opposite directions. The direction in which the most events are found, is selected. Now pixel velocities are computed using the events in the selected direction, and averaged to obtain a final flow estimate.

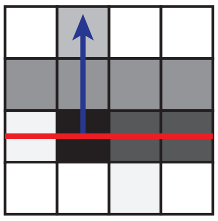


Figure 7-1: Working principle of the NES algorithm using a simple pixel grid. The darkness of a square indicates how recently an event has fired. A horizontal edge orientation (red line) is assigned here, since in this direction the most recent events occurred. Normal to this edge, events are found searching upward (blue vector), which is evidence for downward optical flow.

7-1-3 Space-Time Plane Fitting

The third algorithm is the STPF algorithm proposed by Benosman et al. (2014). As discussed in Section 4-1-3, this algorithm estimates by robust fitting of a local plane to events, represented as three-dimensional points as (x, y, t). For each incoming event, the algorithm collects all events within a spatiotemporal neighborhood. Then, in an *iterative refinement* process, a plane is fitted and improved by rejecting outliers.

The plane fitting process is detailed as follows. A plane $\mathbf{\Pi}$ is represented by the relation $p_x x + p_y y + p_t t + p_0 = 0$. In this preliminary analysis, $p_t = 1$, obtaining a nonhomogeneous equivalent expression for Π. To obtain an initial plane estimate, a linear least-squares plane solution is computed using all points. Then, points whose distance to the plane is larger than a predefined threshold, are rejected, and a new least-squares fit is computed. This process continues until the plane parameters do not change significantly anymore, or the amount of points supporting the plane is insufficient. It was found that usually only two or three iterations are necessary.

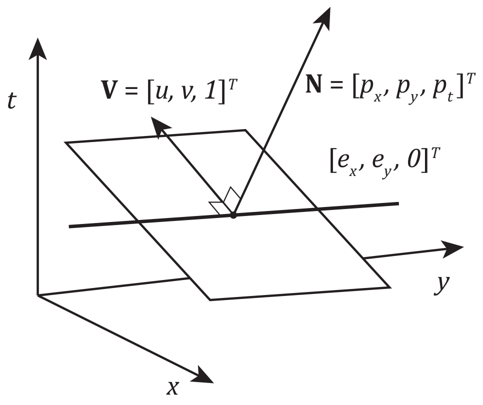


Figure 7-2: Illustration of the orthogonal system of the flow vector $\mathbf{V}$, a plane normal vector $\mathbf{N}$, and a luminance edge described by the vector $[e_x, e_y, 0]^T$.

Having obtained the best plane fit, the approach proposed by Brosch et al. (2015) is used to compute optical flow. In this approach, flow is computed from the orthogonal system of the normal vector of the fitted plane, a representation of the flow vector, and a hypothetical luminance edge that moves along the plane with the flow vector. This normal system is shown in Figure 7-2. It can be shown that in this case, the normal flow of the edge is related to the plane normal vector as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = -\frac{p_t}{p_x^2 + p_y^2} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{7-4}$$

7-1-4 Direction Selective Filtering

The last optical flow algorithm considered in this work is the Direction Selective Filtering (DSF) approach presented by Brosch et al. (2015). In this approach, flow is computed for each incoming event through convolving past events with probability filters for combinations of direction and speed.

Filters are constructed using two spatial and two temporal filters as presented in Section 4-1-5. The spatial filters are represented by the real and imaginary parts of a Gabor filter:

$$G(x,y) = \frac{2\pi}{\sigma_G^2} \exp\left(2\pi j f_G\left(x+y\right)\right) \exp\left(-\frac{2\pi^2\left(x^2+y^2\right)}{\sigma_G^2}\right) \tag{7-5}$$

The monophasic and biphasic filters are composed of several Gaussian filter kernels:

$$\begin{aligned} T_{mono}(t) &= \exp\left(-\tfrac{(t-\mu_{mono})^2}{2{\sigma_{mono}}^2}\right) \\ T_{bi}(t) &= -s_1 \exp\left(-\tfrac{(t-\mu_{bi1})^2}{2{\sigma_{bi1}}^2}\right) + s_2 \exp\left(-\tfrac{(t-\mu_{bi2})^2}{2{\sigma_{bi2}}^2}\right) \end{aligned} \tag{7-6}$$

A full filter is then a combination of these filters:

$$F(x,y,t) = \mathfrak{Im}\left(G(x,y)\right) T_{mono}(t) + \mathfrak{Re}\left(G(x,y)\right) T_{bi}(t) \tag{7-7}$$

In Brosch et al. (2015) the filter parameters (f_G, σ_G, μ_{mono}, σ_{mono}, μ_{bi1}, σ_{bi1}, μ_{bi2}, σ_{bi2}) are estimated based on biological results from De Valois et al. (2000).

In this implementation, an extensive filter bank is created accounting for both direction selectivity and speed selectivity. The spatial filters regulate direction selectivity through their orientation. Speed selectivity can be regulated through adapting the parameters of the temporal filters as described by Brosch et al. (2015). We create filters that account for four different spatial directions (similar to the NES algorithm) and ten different speeds. Hence, a filter bank of 40 combined filters is generated. For fast evaluation the filter components (the spatial and temporal filters separately) are implemented in the form of look-up tables.

For each filter, a confidence value is created by summing the filter outputs for all nearby events. In each direction, the speed with the highest confidence is selected (winner-takes-all). Then, the optical flow is found by summing the speed vectors identified for all directions.

7-2 Evaluation of algorithm performance

7-2-1 Flow error metrics

We evaluated the accuracy of algorithms by computing three error metrics per flow vector, which are depicted in Figure 7-3. By computing the average values of these error metrics for all flow vectors in a datasets, these metrics provide simple means to assess algorithm accuracy in this dataset.

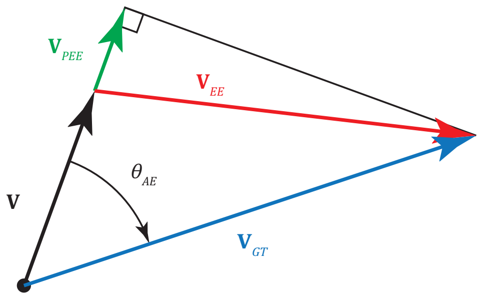


Figure 7-3: Illustration of error metrics. The vector V indicates the estimated flow, while V_{GT} indicates ground truth.

First, the *Endpoint Error (EE)* is considered: the magnitude of the endpoint-to-endpoint error vector:

$$\text{EE} = \|\mathbf{V} - \mathbf{V}_{GT}\| = \sqrt{(u - u_{GT})^2 + (v - v_{GT})^2} \tag{7-8}$$

Second, the *Angular Error (AE)* is computed, which is the difference in direction between the two vectors:

$$\text{AE} = \arccos \frac{\mathbf{V} \cdot \mathbf{V}_{GT}}{\|\mathbf{V}\| \, \|\mathbf{V}_{GT}\|} \tag{7-9}$$

These quantities are commonly used in benchmarking techniques, including the recently presented datasets for event-based optical flow evaluation (Barranco et al., 2016; Ruckauer & Delbruck, 2016). However, as discussed in Chapter 4, most event-based techniques suffer from the aperture problem. Therefore, a quantity for evaluating normal flow estimation performance is a useful addition. For this, we propose the *Projection Endpoint Error (PEE)*, which is the absolute difference between the magnitude of ground truth flow *projected to the estimate flow vector* and the magnitude of the estimated flow:

$$\text{PEE} = \left| \|\mathbf{V}\| - \frac{\mathbf{V}}{\|\mathbf{V}\|} \cdot \mathbf{V}_{GT} \right| \tag{7-10}$$

7-2-2 Processing time and flow output density

The *average processing time per event* for each algorithm is used to assess the computational load of an algorithm. This average is computed for each dataset separately to quantify how much the computational load varies under different conditions.

Lastly, the *flow output density* η is the percentage of the input events for which a flow vector is identified. This value indicates how well an algorithm is compatible with visual observable estimation, and how strict any event or flow vector filtering can be (see Section 7-4). A high value for η requires more processing in the visual observable estimation stage due to the larger amount of flow vectors. However, it also allows for stricter filter settings such that flow quality can increase further, which in turn reduces flow throughput.

7-3 Results

For generating the results in this section, the algorithm parameters were set to use the same space-time window (5x5 pixel neighborhood, events within 3 ms and 100 ms before the current event) for detecting events. For the NES algorithm, this is equivalent to a search distance of 2 pixels. We first show qualitative flow results of the algorithms, followed by a quantitative comparison through the calculated error metrics.

7-3-1 Qualitative results

Figure 7-4 shows a detail view of optical flow estimated during set 3 for each algorithm, along with corresponding ground truth flow. Clear differences are visible in the algorithm output.

The LK algorithm produces flow where each vector by itself is inaccurate, and it appears very difficult to reconstruct the original motion pattern from this flow field. It does result in high density flow, but as can be seen in Figure 7-4 most of the flow vectors have very low magnitude. More desirable performance is seen by the NES algorithm. The observed flow vectors match closely to the ground truth flow. Some outliers are observed, but these are mostly individual outliers. The flow density is in this image very low, and the algorithm appears to miss several clear motion patterns.

The STPF algorithm shows the best performance, and produces highly coherent flow estimates. Deviations are mainly seen at the end points of edges. Also note that some coherent flow estimates deviate from the ground truth, which is due to the aperture problem. Lastly, the DSF algorithm produces flow that is, like the LK algorithm, highly varying in angle and magnitude. The flow orientation is in most cases better than for the LK algorithm, but also many incorrect estimates are seen at smaller features.

7-3-2 Quantitative results

Table 7-1 summarizes the overall performance of the algorithms applied to the three different datasets. The error quantities are shown as the average errors (e.g. average EE, denoted as AEE of the complete datasets. In addition, the computation time needed for the dataset as well as the number of identified velocity vectors are shown.

The STPF algorithm provides the most accurate estimate on all datasets and on all types of errors, but for most datasets it also takes significant computation time. This is sensible, since it involves multiple least-squares operations necessary for plane fitting. Interestingly, it is relatively fast in the case of pure ventral motion. A likely reason is that for this set less plane fit iterations are required per event.

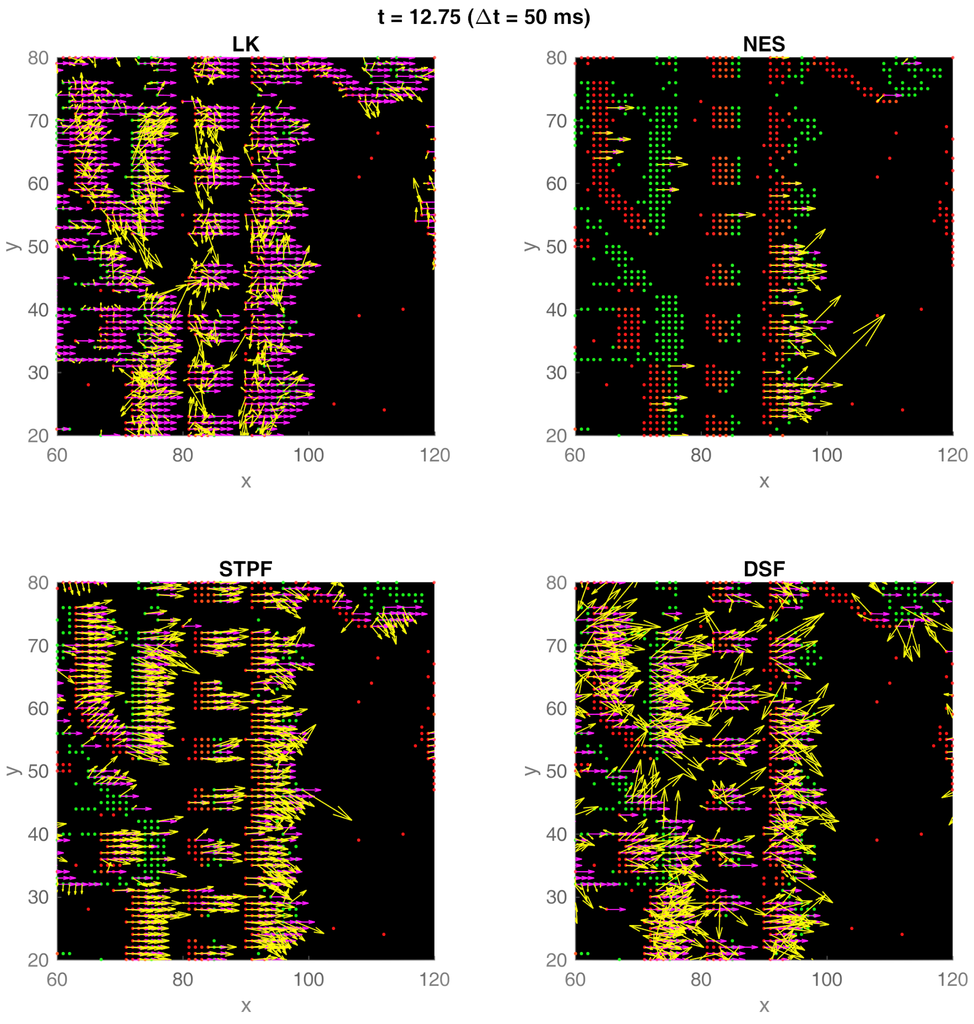


Figure 7-4: Events, estimated flow vectors, and ground truth flow obtained from the four base algorithms in set 3 within a 100 ms time window. Only a limited part of the DVS pixel grid is shown for clarity. The green and red dots indicate events (green = positive polarity, red = negative). The yellow arrows indicate flow estimates for each algorithm. The purple arrows are the accompanying ground truth flow vectors.

NES, on the other hand, appears to be the least computationally demanding. With its current setting it performs slightly less accurate than LK. However, its AAE is small, especially considering that it can only identify motion in eight discrete directions. With current settings the new DSF algorithm shows the lowest performance in absolute errors and medium performance in flow direction estimation (AAE) and computation time.

Clearly, all algorithms perform the best on set 2, where due to the checkerboard texture, and the motion direction, the observed edges perform approximately normal motion. The road map texture appears to be slightly less challenging than the checkerboard texture, despite the higher event density. Note that computation time values per event are too high for real-time

Table 7-1: Optic flow errors and computation characteristics for the three datasets. The A-prefix to error quantities indicates that these are the average of the complete dataset. The symbol t_C indicates computation time per event and η indicates the flow density.

	AEE [pix/s]	AAE [deg]	APEE [pix/s]	t_C [µs]	η [%]
Set 1					
LK	65.66	67.94	44.16	288.0	57.67
NES	77.00	41.59	57.72	**229.5**	26.43
STPF	**60.43**	**39.90**	**39.55**	367.1	32.53
DSF	98.14	57.71	80.49	268.9	28.84
Set 2					
LK	34.20	59.85	27.01	259.4	38.48
NES	33.74	19.57	31.95	239.6	15.61
STPF	**20.11**	**18.13**	**17.77**	243.3	20.21
DSF	54.95	37.25	51.16	**239.1**	22.57
Set 3					
LK	60.26	79.16	39.96	235.6	78.87
NES	58.34	41.84	41.49	**222.8**	9.107
STPF	**41.55**	**31.52**	**25.34**	435.3	30.36
DSF	80.62	53.82	65.08	276.4	22.98

performance. Typical event rates exceed 10k events per second, while in the best case, the NES algorithm can process 4.5k events per second. However, the values are still preliminary and are likely to improve significantly when implementing in C.

As may expected, the newly defined PEE has lower values than the endpoint error, and penalizes algorithms less when their angular error is high. For set 2, the APEE is approximately equal to the commonly used AEE. For the other datasets, the APEE values are lower.

7-3-3 Comparison to results in literature

We validated the results of this work to recent results seen in literature. A first-order comparison of our results to those obtained by Ruckauer and Delbruck (2016) is possible for the LK, NES, and STPF algorithms. Example results are shown in Figure 7-5 for a synthetic dataset.

Interesting is that the LK implementations in Ruckauer and Delbruck (2016) produces much more consistently oriented flow than in this work. This may be due to our choice of flexibly computing the temporal derivative I_{tt}. In addition, Ruckauer and Delbruck (2016) applied an eigenvalue threshold for the least-squares covariance matrix $\mathbf{A}^T\mathbf{A}$ to filter inaccurate flow. However, similar to the original results by Benosman et al. (2012), output flow magnitude is inaccurate, as is visible in Figure 7-5.

The NES approach in our work computes low density flow in comparison to the version by Ruckauer and Delbruck (2016). This is likely due to less strict threshold settings. Qualitative results are similar. The Java-based implementation (referred to as DS) is very efficient and reaches a computational efficiency of 0.36 µs.

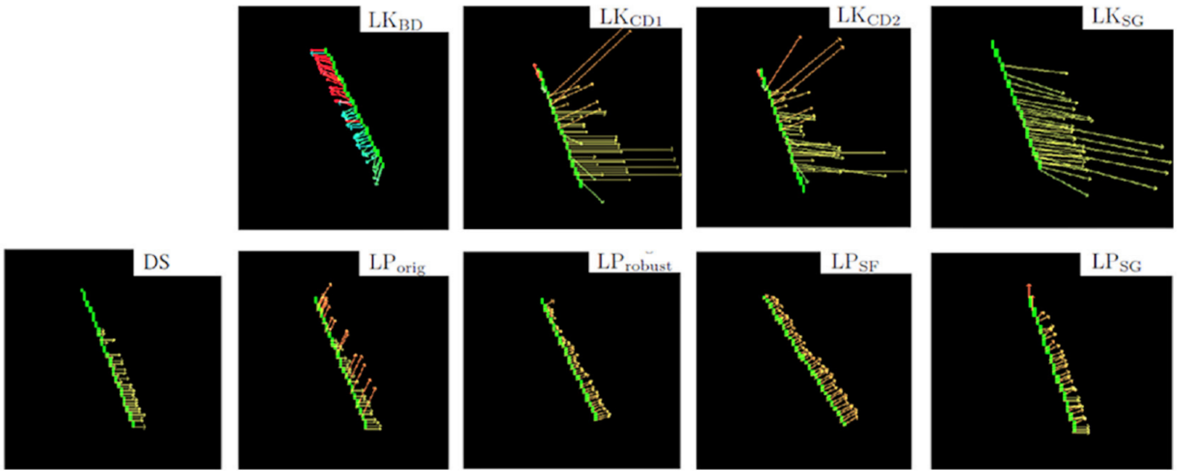


Figure 7-5: Flow estimate results in the synthetic rotating bar sequence of Ruckauer and Delbruck (2016). Results are shown from four variants of the Lucas-Kanade (LK) method and the local plane fitting (LP) method, and from the *DvsDirectionSelectiveFilter* algorithm (DS). Adapted from Ruckauer and Delbruck (2016).

Our implementation of STPF is well comparable to the Java-based LP$_{\text{robust}}$ algorithm in Ruckauer and Delbruck (2016), which is based on the same principles set out by Brosch et al. (2015). The LP$_{\text{robust}}$ algorithm performs reasonably accurate estimates at 4.51 µs computation time per event. However, the authors propose a Savitsky-Golay filter based version (LP$_{\text{SG}}$) of this algorithm which reduces computation time to 0.58 µs per event, and improves angular accuracy, though at the cost of magnitude accuracy. This may be an interesting alternative for achieving real-time performance of the STPF approach, if the magnitude accuracy can be increased.

7-4 Improving performance through filters

As part of the analysis we investigated two extensions in order to obtain better performance. These are implemented in the form of separate filters. First, a background activity filter is proposed to remove standalone events that are not clearly part of any moving structure. Second, a flow regularization filter was introduced to add coherence to the flow vectors, and remove unsupported flow vectors. In this section the working principles of the filters are described. Also, their performance is evaluated in comparison to results in Section 7-3.

7-4-1 Background activity filter

The software package jAER includes a functionality named BackgroundActivityFilter(). This simple but effective event filter rules out events that are not supported by direct neighbor events that have occurred within a specified time window. Specifically, if an event occurs, the filter checks the time of the last occurring events on neighbor pixels. If the time difference between the new event and the last neighbor event is smaller than a time threshold, the event is preserved for further processing. If not, the event is rejected. All events are, however, preserved to support future events.

In jAER this filter was quite effective at keeping noise events out, preserving mainly moving features. This favorably influences processing time in subsequent processes. Also, due to its simplicity it uses very little computational resources by itself. Therefore, the filter was incorporated into the MATLAB analysis as well to see how it affects optical flow estimate performance and processing time.

7-4-2 Flow regularization filter

Through existing event-based optical flow algorithms, local optical flow is mainly detected along moving edges. In this case, multiple single flow vectors are identified next to each other. Use may be made of this property to identify motion vectors with higher degrees of confidence, rejecting flow that is not supported by neighbor flow, or does not match with neighbor flow patterns.

A simple causal filter was designed that operates on each flow vector after identification. Figure 7-6 illustrates its working principle by example vectors. The filter is based on two main assumptions: (1) a moving edge generates more than one flow vector, and (2) flow vectors along an edge are similar in magnitude and orientation. This leads to two criteria: (1) a flow vector needs to be supported by at least one neighbor, and (2) the difference in orientation and magnitude must be smaller than a threshold. If a flow vector fulfills these criteria, the magnitude and orientation of the corresponding output flow vector are averaged over the input flow vector and its supporting vectors.

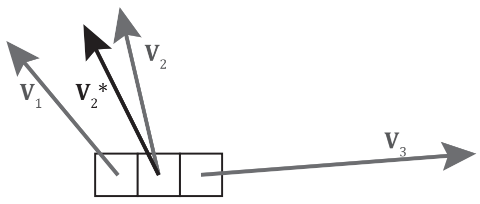


Figure 7-6: Working principle of the current flow regularization filter illustrated through 3 sequentially detected flow vectors. Flow vector 1 is detected first, but not supported by other vectors and therefore rejected. Vector 2 is supported by vector 1, which has similar magnitude and orientation, and is therefore preserved as vector 2*. Vector 3 is detected last and supported by vector 2, but has a different magnitude and orientation, and is rejected.

7-4-3 Effect of filters on flow estimation performance

The effect of extensions on the algorithm performance is demonstrated in this section. We repeat the analysis of the STPF algorithm on set 3 with each extension added separately. First, qualitative results are shown per filter by means of event and flow visualizations. Second, a quantitative comparison to results for the STPF algorithm presented in Section 7-3-2 is made.

Background activity filter

First we consider some details of the filter results, starting with the background activity filter. Figure 7-7 shows a subset of the events on the pixel grid, and the filtered events. We see that the filter effectively suppresses most stray events, preserving mainly clear features formed by events. However, some features are already seen to 'shrink', showing that there is a practical trade-off to be made between event rate and feature quality. Figure 7-8 shows for the complete dataset how the filter threshold affects event rate over time. The peaks in the event rate are due to camera movement. Note that these peaks are preserved well with a filter threshold above 10 ms, but that with 1 ms it appears that significant data loss occurs.

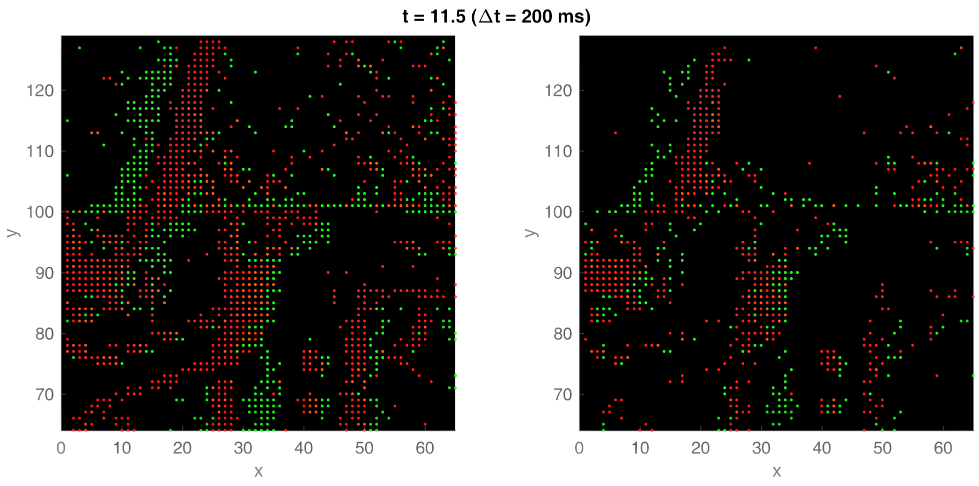


Figure 7-7: Effect of the background activity filter applied to the upper left quarter of the pixel grid in set 3. On the left, the unfiltered events are shown. The right image shows filtered events, using a threshold of 10 ms.

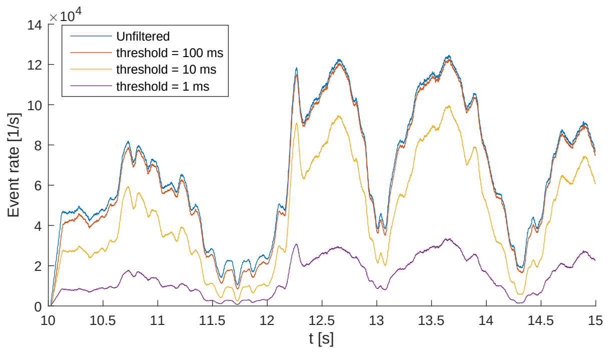


Figure 7-8: Background activity filter effect on event rate in set 3, with various threshold settings.

Flow regularization filter

Figure 7-6 shows two event and flow patterns obtained from the STPF algorithm, of which the right image results from applying the flow regularization filter. For clarity, the ground truth optical flow vectors are left out. Clearly, the filter reduces the amount of flow vectors significantly, and only the flow vectors that have sufficient support in the original flow pattern are preserved. As a result, most inconsistent and individual flow vectors are rejected, while the flow along edge-like features is preserved.

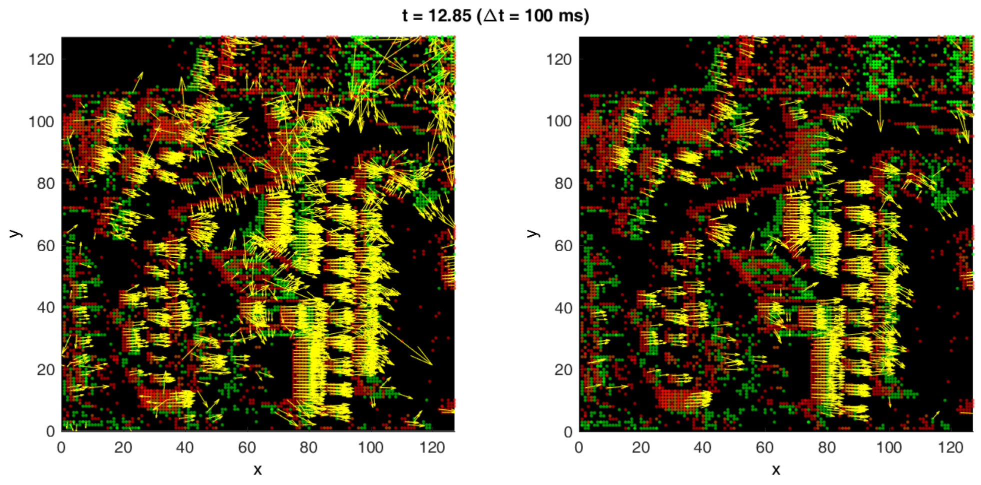


Figure 7-9: Effect of the flow regularization filter on optical flow estimates from the STPF algorithm applied to set 3. Both images show events (green = positive polarity and red = negative) and flow vectors (yellow) obtained during the same time window. The left image is the basic algorithm result; the right image shows the filtered flow vectors.

Quantitative results

Table 7-2 shows how the performance of the STPF algorithm varies when subjected to the different filters. We see that the background activity filter significantly reduces computation time. Overall, the error does appear to increase slightly. However, with other datasets and methods, this did not appear to be a consistent result; sometimes the error actually decreased.

On the other hand, the flow regularization filter does reduce errors. The filter is successful at rejecting several outliers, favorably affecting the output flow accuracy. The computational load penalty for this filter is small.

Table 7-2: Effect of applying filters (individually and combined) to the STPF algorithm evaluating event set 3. The background activity (BA) filter operates at a threshold of 10 ms. The flow regularization (FR) filter requires flow input to have at least one direct neighbor, which has a maximum orientation difference of 20° and a magnitude difference of 0.5 of the new flow vector magnitude

	AEE [pix/s]	AAE [deg]	APEE [pix/s]	t_C [µs]	η [%]
STPF	41.55	31.52	25.34	435.3	30.35
STPF with BA filter	43.11	29.22	27.34	**261.6**	22.56
STPF with FR filter	**35.71**	25.34	**20.50**	447.7	17.08
STPF with both filters	36.65	**23.91**	21.57	270.2	13.66

Chapter 8

Analysis of Visual Observable Estimation Methods

In this part of the analysis we evaluate the performance of five algorithms for computing visual observables from optic flow vectors. In three of these algorithms the FoE is estimated first, followed by the computation of ventral flow and divergence. The other two algorithms directly estimate ventral flow and divergence. We discuss briefly the influence of the number of flow vectors available and their spread. Then, qualitative and quantitative results are presented.

8-1 Algorithms for estimating the Focus of Expansion

In these algorithms the problem of estimating the FoE is separated from computing visual observables. These techniques are therefore generally applicable, and are not necessarily based on assumptions regarding the flow field structure. Hence, they would also be applicable to scenes where depth variations occur.

8-1-1 Probability mapping method

Introduced by Clady et al. (2014), the Probability Mapping (PM) method is a technique for estimating the most likely location of the FoE. This method was introduced in Section 4-3-2 and is detailed further in this section.

Let $\mathbf{V} = [u,\, v]^T$ be an incoming flow vector and $\mathbf{d}' = [x - x',\, y - y']^T$ the displacement vector from the flow vector origin to any location (x', y') on a map. $\mathbf{V}$ can be a normal vector, hence providing no direct indication of the location of the FoE. However, $\mathbf{V}$ bounds a *half-plane* in the map in which the FoE location should be. The locations (x', y') that lie in this half-plane, are identified through:

$$\mathbf{d}' \cdot \mathbf{V} > 0 \tag{8-1}$$

Through addition of multiple flow vectors, an area on the map is bounded where the FoE is located. To account for shift in the FoE over time, an exponential decay is applied to the full map.

Two main modifications are made with respect to this original method. First, during general motion the approach should work for both expansion and contraction. To achieve this, we assign *negative probability* to the map section where the FoE cannot exist. With this addition, the location with the largest *absolute* probability indicates either the FoE or the FoC. Second, PM includes a modification that allows a trade-off of computational complexity and accuracy. A full-size probability map would require maintaining and updating a 128×128 array. By downsampling the resolution, the array size can be reduced, requiring less computations to update the map. In this work the array is downsampled by a factor 4.

8-1-2 Integration method

In the Integration Method (IM) by Paz Gomes Verdugo (2015) the location of the FoE is sought by finding the location where the *magnitude* of flow velocity is minimal. Similar to the PM algorithm, this is a map-based approach. Two independent one-dimensional arrays $M_u(x), M_v(y)$ map the average u- and v- components of flow vectors separately to the corresponding x and y respectively. For these maps no downsampling is necessary.

For each incoming flow vector, the u- and v-maps corresponding to the flow location are adjusted through a low-pass filter. After updating the maps, the FoE is found from the location where the low-pass filtered absolute velocities $|M_u(x)|$ and $|M_v(y)|$ are minimal.

8-1-3 Cross-product method

In Section 4-3-1 a sequential method for estimating the FoE was discussed. Proposed by Paz Gomes Verdugo (2015), the method is based on the assumption that, at the FoE, all flow vectors *intersect.* In this analysis it is implemented as the Cross-Product (CP) method.

Given a flow vector $\mathbf{V}$ and its displacement vector from the FoE, $\mathbf{d_f} = [x - x_f,\, y - y_f]^T$, the condition for the FoE implies that $\mathbf{d_f} \times \mathbf{V} = 0$. Or, equivalently:

$$uy_f - vx_f = uy - vx \tag{8-2}$$

With several sequential flow vectors available, (8-2) extends to an overdetermined system of equations. The most likely values for x_f and y_f are then estimated through ordinary least-squares. Currently this is done for each new incoming flow vector, solving for the normalized velocities using a fixed number of previous flow vectors. This number is at present the only parameter of the method.

8-1-4 Estimating visual observables using the Focus of Expansion

The methods discussed in this section compute the FoE location, which is an in-between parameter necessary for obtaining the visual observables. Knowing the FoE location, ϑ_z is computed through the following relation:

$$\vartheta_z = \frac{(x - x_f)u + (y - y_f)v}{(x - x_f)^2 + (y - y_f)^2} \tag{8-3}$$

This equation is based on the approach used by Clady et al. (2014) in conjunction with the PM algorithm.

For these approaches, the ventral flows ϑ_x, ϑ_y are computed separately as the moving averages of $\hat{u} = u/f$ and $\hat{v} = v/f$. To compute these averages, a recursive low-pass filter is employed. Thus, ϑ_x is computed through:

$$\vartheta_x(k+1) = \vartheta_x(k) + (-\hat{u}(k+1) - \vartheta_x(k))\,\frac{t(k+1) - t(k)}{k_t} \tag{8-4}$$

and ϑ_y through an equivalent form of this relation. k_t is the low-pass filter time constant. Hence, for each of the methods in this section, the ventral flow estimates are equal.

8-2 Algorithms for simultaneous estimation of visual observables

In these two techniques, the visual observables are estimated simultaneously from a set of optical flow vectors. It is therefore not necessary to explicitly estimate the FoE and to derive divergence separately for these.

8-2-1 Estimating a planar flow field

A technique seen frequently in recent frame-based visual landing experiments is to directly estimate the parameters of a planar flow field (Alkowatly et al., 2015; De Croon et al., 2015; Ho & De Croon, 2016). In Section 2-1 two models for a planar flow field were proposed and discussed. The model in Eq. (2-6) provides a structure that directly relates local flow components to ventral flows and divergence. Hence, for this approach it is not necessary to first estimate the location of the FoE, and then compute divergence and ventral flow separately. This model forms the basis for the Planar Flow (PF) algorithm.

Similar to the CP method, this approach uses a fixed number of sequential flow vectors. Using these flow vectors, the flow field parameters $\vartheta_x, \vartheta_y, \vartheta_z$ in Eq. (2-6) are computed through linear least-squares regression. Since there is a common parameter ϑ_z in both equations, the least-squares system is composed as follows:

$$\begin{bmatrix} 1 & \hat{x}_1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & \hat{x}_n & 0 \\ 0 & \hat{y}_1 & 1 \\ \vdots & \vdots & \vdots \\ 0 & \hat{y}_n & 1 \end{bmatrix} \begin{bmatrix} \vartheta_x \\ \vartheta_z \\ \vartheta_y \end{bmatrix} = \begin{bmatrix} \hat{u}_1 \\ \vdots \\ \hat{u}_n \\ \hat{v}_1 \\ \vdots \\ \hat{v}_n \end{bmatrix} \tag{8-5}$$

using the normalized equivalents $(\hat{x}, \hat{y})$ and $(\hat{u}, \hat{v})$. From the flow field parameters, ventral flows and divergence are immediately available.

8-2-2 Estimating a planar normal flow field

The PF algorithm holds when optical flow is fully observable. However, when only normal flow can be observed, Eq. (2-6) no longer holds. Therefore, we introduce an extension of PF based on the fundamental assumption that *only normal flow* can be detected. This approach will be referred to as the Planar Normal Flow (PNF) algorithm.

Let $\mathbf{n} = [n_1,\, n_2]^T$ be a unit vector normal to an edge, $\hat{\mathbf{V}}_n = [\hat{u}_n,\, \hat{v}_n]^T$ the observable normal flow vector in the direction of $\mathbf{n}$, and $\hat{\mathbf{V}} = [\hat{u},\, \hat{v}]^T$ the corresponding optical flow vector. Since $\hat{\mathbf{V}}_n$ results from a projection of $\hat{\mathbf{V}}$ to $\mathbf{n}$, the magnitude of the projection is:

$$\left\|\hat{\mathbf{V}}_n\right\| = \hat{\mathbf{V}} \cdot \mathbf{n} = \hat{u}n_1 + \hat{v}n_2 \tag{8-6}$$

The flow field structure in Eq. (2-6) is then substituted into Eq. (8-6), obtaining a relation of the flow field to the normalized velocities:

$$\left\|\hat{\mathbf{V}}_n\right\| = -\vartheta_x n_1 - \vartheta_y n_2 + (\hat{x}n_1 + \hat{y}n_2)\,\vartheta_z \tag{8-7}$$

or, since $n_1 = \hat{u}_n / \left\|\hat{\mathbf{V}}_n\right\|$, $n_2 = \hat{v_n} / \left\|\hat{\mathbf{V}}_n\right\|$:

$$\left\|\hat{\mathbf{V}}_n\right\|^2 = -\vartheta_x \hat{u}_n - \vartheta_y \hat{v}_n + (\hat{x}\hat{u}_n + \hat{y}\hat{v}_n)\,\vartheta_z \tag{8-8}$$

Similar to the PF algorithm, the original flow field parameters can now be obtained through a linear least-squares solution, using a fixed number of sequential flow vectors.

8-3 Datasets

The datasets used for evaluating visual observable estimation methods consist of optical flow vectors identified using the STPF algorithm in order to represent realistic optical flow. Next to this, for each dataset, two ground truth velocity vector sets are computed:

- From the Optitrack validation measurements, ground truth visual observables are determined. Then, for each flow vector identified from the algorithm, a ground truth vector is computed based on the event's location and timestamp.
- As discussed before, in many cases optical flow can only be identified normal to a moving edge. Therefore, we also generate a dataset of ground truth normal flow, by projecting the ground truth vectors onto the estimated flow vectors.

Currently this strategy is applied to generate datasets based on set 1 and 3 discussed in Section 6-3, hence obtaining 2 sets with ground truth flow, ground truth normal flow, and flow estimates.

8-4 Flow statistics

During implementation it was found that the accuracy of visual observable estimates decreased in parts of the dataset with limited event rate. Closer examination revealed that in these parts, either too few flow vectors were available, or that they were distributed insufficiently across the field of view. In fact, this was related to the motion and position of the camera. If the motion is too small, the event rate is limited and optical flow cannot be estimated with good accuracy. Also, if insufficient texture is recorded, optical flow is either not found, or only found in certain parts of the retina. If flow is only available when grouped closely together, small errors in local flow have a larger influence on the visual observable estimates than if they are well spread out.

In order to quantify these effects we introduce and monitor three statistics: *flow detection rate*, and *flow position variances* for both x and y. These statistics are computed in a sliding window for each new flow vector. If the position variance is small, there is insufficient spread in flow vectors.

8-5 Results

We first consider in slight detail two special cases of motion, contraction and ventral motion, and highlight some detailed results through plots of the validation data over time. Then, error metrics obtained for the estimate datasets are presented. The accuracy of the methods is assessed through the *mean absolute error* and the *error variance.*

8-5-1 Detailed flow sequences

Contracting flow in set 1

Figure 8-1 shows results for a part of set 1 containing contraction with the focus of expansion to the left. Flow detection statistics (rate and positional variance) are shown as well. Note that in this set, little texture can be perceived until $t = 12.4$, as can be seen in the flow statistics. In particular, the flow position variance is rather low.

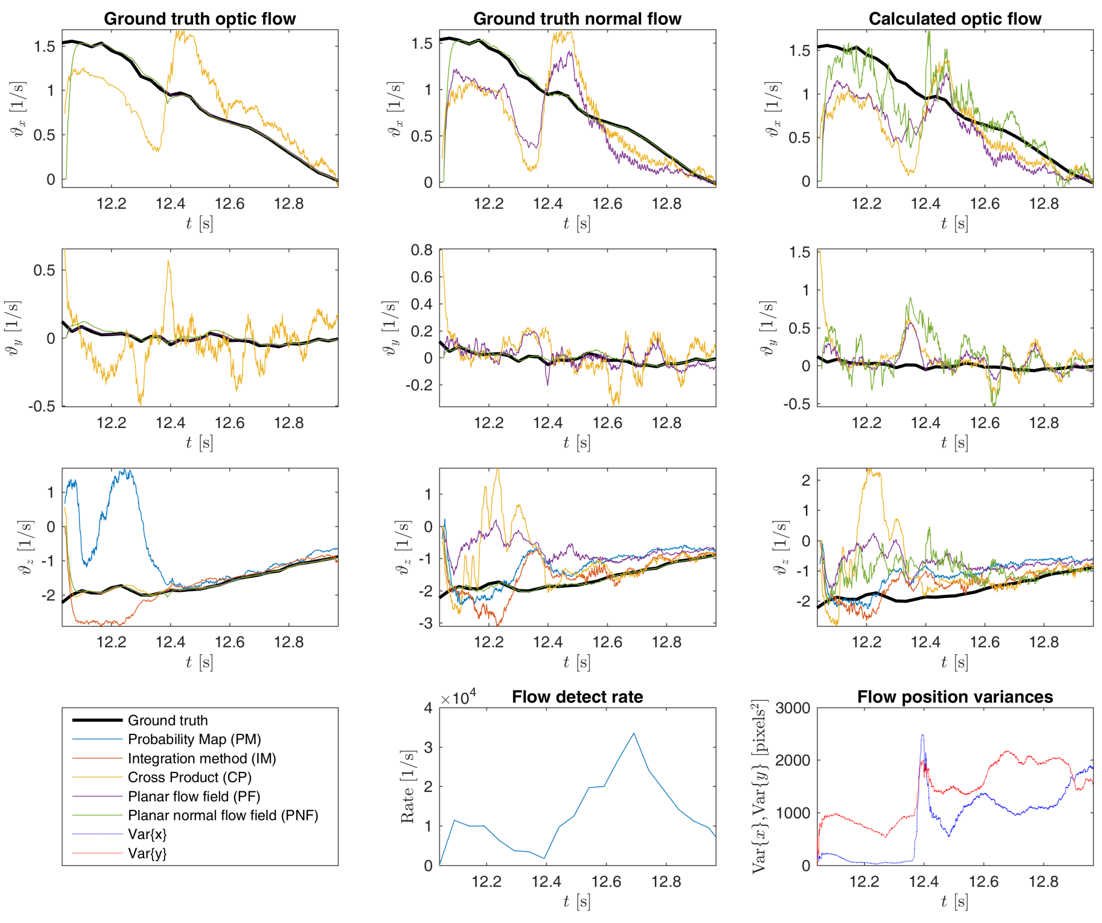


Figure 8-1: Flow field parameters estimated by different methods on contracting flow in set 1 (checkerboard texture). Flow estimate statistics are shown as well.

Where the flow detection rate and variances are high, ground truth divergence results for all methods are accurate. Ventral flow results are off, however, for the methods where ventral flows are estimated independently from divergence (PM, IM, and CP).

When ground truth normal flow is analyzed, most algorithms lose accuracy, even with good flow coverage. The PF and PM methods also under-estimate divergence. The PF method also fails to correctly estimate ventral flows. In contrast, the normal flow variant (PNF) still provides very accurate results for both ventral flow and divergence.

In the real estimates, performance of all methods degrades significantly. Of the used methods, the PNF method still seems to perform the best on estimating divergence, and recognizing the global trend in ϑ_x. However, in ϑ_y the algorithm shows intensified noise peaks compared to the other methods.

Ventral flow in set 3

In Figure 8-2 results are presented for a subset of set 3 including ventral motion. In the ground truth flow results, most methods perform well at estimating ventral flows. However, a particularly interesting result is seen for the PNF method: there are significant peaks visible that are not present in either the estimated flow results or the normal ground truth flow results. Closer inspection shows that this occurs due to ambiguity in the flow direction and magnitude. Also interesting is that at the same time, the divergence estimate is still highly accurate. This phenomenon is less visible in the calculated flow results, although ventral flow estimates are more noisy than most results. The divergence is slightly off here.

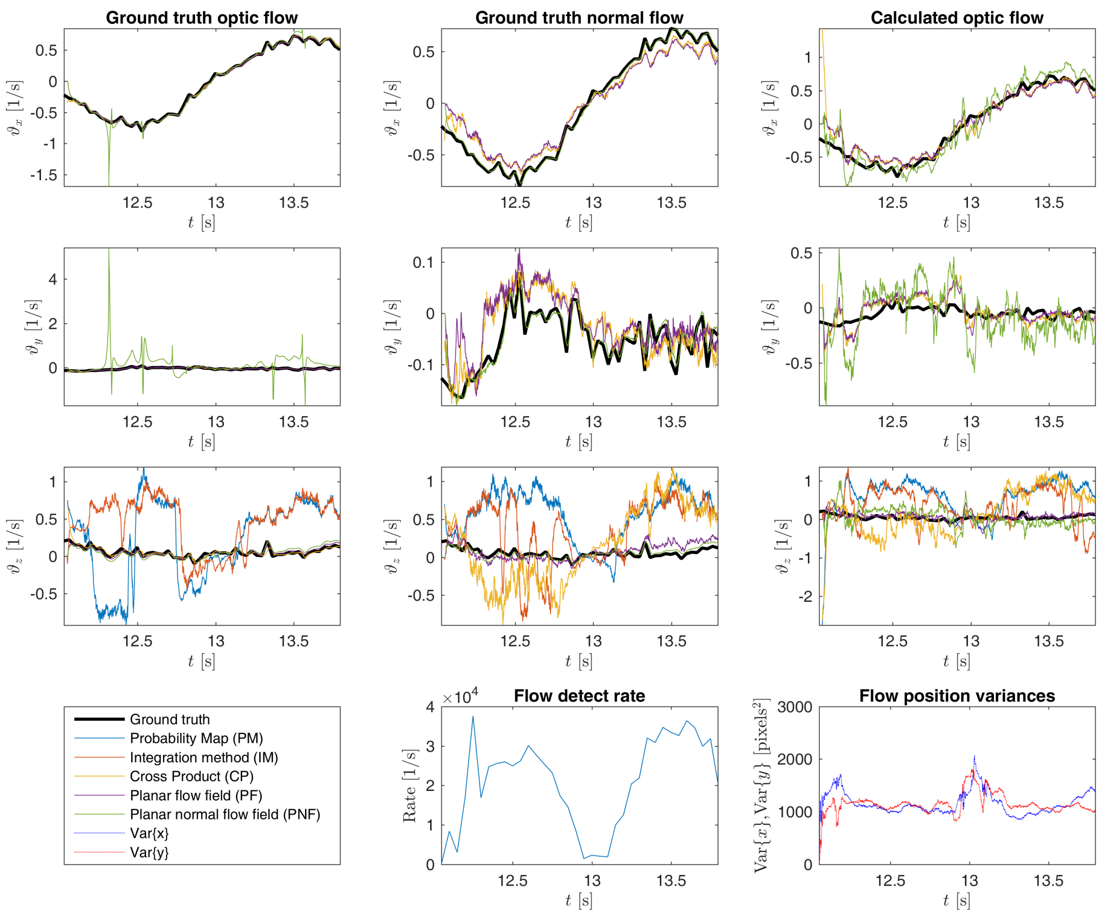


Figure 8-2: Flow field parameters estimated by different methods while perceiving ventral flow in set 3 (roadmap texture).

Note also that the methods that are limited to the field of view of the camera (PM and IM) provide incorrect divergence estimates due to wrong estimation of the FoE, even with ground truth flow. Since ventral flows are estimated independently for these methods, these estimates are not affected and quite accurate, even for calculated flow.

The CP method divergence estimates are accurate for ground truth flow, but fail when normal flow is evaluated. This result is also seen in the calculated flow.

Lastly, the PF method suffers little from normal flow in this case. Divergence estimates are reasonably accurate in the calculated flow. Ventral flow results are slightly off: ϑ_x is slightly underestimated. Since ϑ_y is slightly over-estimated when ϑ_x is large, this might also be due to slight mis-alignment of the camera and Optitrack reference frames.

8-5-2 Quantitative results and computational effort

Table 8-1 shows the results from the analysis of the full sets 1 and set 3. Here only the flow estimates from the STPF algorithm are considered. Note that in these results, no correction is incorporated for flow detection rate or position variances. Therefore, these results should be interpreted with care.

Table 8-1: Visual observable estimation errors (mean absolute error and variance) when applied to the flow estimate dataset. Computation times per event (t_C) are shown as well.

	ϑ_x [1/s]		ϑ_y [1/s]		ϑ_z [1/s]		t_C [µs]
	Mean	Var	Mean	Var	Mean	Var	
Set 1							
PM	0.2384	0.1211	0.2671	0.1358	0.2856	0.1506	40.74
IM	0.2384	0.1211	0.2671	0.1358	0.3966	0.2900	**21.79**
CP	0.2384	0.1211	0.2671	0.1358	0.4766	0.4437	40.15
PF	**0.2029**	**0.09058**	**0.2125**	**0.09156**	**0.2553**	**0.1294**	55.51
PNF	0.3174	0.1502	0.2727	0.1475	0.3553	0.2854	57.23
Set 3							
PM	0.1409	0.03556	0.17	0.06707	0.5778	0.3059	16.19
IM	0.1409	0.03556	0.17	0.06707	0.4876	0.3197	**8.69**
CP	0.1409	0.03556	0.17	0.06707	0.4338	0.2859	16.10
PF	**0.121**	**0.02384**	**0.1303**	**0.03234**	**0.2363**	0.1458	21.96
PNF	0.1857	0.06045	0.2238	0.07598	0.2757	**0.1368**	22.83

Quantitatively, the highest accuracy is seen mainly by the PF algorithm for both ventral flows and divergence, showing errors with limited variation. Interestingly, this result is not visible in the qualitative part shown in 8-1, where the algorithm performs less than most other methods. The PNF approach shows the lowest accuracy and high error variance in estimating ventral flows, but reasonable results for divergence estimates.

In set 1, the map-based methods PM and IM perform reasonably well, since in general, the FoE lies within or close to the field of view. However, set 3 contains a section of pure ventral flow which was highlighted in Figure 8-2. This clearly increases the errors for these methods, mainly for PM.

The computational load of the PF approach is, however, relatively high. It is possible to reduce computational load by incorporating a smaller number of previous events per fit. The IM approach appears the least demanding. This is favorable since no downsampling is applied to this method.

Chapter 9

Discussion of Preliminary Results

A preliminary comparison framework is set up for evaluating event-based optical flow performance. Using a DVS for recording event sequences and an Optitrack system for pose tracking, datasets are constructed representing the basic motion types relevant in landing maneuvers for MAVs. These are used for evaluating the performance of event-based methods for estimating optical flow and visual observables. This chapter summarizes and discusses the results obtained in the preliminary work. At the end of this chapter, the relation between the preliminary results and the current work is described.

9-1 Datasets

We recorded three datasets with different motion types using two types of texture. In the present datasets, several improvements are still possible. First, the datasets are relatively long and consist of parts where spatial event coverage and event rates are limited. A better division into smaller sections with clear characteristics may provide a better overview of what factors influence the algorithm performance. Second, values for ventral flow and divergence in this dataset are relatively high when compared to values seen in landing experiments (Herissé et al., 2012; Ho & De Croon, 2016). Third, as can be seen in Figure 6-3, the spatial and temporal resolution in the currently recorded datasets are limited. This can be remedied by changing settings for the Optitrack logging process.

9-2 Performance of optical flow methods

For estimating local optical flow, the STPF method shows the most desirable behavior. It produces consistent and accurate flow estimates along edges. The runner-up is the NES algorithm, which had the best results in Marta's work (Paz Gomes Verdugo, 2015). LK and DSF produce inconsistent flow, even along clear edge features. In related work (Ruckauer & Delbruck, 2016), more accurate results for orientation were produced for LK than in the presented method, though the optical flow magnitude is generally incorrect.

The filter extensions show to be simple but powerful additions. While the background activity filter limits event throughput and thus computation time, the edge flow filter improves overall flow quality and rejects inconsistent flow estimates. They are compatible with all local flow estimation methods and are thus useful to incorporate in the on-board implementation.

Computation times for the algorithm are high at this point. With the MATLAB implementations, real-time performance is not feasible with common event rates. However, the implementations are not optimized. Also, the laptop used for computations was operating with limited CPU (10%) to prevent overheating. Lastly, implementing in C will greatly reduce computational effort. On the other hand, an on-board processor has less computation capability compared to the laptop. A final judgment can thus only be made when the algorithms are tested on-board.

For the STPF approach, several means are available to reduce computational load. It can be done through (1) a faster approach to plane estimation (e.g. the Savitsky-Golay filter approach by Ruckauer and Delbruck (2016)), (2) a limited number of plane fit iterations, which is less computationally demanding, but also less robust, or (3) applying the background activity filter to reduce the amount of evaluations.

9-3 Performance of visual observable estimation methods

Using the approaches described in this work, visual observable estimation can be performed reasonably well for pure ventral flows. However, when flow patterns become more complex, as in the case of contracting/expanding flow, estimating ventral flows and in particular divergence correctly is challenging. Quantitatively the errors are too high to commence with implementation. In comparison, Herissé et al. (2012) applied a divergence setpoint of 0.1 during experiments and 0.5 during simulation, and in the simulations by Ho and De Croon (2016) a setpoint of 0.3 was used.

In combination with estimated optical flow, the best quantitative performance is seen by the PF method, which provides simultaneous estimates of ventral flow and divergence. This is seen to perform better than separately computing ventral flows and divergence. However, these results should be used with care, since the full datasets has several regions with limited event rate and flow coverage, which leads to unpredictable behavior. Through a better selection of datasets for evaluation, these quantitative results may become more relevant. Alternatively, through setting thresholds for flow detection rate and position variance, it may be possible to automatically disregard flow data when these statistics are too low.

As expected, mapping-based methods (PM, IM) fail even for ground truth optical flow vectors when the FoE is outside the field of view. When this happens, mainly divergence estimates are off. This can occur quite often such as in pure ventral flow. Therefore, map-free methods are preferable. Alternatively, a map-based method may be considered that allows for extrapolating the FoE location.

Most methods underestimate divergence when only the normal component of optical flow can be observed. This is sensible, since normal flow has a smaller magnitude than its 'true' optical flow equivalent. In an attempt to overcome this issue, the PNF method was introduced. This method is accurate for ground truth normal flow, but brings significant noise with it when applied to flow estimates, and fails to estimate ventral flows when flow is highly ambiguous.

The computation times necessary for the discussed methods for estimating visual observables are relatively low compared to the optical flow estimation methods. However, in the current implementation they scale with the amount of flow vectors estimated. A less demanding approach would be to split up the field of view into sections, where for each section an average flow vector is maintained, and updated with each newly estimated flow vector in that section. Then, we can periodically compute visual observables from this 'average flow field' at a flexible sampling frequency. This may also favorably affect robustness.

9-4 Relation to results presented in the paper

The analysis presented in this part serves as the basis for the final thesis work in several ways. Mainly, the algorithms that are presented in the paper in Part I are based on the best performing methods in this analysis.

Our final plane fitting algorithm is highly based on the STPF algorithm. Improvements have primarily been made by reducing the number of parameters to two instead of three to reduce computational effort, and by incorporating timestamp-based clustering to obtain more accurate estimates for a wide range of optical flow magnitudes. Eventually, the background activity filter and flow regularization filters are not incorporated to limit the number of parameters in the full perception pipeline. In addition, with the settings presented in this preliminary analysis, the background activity filter prevents perception of low optical flow velocities, which in turns prevents sensing of visual observables at low speed.

The previous analysis demonstrated that, for estimating visual observables, the PNF algorithm performs the best on ground truth normal flow, circumventing the aperture problem as opposed to the PF algorithm. However, it is more sensitive to noise and it showed near-singular behavior during pure ventral flow. To overcome these issues, grouping by direction is introduced, enabling assessment of the quality of optical flow per direction. The weighted least-squares estimator presented in the paper derives from Eq. (8-7). In addition, the introduction of a confidence metric is partly the result of the datasets used in this preliminary analysis.

Furthermore, the datasets used here highlight several issues that are taken into account in currently used datasets. The 'flashes' that occur in the preliminary datasets appear to be related to software settings that control the DVS hardware. With the current settings (see Section 12-4), the flashes are no longer observed. In addition, the Optitrack system emits strobing infrared light in order to track objects fitted with passive markers. However, the DVS is most sensitive in the infrared spectrum, which led to interference in prior work by Censi et al. (2013). An altered experimental setup (discussed in Section 11-1) does not require infrared strobing. This improved the quality of the datasets significantly.

Part IV

Appendices

Chapter 10

Calibration of the Dynamic Vision Sensor

In most cameras, the image perceived by the sensor is not a perfect representation of the pinhole camera model. Therefore, calibration is performed to measure the camera *intrinsic parameters* (focal length, principal point, skew) and to correct for *lens distortion.* The DVS is no exception. Figure 10-1 shows events obtained from a checkerboard surface. The normally straight lines of the checkerboard are represented with some curvature by the event locations, indicating radial distortion.

However, a normal calibration procedure involves computer analysis of images shot by the camera (Klette, 2014). This is not a trivial approach for purely event-based cameras such as the DVS, since they are not capable of measuring absolute brightness, requiring change to drive their output.

We propose an approach that approximates a normal procedure by reconstructing artificial images from events. To accomplish this, events are recorded from blinking patterns, hence allowing state-of-the-art calibration tools to be used.

10-1 Calibration model

In this work we apply the models as defined in the MATLAB Camera Calibration Toolbox (Bouguet, 1999). In the calibration process, the parameters are estimated of a model that relates world points in an inertial world reference frame $(X_{\mathcal{W}}, Y_{\mathcal{W}}, Z_{\mathcal{W}})$ to pixel locations (x, y). In the pinhole camera model, this includes two linear transformations correcting for the camera *extrinsic parameters* (the camera pose with respect to the world reference frame) and the *intrinsic parameters* (Klette, 2014). Further, to correct for lens distortion, a nonlinear transformation is used.

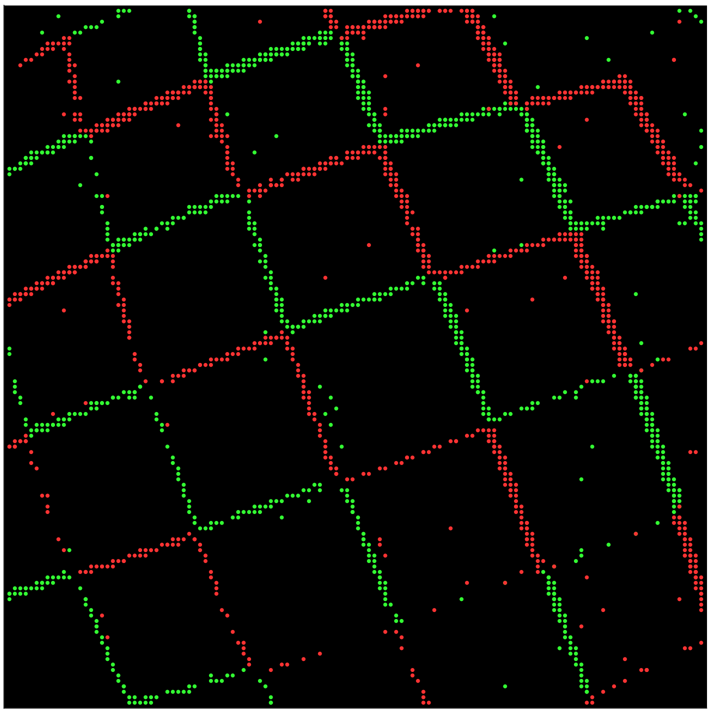


Figure 10-1: Lens distortion observed in DVS event output when a checkerboard pattern. Events are accumulated over a duration of 100 ms. Green dots are events with positive polarity, red dots indicate negative polarity.

10-1-1 Intrinsic parameters

For the presented research, we are mainly interested in obtaining the intrinsic parameters of the DVS. The points in the camera reference frame can be related to (x, y) through the *intrinsic matrix*, which contains the intrinsic camera parameters: focal length f, principal point (x_p, y_p), and skew s.

$$Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & x_p \\ 0 & f_y & y_p \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{\mathcal{C}} \\ Y_{\mathcal{C}} \\ Z_{\mathcal{C}} \end{bmatrix} \tag{10-1}$$

In this definition, f contains the ratio of the lens focal length to the pixel dimensions, but is for simplicity referred to as the focal length. It is assumed constant for both x-coordinates and y-coordinates, since both the DVS pixel grid and the individual pixels have equal length and width. Note that through Eq. (10-1) we can relate the pixel array locations to the normalized coordinates (x, y), hence relating the pixel coordinates to the equations derived in Chapter 2:

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} f_x & s & x_p \\ 0 & f_y & y_p \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10-2}$$

10-1-2 Lens distortion

Lens distortion causes image points projected on the pixel array to deviate from the pinhole camera model. Distortion can be separated into a *radial* transformation and *tangential* transformation. The distortion model in the MATLAB Camera Calibration Toolbox is based on the Brown-Conrady model (Brown, 1966). The transformation from a pixel coordinate (x, y) to its undistorted equivalent (x_u, y_u):

$$\begin{bmatrix} x & y \end{bmatrix}^T = \begin{bmatrix} x_u & y_u \end{bmatrix}^T \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + \begin{bmatrix} \Delta x_t & \Delta y_t{}^T \end{bmatrix} \tag{10-3}$$

where the multiplication factor is the radial transformation, and $\Delta x_t \ \Delta y_t{}^T$ is the tangential transform:

$$\begin{bmatrix} \Delta x_t \\ \Delta y_t \end{bmatrix} = \begin{bmatrix} 2k_4 x_u y_u + k_5 \left(r_u^2 + 2x_u^2\right) \\ k_4 \left(r_u^2 + 2y_u^2\right) + 2k_5 x_u y_u \end{bmatrix} \tag{10-4}$$

In the above equations, $r_u = \sqrt{(x_u - x_p)^2 + (y_u - y_p)^2}$ is the radial distance to the principal point. $k_1, \ldots, k_5$ are the model parameters. Depending on the application, it may be sufficient to use only the radial distortion correction.

10-2 Calibration approach

In a standard calibration approach, several images are captured of a high contrast planar pattern under varying angles. In most standard computer tools, a checkerboard pattern is used. In each image, a world point grid is then estimated from the pattern by locating the corners of the checkerboard squares. Knowing the metric square dimensions and having several images available, one can then fully estimate (1) the extrinsic parameters in each image, (2) the camera intrinsic and distortion parameters (Bouguet, 1999).

Since the DVS does not provide absolute brightness values, different means to locate world points need to be found. Huber (2014) (see also Mueggler et al. (2014b)) performed a calibration procedure by recording events viewing a LED screen, which shows dot grid. Due to the nature of LED screens, high-frequency blinking can be observed from the screen using the DVS without manually hiding and showing the pattern. Events indicating the dots directly provide the world points for camera calibration. Decent calibration results were obtained for a 2.8 mm focal length lens, but due to poor viewing angles from the LED screen, the procedure was not successful for a 3.5 mm lens. Mueggler et al. (2014b) published a C++ ROS driver online[1] that allows calibration with a blinking LED grid. This approach would, however, require manufacturing of a calibration setup for these LEDs.

The approach used in this work is similar to Huber (2014). We record events from the DVS viewing a LCD computer monitor using the accompanying software jAER. However, instead of using a 'static' pattern of dots, we use a checkerboard pattern that is flashed several times against a static black background. We then reconstruct an 'artificial image' of the checkerboard by accumulating all events per pixel location, regardless of polarity. The resulting

[1]Available at `https://github.com/uzh-rpg/rpg_dvs_ros`

checkerboard images are then used to locate world points and estimate camera parameters through the standard calibratino methodology in the MATLAB Camera Calibration Toolbox.

10-3 Calibration procedure results

We composed 30 artificial images from DVS events viewing a flashing checkerboard pattern under various angles. For each image the pattern was flashed 20 times to clearly distinguish the pattern. Table 10-1 shows the estimated intrinsic and distortion parameters. Tangential distortion coefficients were found to be small (≈ 0.0015) and were eventually not included in the model. Further, if was found that a two-parameter version of the radial distortion model was more appropriate, to prevent extrapolation error at pixels in the corners of the field of view. Note also that skew is rather small. The principal point is slightly off the center of the image.

According to the DVS128 specifications (see Table 3-1) the camera should have a pixel size of 40 µm. The lens has a focal length of 4.5 mm. Computing the focal length scaling factor using these values results in a value of 112.5, which compares reasonably well to the lengths estimated by calibration.

Table 10-1: Calibration results for the DVS.

Focal lengths	f_x, f_y	115.3, 114.8
Principal point	x_p, y_p	76.70, 56.93
Radial distortion coefficients	k_1, k_2	-0.2857, 0.0829
Mean reprojection error		0.2020

To validate the calibration results, we examined the image estimates, corresponding world points, and the effect of undistorting the image. The latter was performed using the image undistortion function of the MATLAB toolbox. Figure 10-2 shows three images obtained from the reconstruction process, the world point estimates, and their undistorted counterparts. World point locations can be estimated with good accuracy, even with the limited resolution of the DVS. The corrected images are obtained by normal image undistortion (i.e. not by displacing events). The checkerboard patterns are rectified quite well.

10-4 Undistortion of events

Undistorting point coordinates from events is possible using Eq. (10-3). However, since the mapping provided by this equation is nonlinear, computing the undistorted event coordinates for each single event is a relatively complex procedure. Instead, as part of the calibration procedure, we solve Eq. (10-3) for all pixel locations and store the solutions in a look-up table. Hence, a *pixel undistortion map* $\mathcal{D} : (x_d, y_d) \to (x, y)$ is obtained.

Good correction results are observed when considering the undistorted event positions. Figure 10-3 shows the same events shown in Figure 10-1, in comparison to the undistorted events. The checkerboard lines formed by the events appear close to straight. Note also that the undistorted pixel locations are significantly further away from the principal point. Hence,

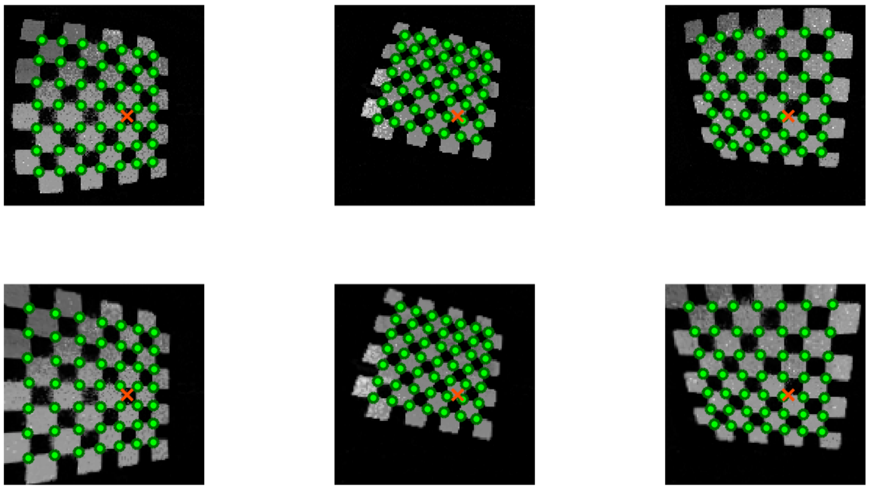
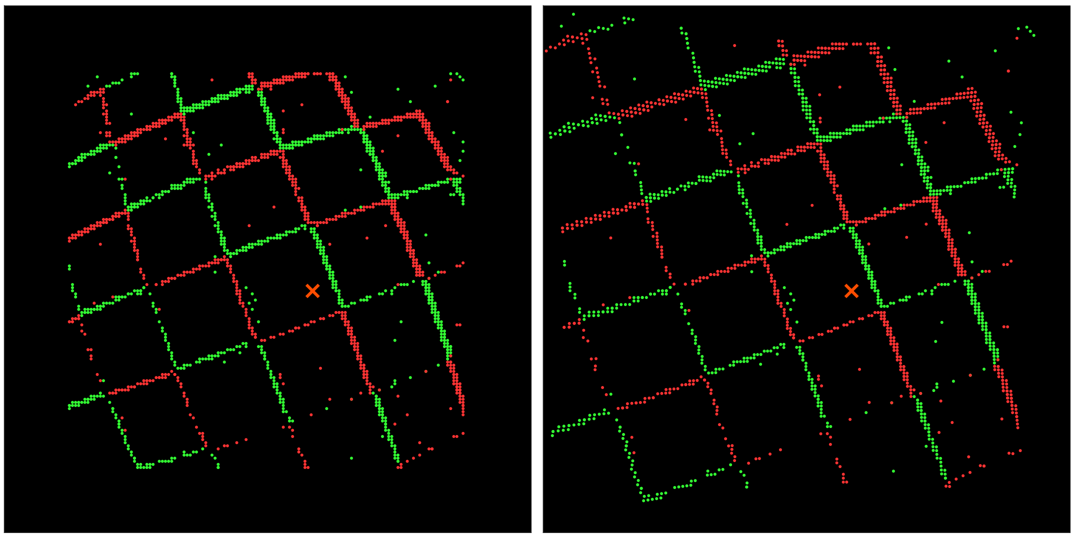


Figure 10-2: Reconstructed images and world points obtained from flashing the checkerboard pattern. The top row shows the input (distorted) images. The images on the bottom row are undistorted based on the calibration result. The brightness of a pixel corresponds to the number of events at that location. The green markers indicate identified world points and the red cross is the estimated location of the principal point.

at the borders of the chip, optical flow estimates from undistorted pixels are significantly larger than when the distorted pixel locations are used.



(a) Uncorrected events with distortion **(b)** Corrected events

Figure 10-3: Comparison of (a) the scene with uncorrected events in Figure 10-1 to (b) corrected, undistorted events. The scale of the uncorrected scene in (a) is adjusted to match the corrected scene. The principal point is shown as a red cross.

10-5 Undistortion of event-based optic flow

In a regular frame-based optic flow estimation procedure, the full image is undistorted before applying the flow estimation algorithm. Hence, the output flow is not affected by lens distortion. For event-based flow estimation, however, a different approach is necessary. Event positions are normally represented by integer coordinates. Undistortion of these coordinates results into floating point pixel locations. For some algorithms, including the STPF algorithm and the final approach applied in the paper, floating point pixel locations can be directly incorporated to obtain corrected optical flow output. In STPF one can use integer pixel locations for gathering events in the space-time window, but use the corrected floating point pixel locations for computing the plane. This directly yields undistorted optical flow. However, for algorithms that rely much more on pixel indexing, such as NES, a rather tedious conversion needs to be performed.

For such algorithms an alternative approach is derived where optical flow is corrected through a look-up table approach, similar to the event position mapping. Note that it is not applied in the paper, but is presented here for completeness and reference for possible follow-up research.

The foundation of this approach is the orthogonal system solution of the STPF method described by Eq. (7-4). In the approach, the assumption is made that each flow vector is equivalently described by a plane. Eq. (7-4) can then be inverted to compute the hypothetical slopes of such a plane (assuming $c = 1$):

$$\begin{bmatrix} a_d \\ b_d \end{bmatrix} = -\frac{1}{{u_d}^2 + {v_d}^2} \begin{bmatrix} u_d \\ v_d \end{bmatrix} \tag{10-5}$$

In its simplest form, a tangent plane can be formed by the (distorted) flow vector origin $\mathbf{P}_{0_d} = [x_d, y_d, t]^T$ and two other space-time points. Let these points be the direct neighbors in the pixel grid $\mathbf{P}_{\Delta x_d} = [x_d - 1, y_d, t - \Delta t_{\Delta x}]^T$ and $\mathbf{P}_{\Delta y_d} = [x_d, y_d - 1, t - \Delta t_{\Delta y}]^T$. Since these points are at unit distance from the flow origin, the hypothetical time values of these points $\Delta t_{\Delta x}, \Delta t_{\Delta y}$ are equal to the plane slopes:

$$a_d = \frac{\Delta t}{\Delta x}, \quad b_d = \frac{\Delta t}{\Delta y} \tag{10-6}$$

Now the undistorted point coordinates $\mathbf{P}_0, \mathbf{P}_{\Delta x}, \mathbf{P}_{\Delta y}$ are computed from $\mathbf{P}_{0_d}, \mathbf{P}_{\Delta x_d}, \mathbf{P}_{\Delta y_d}$ through $\mathcal{D}(x_d, y_d)$. Note that the timestamp values of these points do not change. From these points, the normal vector of the *undistorted plane* is then computed through:

$$\mathbf{N} = (\mathbf{P}_{\Delta x} - \mathbf{P}_0) \times (\mathbf{P}_{\Delta y} - \mathbf{P}_0) \tag{10-7}$$

Using Eq. (7-4) with the components of $\mathbf{N}$ the undistorted flow vector is then obtained.

Chapter 11

Experimental Setup

This chapter provides additional details on the experimental setup discussed in the paper, which is used for data acquisition as well as flight tests. First, we expand on the general overview given in the paper in Section 11-1. Second, the communication between different parts of the setup is discussed in more detail in Section 11-3. Third, we address how timestamps of recorded measurements are synchronized in Section 11-4

11-1 Overview

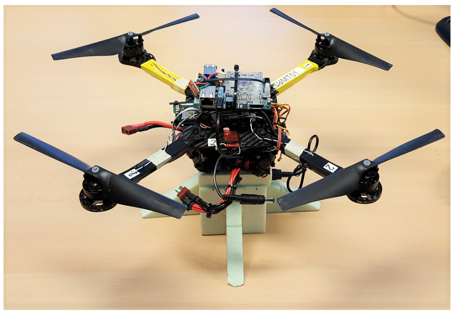
Figure 11-1 shows an overview of the experimental setup, repeated here for convenience. Pictures of the drone, referred to as the MavTec, are shown in Figure 11-1a and Figure 11-1b. The MavTec is a quadrotor with a lightweight carbon body spanning 0.39 m at its maximum length (excluding rotors). A DVS with a 4.5 mm focal length lens is mounted at the bottom. To protect the DVS during flight, it is shielded by a foam cover, which also acts as the landing gear.

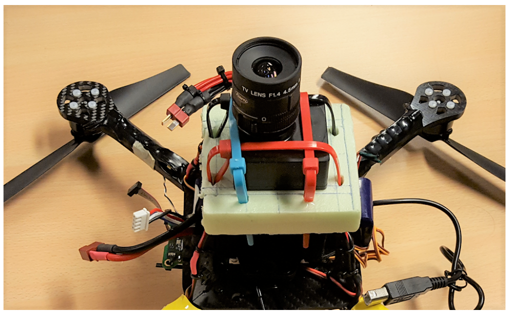
As discussed in the paper, there are two separate computers on-board. The first computer is an Odroid XU4 board, which is mounted on top of the quadrotor. It features a Samsung Exynos 5422 octacore CPU (four cores at 2.1 GHz and four at 1.5 GHz) and runs Ubuntu 15.04. It acts as a separate USB driver for the DVS. This enables using the readily available open-source software cAER, such that no specific USB drivers need to be developed.

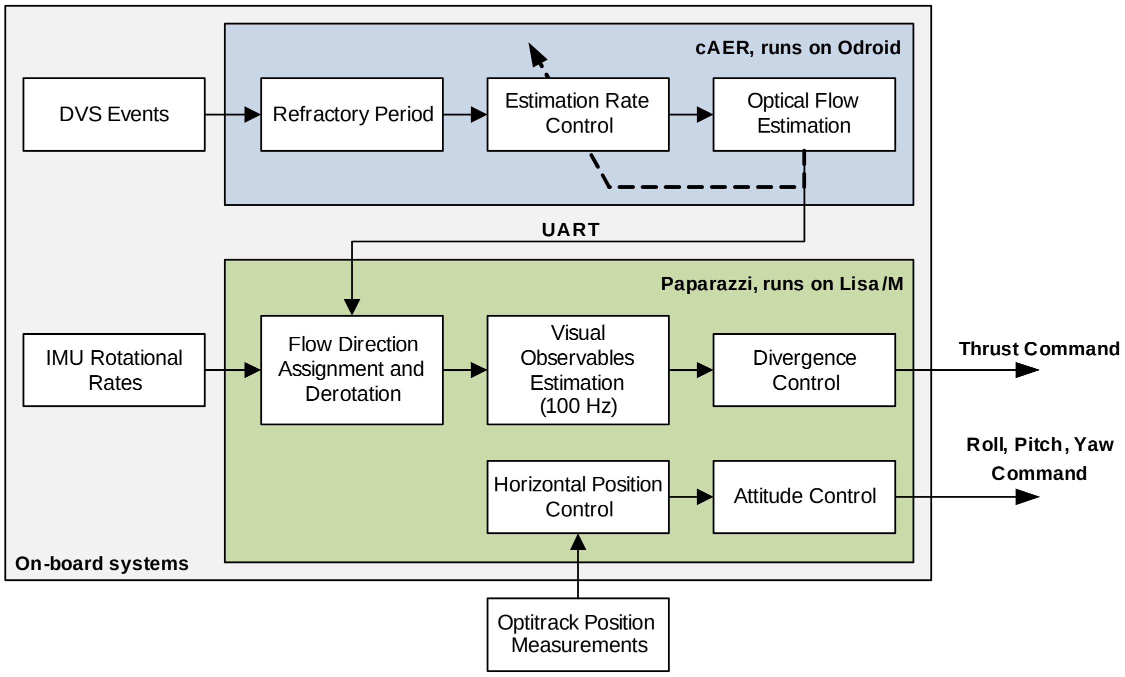
The second computer is the Lisa/M board, which features a 72MHz 32bit ARM microprocessor as well as a pressure sensor and IMU. It handles all main on-board processing necessary for flying using the open-source autopilot software Paparazzi[1]. This software has been extended during this work with a module that estimates visual observables from event-based optical flow measurements. This module also incorporates the divergence controller.

In order to transmit data between the Odroid and the Lisa/M, a serial UART connection is used. This datalink has a limited capacity; therefore, optical flow estimation is performed

[1]Paparazzi UAV, `http://wiki.paparazziuav.org/`


(a) Top view.


(b) Bottom view showing the DVS.



(c) Overview of the implementation

Figure 11-1: Overview of the experimental setup, including pictures of the MavTec. In (a) a top view of the vehicle is shown. The DVS is located at the bottom, protected by a foam cover. In (b) the cover is removed to expose the DVS. In (c) an overview of the processing workflow is shown, indicating the distribution of processes over the Odroid and the Lisa/M processors.

completely in cAER, such that only the events with optical flow need to be transmitted. Also, the Odroid is much more powerful. It appeared capable of processing over 250k events per second in real-time during optical flow estimation, even without control of ρ_F, though only when UART transmission is not incorporated during compilation.

Experiments are conducted in the CyberZoo facility of Delft University of Technology. During operation, an Optitrack pose tracking system measures the position and orientation of the MavTec in real-time. These measurements are used to compute ground truth values of optical flow and visual observables off-board. They are also used for horizontal position control of the MavTec to augment the divergence controller.

Optitrack uses an array of cameras that track objects formed by infrared markers. In a normal setup, test objects are fitted with passive reflecting markers, which reflect infrared strobing light emitted from the cameras. However, in (Censi et al., 2013) the strobing was found to interfere with DVS measurements, since the camera is most sensitive in the infrared spectrum. Therefore, during our experiments the strobing is disabled, and the MavTec is fitted with active infrared LED markers.

11-2 Operation during flight tests

A pilot controls the MavTec during flight using a remote controller. It can also regulate the mode of operation of the drone. Three modes are applied: (1) full manual control, (2) Optitrack-based stable hover at a specified height, and (3) constant divergence landing.

During normal flight, mode 1 (manual control) is used. The pilot is in full control, using the remote controller to send throttle, roll, pitch, and yaw commands. When a landing maneuver is initiated, the pilot first brings the MavTec to the approximate hover height. Once the MavTec is close enough, mode 2 (hover) is initiated to stabilize motion before landing. Then, mode 3 (constant divergence landing) is initiated, at which point the divergence controller takes over vertical control of the MavTec to perform a landing maneuver. The pilot monitors the MavTec's behavior during the maneuver, ready to switch back to mode 1 or 2 in case of unexpected motion. If self-induced oscillations occur close to the ground, the pilot switches back to mode 1, resuming manual control of the drone.

11-3 Communication architecture and data logging

Figure 11-2 shows how the different parts of the experimental setup transfer information to each other. In addition, it shows what measurements are logged during experiments, and where the logs are stored.

The on-board architecture works as follows. The DVS transmits events to the Odroid through a USB 2.0 cable. The Odroid then processes the events to obtain optical flow estimates. During this process, raw event measurements and optical flow are recorded on the Odroid. Events are stored in AER format, encoding position $\hat{x}, \hat{y}$, timestamp t, and polarity P. Optical flow vectors are logged in a CSV format, in which x, y, t, and P, as well as the optical flow (u, v) is recorded. In addition, real-time measurements of optical flow detection rate ρ_F and estimation delay are recorded. The latter is estimated by tracking and comparison of currently processed event timestamps and the Odroid real-time clock timestamp.

Optical flow estimates are then transmitted over a serial UART link to the Lisa/M. The UART link is configured to support a data rate of 921.6 kilobytes per second, enabling transmission of approximately 8500 optical flow vectors per second.

The Lisa/M processes optical flow to obtain visual observable estimates and perform divergence control. It also records its state measurements on an SD-card at 512 Hz. The most

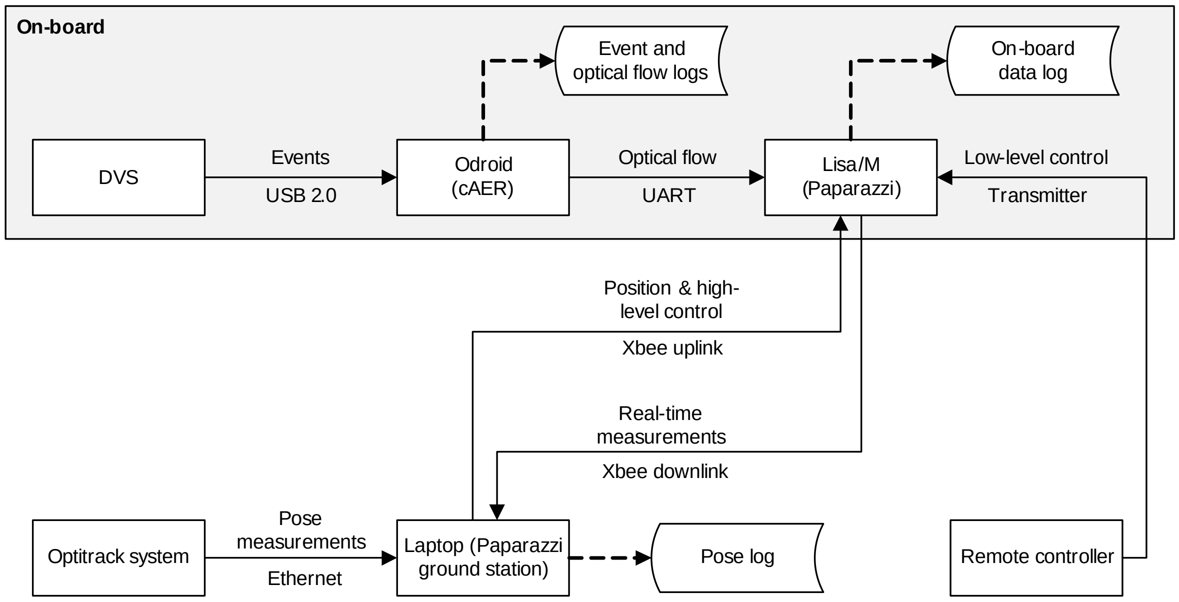


Figure 11-2: Flowchart of the communication between different parts of the experimental setup. Solid lines indicate the main workflow; dashed lines represent interaction with data storage.

important recorded variables are (1) the timestamp, (2) rotational rates (p, q, r), (3) visual observable estimates $(\vartheta_x, \vartheta_y, \vartheta_z)$, (4) optical flow reception rate ρ_F, and (5) the estimate confidence K.

During operation a ground station communicates with the MavTec, sending high-level commands, parameter adjustments, and position estimates from Optitrack, while receiving real-time measurements of relevant states. A laptop is used as a ground station, running the off-board segment of Paparazzi. It receives Optitrack measurements of the MavTec at 120 Hz through an ethernet cable. For wireless communication between the ground station and the Lisa/M, an XBee Pro antenna dongle is attached to the laptop and a compatible receiver is mounted on the Mavtec. Low-level commands are transmitted from a separate remote controller operated by the pilot.

11-4 Timestamp synchronization of log files

The use of different computers simultaneously means that the logs in Figure 11-2 are recorded using three separate clocks. The DVS timestamps, which are used to log measurements in cAER, are generated from an internal clock. On-board logs in Paparazzi are based on the real-time clock timestamp of the Lisa/M. In addition, the laptop records Optitrack measurements. Hence, synchronization of measurement timestamps is not a trivial task. In addition, several sources of latency occur in the pipeline, such as the USB and UART transmission. To synchronize the measurements for off-board processing, two different approaches are used.

First, in order to match the event timestamps to the Paparazzi measurements, the Paparazzi timestamp at which the first optical flow is received by the Lisa/M, is matched to the first event timestamp for which optical flow is estimated. However, this does not always provide a proper synchronization, since a time offset still exists in the respective signals of ρ_F measured by cAER and by Paparazzi. Therefore, follow-up synchronization is performed by computing the cross-correlation of both signals, and correcting for the offset.

Second, to match Paparazzi measurements to Optitrack pose logs, we utilize the active LEDs mounted on the MavTec. It is possible to remotely switch on the LEDs using the ground station, and at the same time, start logging on-board measurements. Since Optitrack only tracks the MavTec if the LEDs are switched on, this can be used as a synchronization point. Still, in several datasets this did not lead to properly synchronized measurements. This was observed in the rotational rates p, q, r, which are measured in Paparazzi and can be calculated from Optitrack pose measurements. In this case, cross-correlation of the rotational rates is used to synchronize the measurements.

Chapter 12

Software and Implementation

This chapter provides a more in-depth overview of the on-board implementation used during the experiments. First, the implementation of the optical flow estimation algorithm in cAER is presented in Section 12-1. Next, the Paparazzi architecture, the visual observable estimator, and the controller are detailed in Section 12-2. An overview of the source code and their locations is provided in Section 12-3, followed by an overview of the DVS settings and all parameters used in the algorithms (Section 12-4). Note that the text in this chapter extends on the content of the scientific paper, and hence assumes a basic understanding of the algorithm components and the experimental setup.

12-1 cAER module for optical flow estimation

This section aims to provide the reader some more insight in the optical flow estimation process performed on-board. This process runs on the Odroid as part of the open-source software cAER. This is a a framework for processing DVS events received through USB. It is written completely in C and can be run on Linux systems. The software is written in a modular fashion to facilitate extensions. For further information on cAER, please refer to the documentation in (Longinotti, 2014). Our optical flow algorithm is implemented in a newly developed module in cAER. It mainly performs optical flow estimation, but also handles logging and UART transmission.

12-1-1 Architecture

The global architecture of the cAER module is shown in Figure 12-1. Events arrive grouped in USB packets and are processed in the cAER mainloop thread, which sequentially performs all event-based processing for cAER modules. Our optical flow estimator is implemented as part of the main loop. The estimator also writes to logs for recording raw events and real-time timing information (primarily optical flow measurement rate and estimation delay).

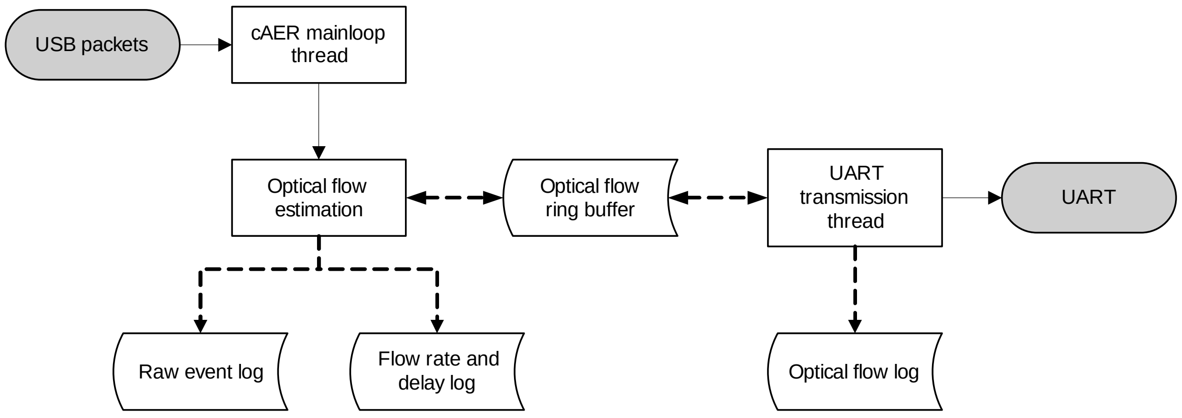


Figure 12-1: Flowchart of the cAER implementation architecture. Solid lines indicate the main workflow; dashed lines represent interaction with date storage.

Transmission across UART is handled in a separate thread to prevent delays in the mainloop due to transmission. Optical flow estimates are transferred from the mainloop thread to the UART handler thread through a ring buffer. This separate thread also logs the optical flow, such that the log exactly represents all optical flow that is transmitted.

12-1-2 Estimation algorithm

A flowchart of the optical flow processing is shown in Figure 12-2. In the following text, the steps taken in the algorithm are detailed.

In the algorithm the event timestamp history is maintained in a map M. which records the last event timestamp for each pixel location (x, y) and each event polarity P. M provides the timestamps of the most recent neighbor pixels, which together form the surface of most recent events Σ_e. It is also used to reject events that occur at a single location within quick succession, i.e within the pixel's refractory period Δt_R.

The workflow for each incoming event starts with two basic pre-filtering steps. When an event $\mathbf{e}_n = [x_n,\, y_n,\, t_n,\, P_n]^T$ arrives, the first filter applies the refractory period Δt_R. If $t_n - M\left(x, y, P\right) > \Delta t_E$, the event is preserved and we set $M(x_n, y_n, P_n) = t_n$. Otherwise, it is immediately rejected. The second filter evaluates if the flow throughput rate ρ_F does not exceed the specified limit $\rho_{F_{max}}$. This is done by computing the time difference $\Delta t_F = t_n - t_F$, rejecting the event if $\Delta t_F < 1/\rho_{F_{max}}$. Then, if the event has not been rejected, a list L is constructed of all events in the spatiotemporal neighborhood using the timestamps in M. This is done by iterating across all relative positions $(\delta x_i, \delta y_i)$ for which $\delta x_i \in \left[-\frac{1}{2}\Delta x, \frac{1}{2}\Delta x\right]$ and $\delta y_i \in \left[-\frac{1}{2}\Delta y, \frac{1}{2}\Delta y\right]$. Note that only events of equal polarity P are considered here. Timestamps for which $-\delta t_i < \Delta t$ (where $\delta t_i = M(x_i, y_i, P_n) - t_n$) are added to L along with $(\delta x_i, \delta y_i)$. An incremental sorting mechanism is applied, such that the resulting list is sorted by decreasing value of δt_i.

With the sorted list L established, first a check is performed for the number of events, proceeding only if sufficient events are available. If this is the case, a basic clustering approach is applied. We iterate through L, starting from the most recent neighbor event, and attempt to find two linearly independent pixel locations (based only on $(\delta x_i, \delta y_i)$). Upon finding the

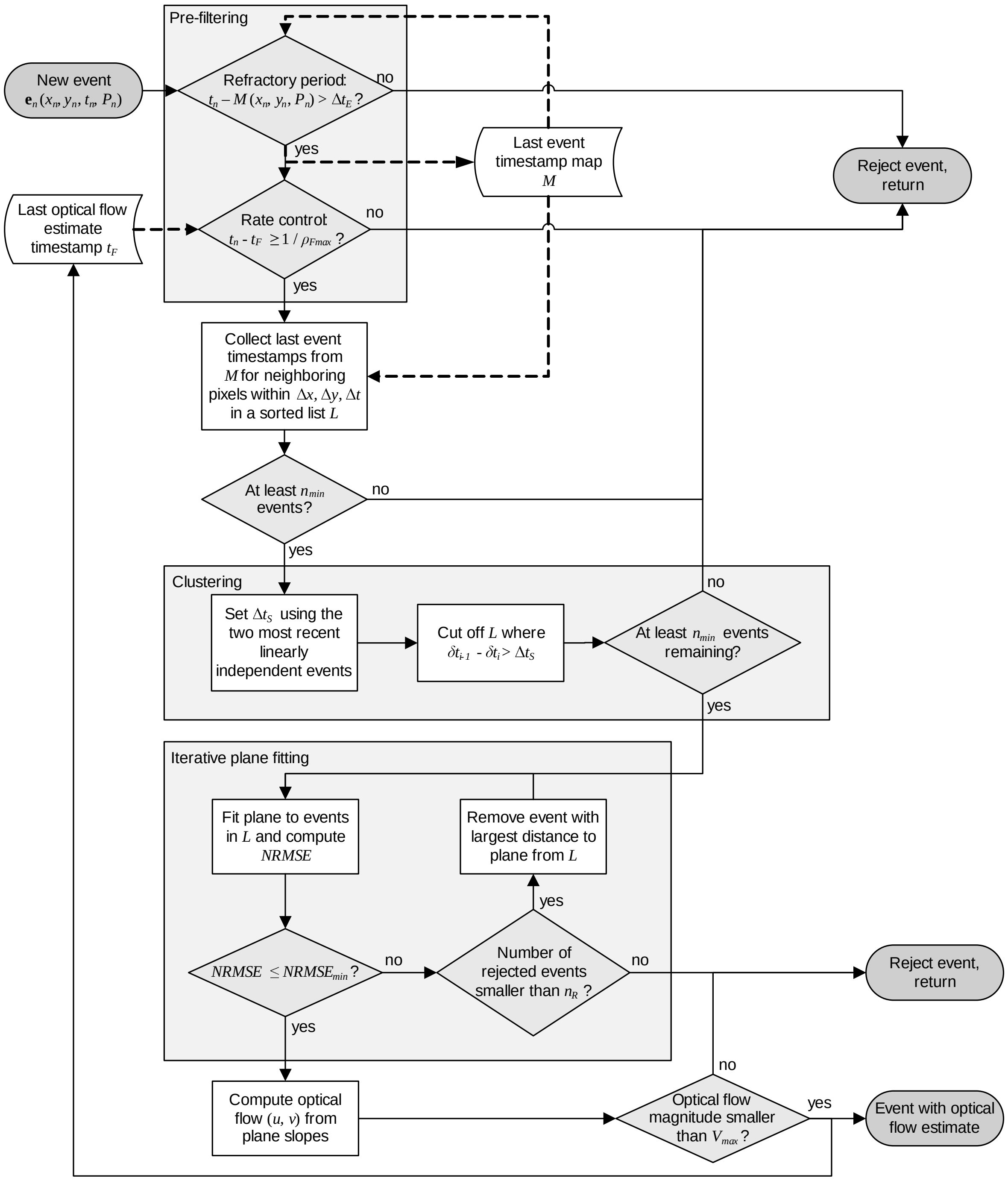

Pre-filtering
New event $\mathbf{e}_n(x_n, y_n, t_n, P_n)$
Refractory period: $t_n - M(x_n, y_n, P_n) > \Delta t_E$?
no
yes
Last event timestamp map M
Reject event, return
Last optical flow estimate timestamp t_F
Rate control: $t_n - t_F \geq 1/\rho_{Fmax}$?
no
yes
Collect last event timestamps from M for neighboring pixels within $\Delta x, \Delta y, \Delta t$ in a sorted list L
At least n_{min} events?
no
yes
Clustering
Set Δt_S using the two most recent linearly independent events
Cut off L where $\delta t_{i+1} - \delta t_i > \Delta t_S$
At least n_{min} events remaining?
no
yes
Iterative plane fitting
Fit plane to events in L and compute $NRMSE$
Remove event with largest distance to plane from L
yes
$NRMSE \leq NRMSE_{min}$?
no
Number of rejected events smaller than n_R ?
no
Reject event, return
yes
no
Compute optical flow (u, v) from plane slopes
Optical flow magnitude smaller than V_{max} ?
yes
Event with optical flow estimate

Figure 12-2: Flowchart of the optical flow algorithm. Solid lines indicate the main workflow; dashed lines represent interaction with date storage.

second pixel location, the maximum separation time $\Delta t_S = -k_S \delta t_i$ is set. Then, we iterate through the remaining events in the list and cut-off L if the time difference between two sequential events in L exceeds Δt_S. If the number of remaining events in L is still sufficient, the algorithm proceeds to the plane fitting procedure.

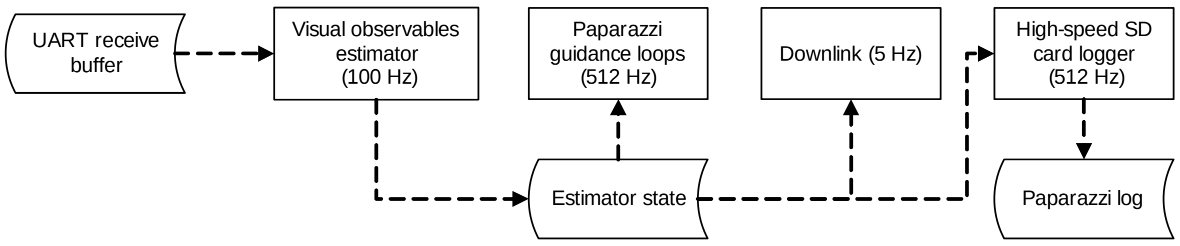

UART receive buffer
Visual observables estimator (100 Hz)
Paparazzi guidance loops (512 Hz)
Downlink (5 Hz)
High-speed SD card logger (512 Hz)
Estimator state
Paparazzi log

Figure 12-3: Overview of the Paparazzi architecture. Solid lines indicate the main workflow; dashed lines represent interaction with date storage.

Using the remaining events in L a least-squares fit of the reduced plane $\mathbf{\Pi}^* = \left[p_x^*, \, p_y^*\right]^T$ is computed, followed by the computation of the Normalized Root Mean Square Error (NRMSE). In this step we use the undistorted pixel locations (x_{u_i}, y_{u_i}) corresponding to (x, y), obtained using the undistortion mapping $\mathcal{D}$. If the NRMSE of the current fit is too large (i.e. above $NRMSE_{min}$), the point with the largest distance $dd_i = \delta x_i p_x^* + \delta y_i p_y^* + \delta t_i$ is removed from L. The plane is then recomputed, re-evaluating the NRMSE until either the fit is consistent enough, or more than n_R points are removed. In the latter case, the event is rejected.

Lastly, from the final plane slopes the optical flow vector (u, v) is computed and a final check of the optical flow magnitude is performed. If $\sqrt{u^2 + v^2} > V_{max}$ the estimate is considered too large and rejected. Otherwise, the algorithm produces a successful optical flow estimate.

12-2 Paparazzi module for visual observables estimation

The second part of the visual pipeline, estimation of the visual observables, is implemented in the open-source Paparazzi autopilot software[1]. This software runs on the Lisa/M board, providing autopilot functionality and facilitating control from ground using either a ground station or a remote controller. It also has a built-in modular structure. Our visual observables estimator is implemented in a standalone module, which handles optical flow reception from UART, visual observables estimation, confidence estimation, and divergence control.

A flowchart of the general architecture is shown in Figure 12-3. The main visual observables estimator runs at 100 Hz. It maintains its information in a shared state, which is used for downlinking information, logging, and control. The developed divergence controller is part of the Paparazzi guidance loops and runs at 512 Hz. Similarly, the high-speed logger module records the estimator state at 512 Hz. While this is much larger than the estimator frequency, this facilitates using higher estimator update frequencies if different hardware is used.

12-2-1 Estimation algorithm

Figure 12-4 shows a flowchart of the workflow in the visual observable estimator. The flowchart describes the periodic function of the estimator which runs at 100 Hz.

At the start, the program reads the UART receive buffer, decoding and obtaining the latest optical flow measurements. Each flow vector j that is successfully received, is assigned to

[1]Paparazzi UAV, `http://wiki.paparazziuav.org/`

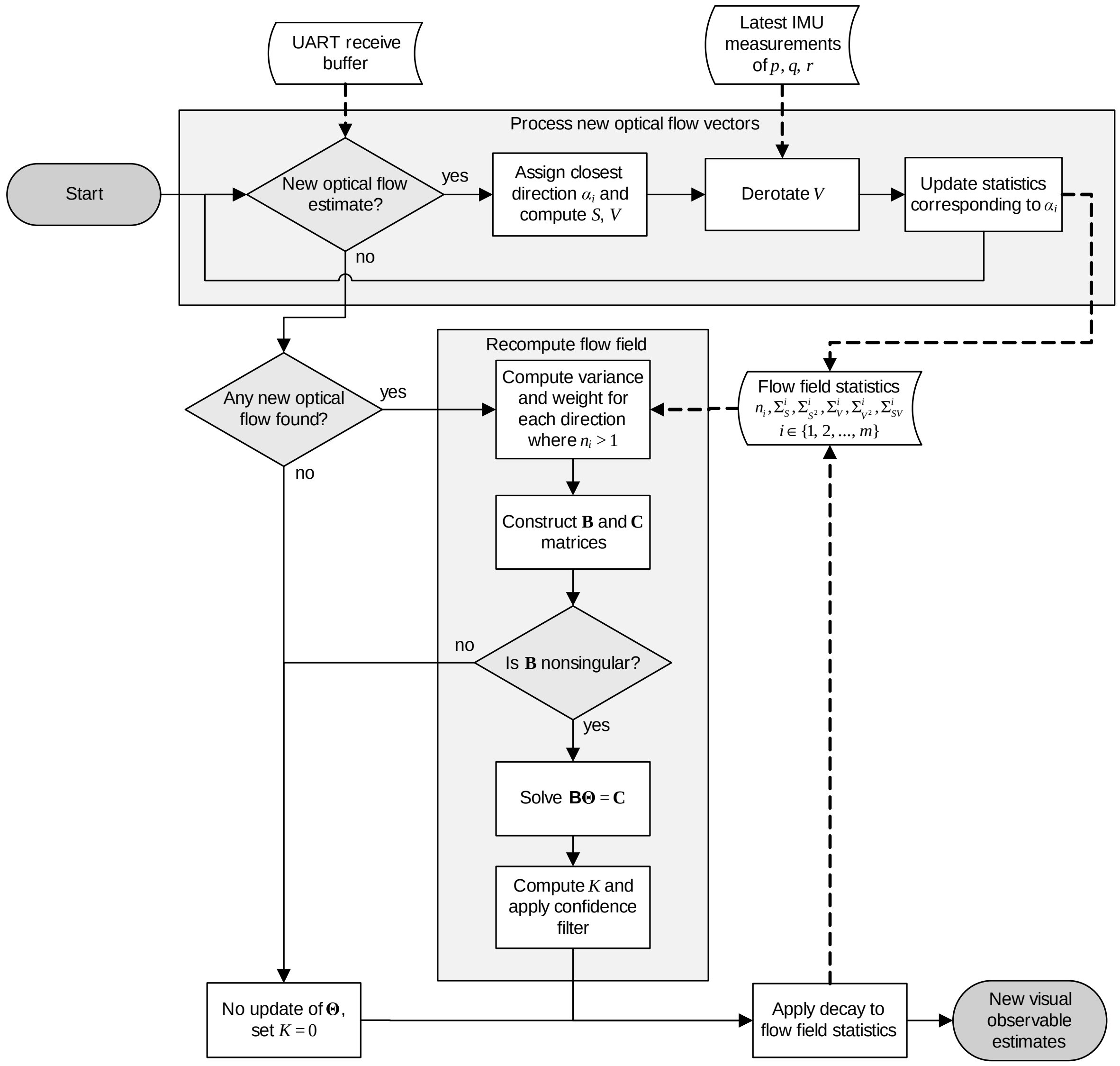

UART receive buffer
Latest IMU measurements of p, q, r
Process new optical flow vectors
Start
New optical flow estimate?
yes
no
Assign closest direction α_i and compute S, V
Derotate V
Update statistics corresponding to α_i
Recompute flow field
Any new optical flow found?
yes
no
Compute variance and weight for each direction where $n_i > 1$
Flow field statistics $n_i, \Sigma_S^i, \Sigma_{S^2}^i, \Sigma_V^i, \Sigma_{V^2}^i, \Sigma_{SV}^i$ $i \in \{1, 2, ..., m\}$
Construct $\mathbf{B}$ and $\mathbf{C}$ matrices
no
Is $\mathbf{B}$ nonsingular?
yes
Solve $\mathbf{B\Theta} = \mathbf{C}$
Compute K and apply confidence filter
No update of $\mathbf{\Theta}$, set $K = 0$
Apply decay to flow field statistics
New visual observable estimates

Figure 12-4: Flowchart of the of the visual observables estimator in Paparazzi. Solid lines indicate the main workflow; dashed lines represent interaction with date storage.

a direction group i with angle α_i. Along the assigned direction, the projected position S_i and velocity V_i are computed. Next, the value of V_i is corrected for rotational motion using the the latest measurements of p, q, r from the on-board IMU. Then, the flow field statistics corresponding to direction i are updated as follows:

$$\begin{aligned} n_i(j) =& \quad n_i(j-i)+1 \\ \Sigma^i_S(j) =& \quad \Sigma^i_S(j-1)+S_j \\ \Sigma^i_{S^2}(j) =& \quad \Sigma^i_{S^2}(j-1)+S_j^2 \\ \Sigma^i_V(j) =& \quad \Sigma^i_V(j-1)+V_j \\ \Sigma^i_{V^2}(j) =& \quad \Sigma^i_{V^2}(j-1)+V_j^2 \\ \Sigma^i_{SV}(j) =& \quad \Sigma^i_{SV}(j-1)+S_jV_j \end{aligned} \tag{12-1}$$

This process repeats with the remaining optical flow vectors available in the UART buffer, until no new vectors are available during this periodic iteration. Once the buffer is empty, the following step is using the latest flow field statistics to compose the weighted least-squares system $\mathbf{B\Theta} = \mathbf{C}$, where $\mathbf{B} = \mathbf{A}^T\mathbf{WA}$ and $\mathbf{C} = \mathbf{A}^T\mathbf{Wy}$. Note that this step is skipped if, during this periodic iteration, no optical flow estimates are available. In that case, the estimate confidence K is set to zero during this run, and no update of $\mathbf{\Theta}$ is performed.

First, the weight of each individual direction i is computed based on the variance $\text{Var}\{S\}_i$. From the flow field statistics we can compute this as follows:

$$\text{Var}\{S\}_i = \frac{\Sigma^i_{S^2}}{n_i} - \left(\frac{\Sigma^i_S}{n_i}\right)^2 \tag{12-2}$$

Then, the weight W_i of this direction is computed from the variance. In order to reduce the influence of directions where a limited number of optical flow vectors has been found, each direction where $n_i \leq 1$ is excluded. Note that, due to the preservation of statistics across periodic iterations, n_i is a floating point number instead of an integer.

Using the flow field statistics and weights, the matrices $\mathbf{B}$ and $\mathbf{C}$ are composed. If $\mathbf{B}$ is singular (i.e. $\det \mathbf{B} = 0$), no update is performed, and $K = 0$. Otherwise, the solution to $\mathbf{B\Theta} = \mathbf{C}$ is computed. With a solution available, we compute the coefficient of determination R^2, the individual confidence estimates $k_{\rho_F}, k_{\text{Var}\{S\}}, k_{R^2}$, and the total confidence value K.

R^2 is computed from the flow field statistics and the $\mathbf{C}$ matrix. With $R^2 = 1 - RSS/TSS$ as introduced in the paper, the Residual Sum of Squares (RSS) and Total Sum of Squares (TSS) are computed as follows:

$$\begin{aligned} RSS =& \quad \mathbf{y}^T\mathbf{Wy} - \mathbf{\Theta}^T\mathbf{A}^T\mathbf{Wy} \\ TSS =& \quad \mathbf{y}^T\mathbf{Wy} - \frac{\left(\sum\limits_{i=1}^{m} W\Sigma^i_V\right)^2}{\sum\limits_{i=1}^{m} Wn_i} \end{aligned} \tag{12-3}$$

It can be shown that the RSS and TSS can be computed directly from the flow field statistics:

$$
\begin{aligned}
RSS = & \sum_{i=1}^{m} W\Sigma^i_{V^2} - \mathbf{\Theta}^T\mathbf{C} \\
TSS = & \sum_{i=1}^{m} W\Sigma^i_{V^2} - \frac{\left(\sum_{i=1}^{m} W\Sigma^i_V\right)^2}{\sum_{i=1}^{m} Wn_i}
\end{aligned}
\tag{12-4}
$$

With the value of K established, the confidence filter is applied to $\mathbf{\Theta}$, which yields the final new estimate of the visual observables. The result is applied to update the divergence controller setpoint. As a last step, the values of the flow field statistics are multiplied by a decay factor $F = 1 - (t(k) - t(k-1))/(k_f)$, where k denotes the current iteration.

$$
\begin{aligned}
n_i(j) = & \quad Fn_i(j-1) \\
\Sigma^i_S(j) = & \quad F\Sigma^i_S(j-1) \\
\Sigma^i_{S^2}(j) = & \quad F\Sigma^i_{S^2}(j-1) \\
\Sigma^i_V(j) = & \quad F\Sigma^i_V(j-1) \\
\Sigma^i_{V^2}(j) = & \quad F\Sigma^i_{V^2}(j-1) \\
\Sigma^i_{SV}(j) = & \quad F\Sigma^i_{SV}(j-1)
\end{aligned}
\tag{12-5}
$$

The remaining values are preserved for the next iteration.

12-2-2 Divergence controller

The divergence controller uses the output of the visual observables estimator to regulate the throttle setting of the MavTec. A flowchart of the controller is shown in Figure 12-5. It enables using either visual estimates of divergence or ground truth divergence estimated from position estimates. The latter are computed from the Optitrack measurements.

As described in the paper, a simple proportional controller is used to control the throttle. With either ground truth or estimated divergence, the deviation from the control setpoint is computed, from which the deviation from nominal throttle ΔT is computed. The nominal throttle T_0 is estimated during hover before a landing is performed, using Paparazzi's standard vertical guidance controller[2]. The final throttle is then set as $T = T_0 + \Delta T$.

12-3 Overview of source code

An overview of the most important modifications and newly added files is given in 12-1. The table contains the files for both our cAER module and the Paparazzi module. The full source code for both modules is publicly available online at GitHub [3].

[2]For reference, see `http://wiki.paparazziuav.org/wiki/Control_Loops#Vertical_loop`

[3]cAER: `https://github.com/baspijhor/caer/tree/flow_adaptive_final`
Paparazzi: `https://github.com/baspijhor/paparazzi/tree/event_based_flow`

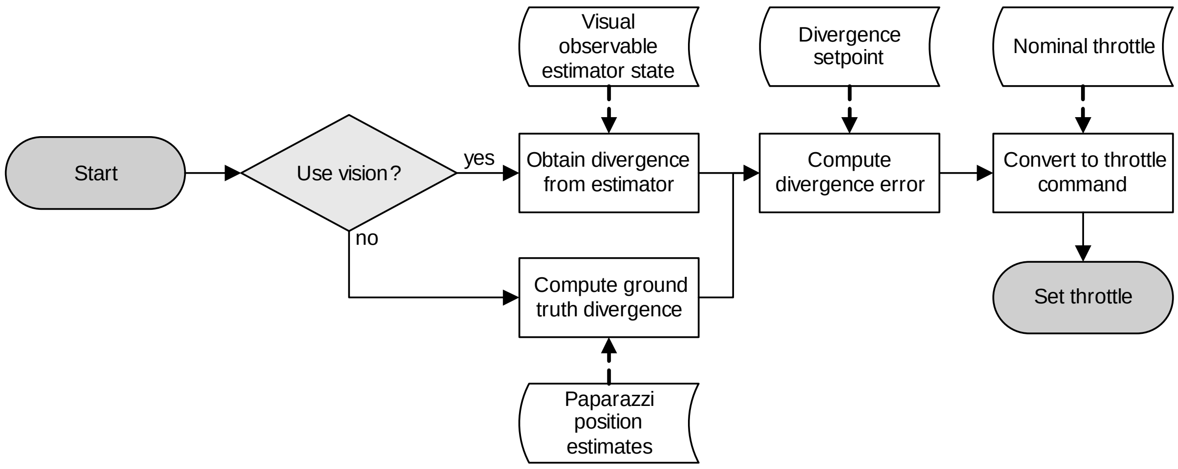

Visual observable estimator state
Divergence setpoint
Nominal throttle
Start
Use vision?
yes
no
Obtain divergence from estimator
Compute divergence error
Convert to throttle command
Compute ground truth divergence
Set throttle
Paparazzi position estimates

Figure 12-5: Flowchart of the divergence controller. Solid lines indicate the main workflow; dashed lines represent interaction with date storage.

In our implementation, cAER is compiled using `cmake`, as is discussed in the accompanied README file on GitHub. To obtain the configuration used in this work, one should run `cmake` with the following options:

```
cmake . -DDVS128=1 -DENABLE_OPTICFLOW=1
```

For Paparazzi, one can install the software using the standard procedure described at GitHub, using the provided repository[3] instead of the default Paparazzi repository.

12-4 Overview of sensor settings and algorithm parameter values

Table 12-2 provides settings of the 'biases' of the DVS used during experiments. The biases are programmable voltages or currents in the DVS pixel circuitry, which can be controlled by cAER[4]. In this work they are set for operation in the test environment (the CyberZoo), such that noise is limited, yet sufficient detail can be perceived.

An overview of all parameter settings of the visual pipeline is given in Table 12-3.

[4]A detailed explanation of each bias is provided at `http://inilabs.com/support/hardware/biasing/`

Table 12-1: Relevant new and modified source files in the open-source software packages cAER and Paparazzi, and a brief description of each file. The symbol # is used here to represent the path `sw/airborne/modules` in the Paparazzi software.

File name and path	Description
cAER	
`main.c`	[modified] Calls the 'run' function of each module for each newly available USB event packet. It is augmented with our module.
`modules/opticflow/dvs128Calibration.c`	Defines the DVS intrinsic parameters and lens distortion model and functions to incorporate them.
`modules/opticflow/flowAdaptive.c`	The main optical flow estimation code. The functionality is described in Section 12-1.
`modules/opticflow/flowEvent.h`	Header file defining the 'flow event': a data structure extending the standard DVS polarity event with optical flow components and undistorted pixel locations. Also contains auxiliary functions for initialization, getting, and clearing.
`modules/opticflow/flowOutput.c`	Handles the optical flow output: in a separate thread, the computed flow events are logged in a CSV file and/or transmitted through UART.
`modules/opticflow/flowRegularizationFilter.c`	Flow regularization filter described in the preliminary analysis in Section 7-4-3. Not included in the final pipeline.
`modules/opticflow/opticflow.c`	Core module file containing the 'run' function as well as functions for initialization and termination.
`modules/opticflow/uart.c`	Defines the functions for initializing UART communication and for data transmission and reception.
`modules/visualizer/visualizer.c`	[modified] Added a functionality for visualizing optical flow for off-board testing. However, at the time of writing, this is not working correctly.
Paparazzi	
`conf/airframes/KS/ks_mavtec1_dvs.xml`	Paparazzi airframe configuration file, defining the hardware, autopilot, and module settings for the MavTec.
`conf/modules/event_optic_flow.xml`	Paparazzi configuration file for the module.
`#/event_optic_flow/event_optic_flow.c`	Main module file containing the initialization, module start, periodic, and stop functionalities. It also contains the functionality for event acquisition from UART, and defines the constant divergence controller.
`#/event_optic_flow/flow_field_estimation.c`	Defines the functions used for updating the flow statistics, derotation, and flow field recomputing.
`#/loggers/high_speed_logger.c`	[modified] Module for logging measurements at the maximal attainable frequency in Paparazzi (512 Hz) to an SD card. The file is modified to log states in the added module.
`sw/ground_segment/misc/natnet2ivy.c`	[modified] Handles Optitrack measurements received on the ground station, which are transmitted to the MavTec. The modification separates logging from transmission to obtain a higher log sampling rate.

Table 12-2: DVS bias settings applied in cAER during the experiments. The values are provided as integers, similar to their definition in cAER.

Bias	Value
`cas`	54
`diff`	30153
`diffOff`	64
`diffOn`	850000
`foll`	61
`injGnd`	1108364
`pr`	217
`puX`	8159221
`puY`	16777215
`refr`	10
`req`	159147
`reqPd`	16777215

Table 12-3: Overview of all algorithm parameter values used in the experiments.

Parameter	Symbol	Value	Unit
Optical flow estimation			
Refractory period	Δt_R	0.1	[s]
Time window	Δt	2	[s]
Spatial window	Δx, Δy	5	[pixels]
Maximum number of rejected events	n_R	2	[-]
Maximal NRMSE	$NRMSE_{max}$	0.3	[-]
Timestamp difference factor for clustering	k_S	3	[-]
Minimum number of events in a fit	n_{min}	8	[-]
Maximum optical flow velocity	V_{max}	1000	[pixels/s]
Optical flow output rate setpoint	$\rho_{F_{max}}$	2500	[1/s]
Visual observable estimation			
Number of flow field directions	m	6	[-]
Flow field preservation time constant	k_f	0.02	[s]
Minimal variance for weight assignment	$\text{Var}\{S\}_{min}$	600	[pixels^2]
Confidence filter			
Minimal optical flow estimation rate	$\rho_{F_{min}}$	1500	[1/s]
Minimal coefficient of determination	R^2_{min}	1.0	[-]
Low-pass filter time constant	k_t	0.02	[s]
Update saturation limit for visual observables	$\Delta\vartheta_{max}$	0.3	[-]

Chapter 13

Supplementary Results

The results provided in this chapter support several observations in the paper. First, we show additional optical flow estimation results in Section 13-1, including a qualitative comparison of the baseline algorithm to the newly developed algorithm, and the quantitative effect of reducing the plane parameters on optical flow accuracy. Second, additional results for the visual observables estimator are presented in Section 13-2, which highlight the effects of flow field preservation and the confidence filter.

13-1 Optical flow estimation

13-1-1 Qualitative comparison between baseline and new algorithm

To augment the quantitative comparison between the baseline optical flow algorithm and the newly developed version performed in the paper (Section IV-D), this section provides side-by-side images of optical flow estimates in several scenes. Figure 13-1 shows four scenes of the event dataset in which significant performance differences are seen. In each scene, the optical flow vectors are scaled by a specified factor for better visualization (i.e. high-speed motion scenes are assigned a lower scaling factor than low-speed scenes). For generating these results, the algorithm parameters are equal to the values used in the paper.

In the first checkerboard scene ($\vartheta_z = 0.2$), the checkerboard contours move rather slowly. The main difference between the two algorithms is that, with the baseline algorithm, optical flow along low-speed lines is more difficult to estimate correctly. In comparison to the final algorithm, many optical flow vectors are missing, or motion is over-estimated at slow moving contours.

In the second scene (checkerboard, $\vartheta_z = 2.0$) the contours move much faster. At several fast-moving contours in the top right corner of the scene, the baseline algorithm fails to perceive the correct motion, converging to optical flow estimates that point in opposite direction. This phenomenon occurs most likely due to interaction with previously occurring features.

Our algorithm is able to also correctly estimate optical flow at these locations due to the pre-selection of interrelated events.

The third scene (roadmap, $\vartheta_z = 0.1$) contains very slow motion and sparse features. Note that for visualization of the events, the time window is rather large. The fixed time window of the baseline approach prevents observing most of the motion in this scene, even though some features are clearly visible. Using the final algorithm, which is capable of observing much slower motion, several optical flow vectors are still successfully found. Note that, in this scene, the vertical component of the ground truth optical flow is inconsistent with the estimates. It appears that in this scene, the vertical ventral flow is locally offset from the real value.

In the last roadmap scene ($\vartheta_z = 1.0$) motion is faster and features are often incomplete, as several events seem to be missing. Here, the baseline algorithm measures accurate normal flow at most locations. Our algorithm shows similar accuracy in this particular scene, but it tends to find a higher optical flow density, without losing significant quality.

13-1-2 Effect of plane parameter reduction on performance

In the paper we propose to reduce the computational complexity of the baseline algorithm, by reducing the number of parameters in the fitted plane. However, this has an effect on the achievable accuracy of the resulting optical flow. The results in this section quantify this effect, by repeating the comparison based on the Projection Endpoint Error (PEE) and density performed in Section IV-D of the paper. Instead of comparing the baseline to the final algorithm, this comparison involves two versions of the baseline algorithm in which the number of parameters is reduced to three and two respectively (according to Section IV-B in the paper).

Table 13-1 shows the resulting PEE and density values for the baseline algorithm and its three- and two-parameter variants. The PEE values show that the three-parameter and two-parameter algorithms obtain only a slightly lower accuracy than the baseline. In fast motion datasets, the errors are even lower for the three-parameter version compared to the baseline. However, a clear difference is seen in the values of the density η. Especially in slow-motion scenes and roadmap scenes, the three-parameter version obtains a significantly lower density. Slightly better results are seen for the two-parameter version.

13-2 Visual observables estimator

13-2-1 Effect of preservation of flow field statistics

The figures in this section aim to demonstrate the effect of preserving flow field statistics in the visual observables estimator, as is discussed in the paper in Section V-B. For this, we analyze estimates of ϑ_z for the two detail sequences in Figure 9(a) in the paper (vertical motion above a checkerboard texture). In the first sequence, slow motion towards the ground is performed, while in the second sequence, fast up-and-down motion is seen. Each sequence is analyzed without preservation ($k_f = 0$) and with preservation ($k_f = 0.5$) of statistics. In both cases, the confidence filter is disabled.

Table 13-1: Projection Endpoint Errors (mean absolute error and standard deviation) and density results of the baseline algorithm, three-parameter and two-parameter plane versions of the baseline algorithm, and the final algorithm. Values highlighted in bold are the lowest PEE or the highest density result for all algorithms.

	Baseline		Three parameters		Two parameters	
	PEE [pix/s]	η [%]	PEE [pix/s]	η [%]	PEE [pix/s]	η [%]
Checkerboard, $\vartheta_y = 1.0$	**18.6** ± 22.9	**50.6**	19 ± 24.5	46.9	20.7 ± 27.5	49.5
Checkerboard, $r = -1.3$	26.8 ± 28.1	**50.7**	**26.3** ± 26.1	47.1	27.6 ± 28.3	49.9
Checkerboard, $\vartheta_z = 0.2$	**7.78** ± 12.1	**12.6**	8.73 ± 19	2.51	12.1 ± 20.8	5.56
Checkerboard, $\vartheta_z = 0.5$	**13.0** ± 17.9	**29.5**	14.4 ± 23.1	18.3	15.4 ± 23.8	24.5
Checkerboard, $\vartheta_z = 2.0$	42.4 ± 58.4	**57.5**	**34.3** ± 31.2	54.2	37.1 ± 33.5	57.2
Roadmap, $\vartheta_z = 0.1$	**7.85** ± 6.91	**3.54**	12 ± 12.7	0.36	15.8 ± 24.3	1.28
Roadmap, $\vartheta_z = 0.5$	**13.5** ± 13.7	**8.44**	15.8 ± 21.3	3.16	16.7 ± 21.9	6.45
Roadmap, $\vartheta_z = 1.0$	26.4 ± 30.3	**13.3**	**24.6** ± 28.2	7.7	25.2 ± 31	12.4

Figure 13-2 and Figure 13-3 show height measurements and estimates of ϑ_z for the first and second sequences respectively. At low speed, flow field preservation makes a clear difference. Without preservation, sharp noise peaks are present due to momentarily inaccurate optical flow estimates. With preservation, many of these sharp peaks are completely rejected.

13-2-2 Effectiveness of the confidence filter

In this section the effectiveness of the confidence filter (Section V-C in the paper) is evaluated. This is done using three sequences with vertical motion, by evaluation of the estimate accuracy of ϑ_z and the confidence value K. The sequences are again part of the checkerboard-based dataset used in the paper in Figure 9(a). In the first sequence (shown in Figure 13-4), the camera is at first standing on the ground, before being moved upwards. In the second sequence (Figure 13-5), slow motion is performed, while in the third sequence (Figure 13-6) the speed is increased.

For each sequence, four results are shown. The first result is generated without any confidence filter active, i.e. where $K = 1$ if optical flow is available, or $K = 0$ otherwise. In the second, third, and fourth results, the confidence metric incorporates an increasing number of individual confidence indicators, starting with $K = k_{\rho_F}$, followed by $K = k_{\rho_F} k_{\text{Var}\{S\}}$, and finally $K = k_{\rho_F} k_{\text{Var}\{S\}} k_{R^2}$. In all results, flow field preservation is not applied (i.e. $k_f = 0$)).

At low speed, filtering based on ρ_F rejects many noise peaks due to temporarily low optical flow availability. Filtering by variance has a smaller effect, but still reduces some local peaks at low speed. In particular, filtering using k_{R^2} has a strong effect, since it significantly reduces the confidence value. Note that at high speed, R^2 appears to be generally larger due to more consistent optical flow. However, around zero-crossings, the fit becomes clearly much less consistent, which results in update delay around the crossing. Note that this effect is mainly contained in R^2, and not so much in ρ_F or $\text{Var}\{S\}$.

Checkerboard, $\vartheta_z = 0.2$
$\Delta t_e = 0.25$ s, $\Delta t_F = 0.05$ s
Flow scale = 0.4

Checkerboard, $\vartheta_z = 2.0$
$\Delta t_e = 0.025$ s, $\Delta t_F = 0.005$
Flow scale = 0.1

Roadmap, $\vartheta_z = 0.1$
$\Delta t_e = 0.5$ s, $\Delta t_F = 0.1$ s
Flow scale = 0.4

Roadmap, $\vartheta_z = 1.0$
$\Delta t_e = 0.12$ s, $\Delta t_F = 0.02$ s
Flow scale = 0.2

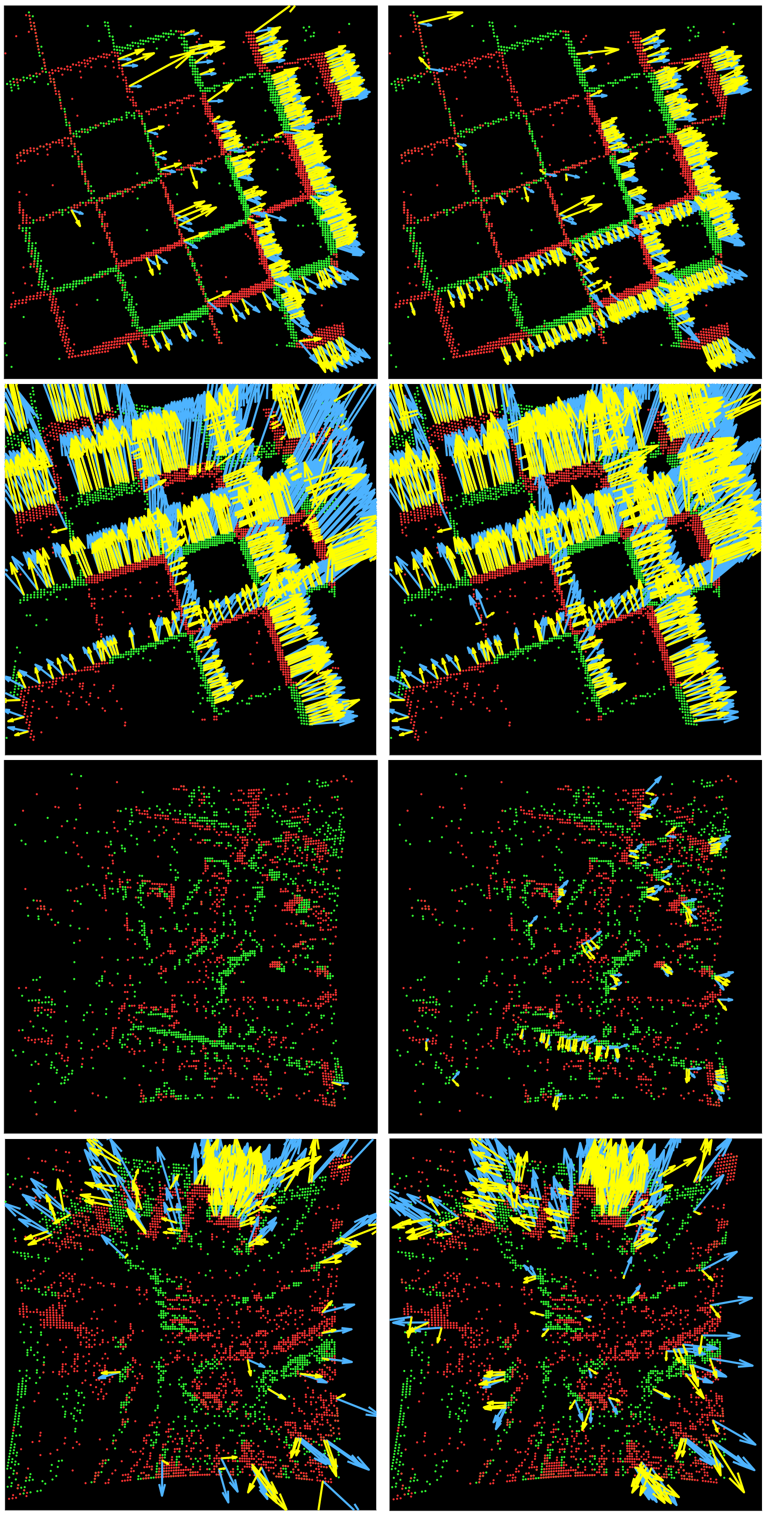


Figure 13-1: Optical flow estimates for four specific sequences. Left: sequence name, time windows for showing events (Δt_e) and optical flow (Δt_F), and the applied scaling factor for the optical flow vector magnitude. Middle: optical flow estimated from the baseline algorithm in the paper (Benosman et al., 2014). Right: estimates from the presented optical flow algorithm. Yellow arrows show the estimated optical flow, while the accompanying blue arrows show the ground truth vectors. Events are shown as green dots (positive polarity) or as red dots (negative polarity).

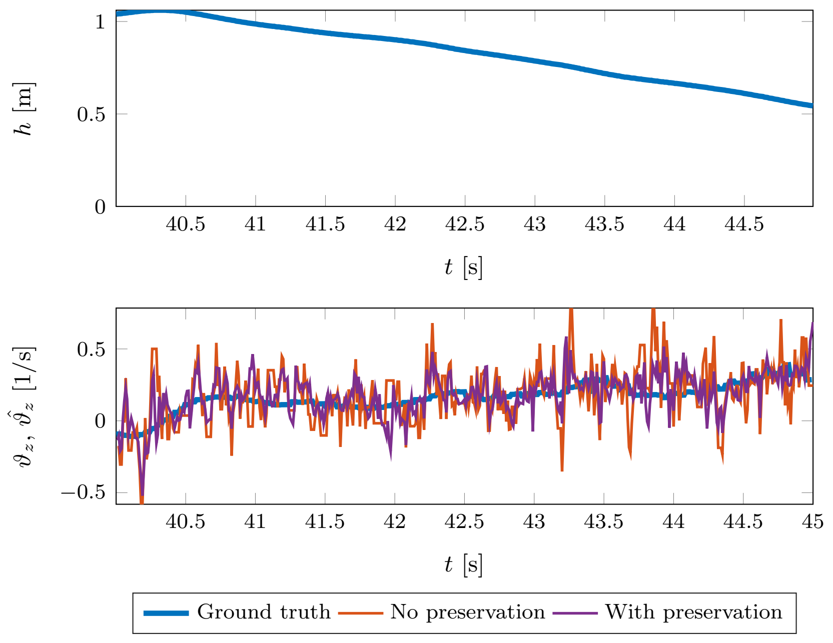


Figure 13-2: Height measurements and ground truth values and estimates of ϑ_z for the first sequence, in which slow downward motion is performed.

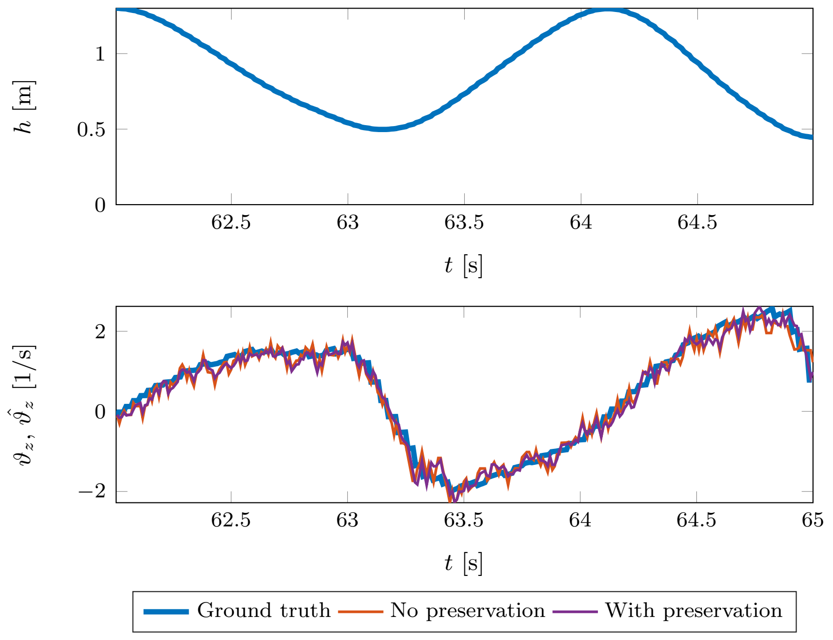


Figure 13-3: Height measurements and ground truth values and estimates of ϑ_z for the first sequence, in which fast up-and-down motion is performed.

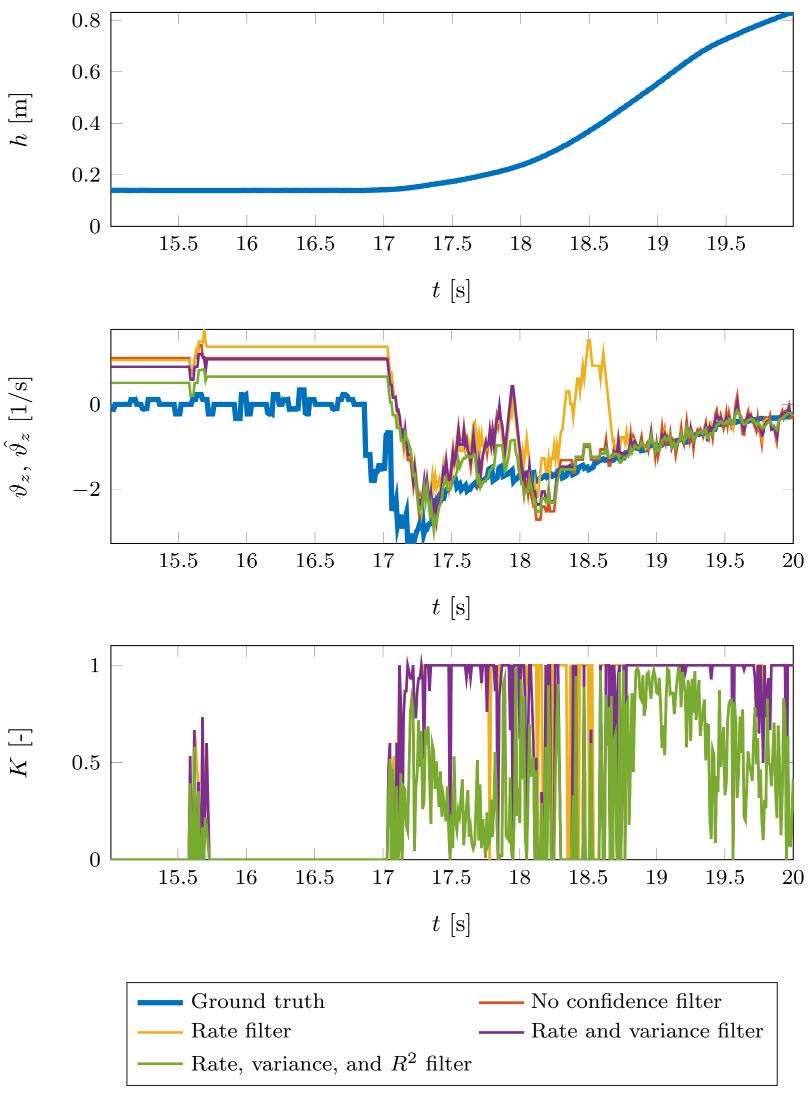


Figure 13-4: From top to bottom: height measurements, ground truth values and estimates of ϑ_z, and confidence values for the first sequence, in which the DVS is standing on the ground before being moved upwards.

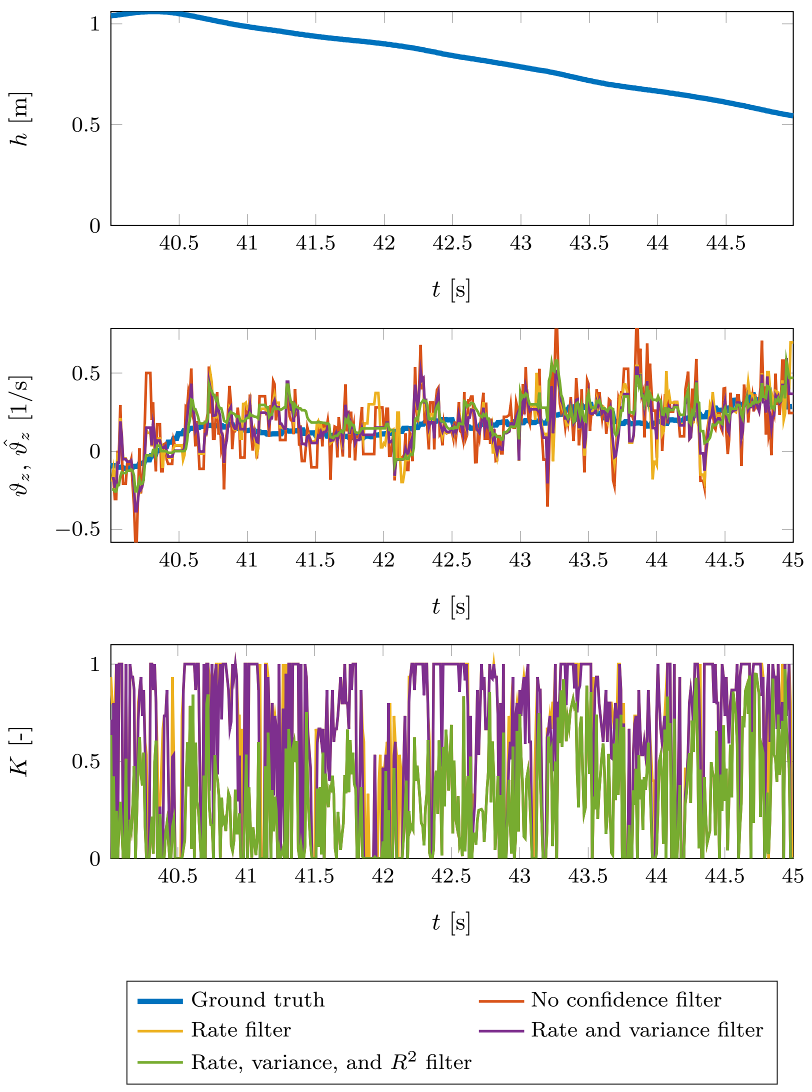


Figure 13-5: From top to bottom: height measurements, ground truth values and estimates of ϑ_z, and confidence values for the second sequence, in which slow downward motion is performed.

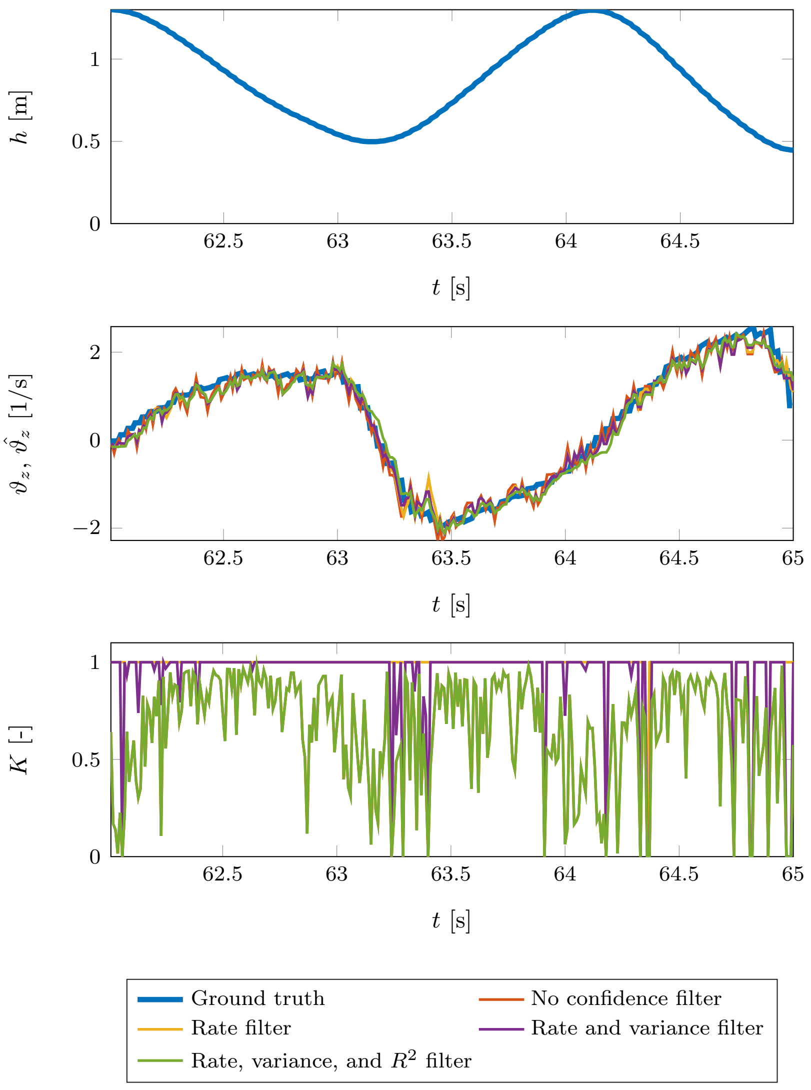


Figure 13-6: From top to bottom: height measurements, ground truth values and estimates of ϑ_z, and confidence values for the third sequence, in which fast up-and-down motion is performed.

Bibliography

Adiv, G. (1985). Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE transactions on pattern analysis and machine intelligence*, *7*(4), 384–401. doi: 10.1109/TPAMI.1985.4767678

Alkowatly, M. T., Becerra, V. M., & Holderbaum, W. (2015). Bioinspired Autonomous Visual Vertical Control of a Quadrotor Unmanned Aerial Vehicle. *Journal of Guidance, Control, and Dynamics*, *38*(2), 249–262. doi: 10.2514/1.G000634

Baird, E., Boeddeker, N., Ibbotson, M. R., & Srinivasan, M. V. (2013). A universal strategy for visually guided landing. *Proceedings of the National Academy of Sciences of the United States of America*, *110*(46), 18686–18691. doi: 10.1073/pnas.1314311110

Baker, S., Scharstein, D., Lewis, J. P., Roth, S., Black, M. J., & Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, *92*(1), 1–31. doi: 10.1007/s11263-010-0390-2

Bardow, P., Davison, A. J., & Leutenegger, S. (2016). Simultaneous Optical Flow and Intensity Estimation from an Event Camera. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)* (pp. 884–892). doi: 10.1109/CVPR.2016.102

Barranco, F., Fermuller, C., Aloimonos, Y., & Delbruck, T. (2016). A Dataset for Visual Navigation with Neuromorphic Methods. *Frontiers in Neuroscience*, *10*(February), 1–9. doi: 10.3389/fnins.2016.00049

Barron, J. L., Fleet, D. J., & Beauchemin, S. S. (1994). Performance of optical flow techniques. *International Journal of Computer Vision*, *12*(1), 43–77. doi: 10.1007/BF01420984

Beauchemin, S. S., & Barron, J. L. (1995). The computation of optical flow. *ACM Computing Surveys*, *27*(3), 433–466. doi: 10.1145/212094.212141

Benosman, R., Clercq, C., Lagorce, X., Ieng, S.-H., & Bartolozzi, C. (2014). Event-based visual flow. *IEEE transactions on neural networks and learning systems*, *25*(2), 407–17. doi: 10.1109/TNNLS.2013.2273537

Benosman, R., Ieng, S.-H., Clercq, C., Bartolozzi, C., & Srinivasan, M. (2012). Asynchronous frameless event-based optical flow. *Neural Networks*, *27*, 32–37. doi: 10.1016/j.neunet.2011.11.001

Bouguet, J.-Y. (1999). *Complete Camera Calibration Toolbox for Matlab.* Retrieved 2016-04-01, from `http://www.vision.caltech.edu/bouguetj/calib_doc/`

Brandli, C., Berner, R., Yang, M., Liu, S.-C., & Delbruck, T. (2014). A 240x180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE Journal of Solid-State Circuits*, *49*(10), 2333–2341. doi: 10.1109/JSSC.2014.2342715

Brosch, T., Tschechne, S., & Neumann, H. (2015). On event-based optical flow detection. *Frontiers in Neuroscience*, *9*(137), 1–15. doi: 10.3389/fnins.2015.00137

Brown, D. (1966). Decentering Distortion of Lenses. *Photometric Engineering*, *32*(3), 444–462.

Camus, C. (2010). *Extraction of optical flow fields from a silicon retina optical sensor* (MSc thesis, Technische Universität Darmstadt). Retrieved from `http://www.honda-ri.de/intern/pub/id/mastersthesisreference201010080849933877`

Censi, A., & Scaramuzza, D. (2014). Low-Latency Event-Based Visual Odometry. In *2014 IEEE International Conference on Robotics & Automation (ICRA)* (pp. 703–710). doi: 10.1109/ICRA.2014.6906931

Censi, A., Strubel, J., Brandli, C., Delbruck, T., & Scaramuzza, D. (2013). Low-latency localization by active LED markers tracking using a dynamic vision sensor. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 891–898). doi: 10.1109/IROS.2013.6696456

Chahl, J. S., Srinivasan, M. V., & Zhang, S. W. (2004). Landing Strategies in Honeybees and Applications to Uninhabited Airborne Vehicles. *The International Journal of Robotics Research*, *23*(2), 101–110. doi: 10.1177/0278364904041320

Chao, H., Gu, Y., & Napolitano, M. (2014). A Survey of Optical Flow Techniques for Robotics Navigation Applications. *Journal of Intelligent & Robotic Systems*, *73*(1-4), 361–372. doi: 10.1007/s10846-013-9923-6

Cho, D.-i. D., & Lee, T.-j. (2015). A Review of Bioinspired Vision Sensors and Their Applications. *Sensors and Materials*, *27*(6), 447–463. doi: 10.18494/SAM.2015.1083

Clady, X., Clercq, C., Ieng, S.-H., Houseini, F., Randazzo, M., Natale, L., ... Benosman, R. (2014). Asynchronous visual event-based time-to-contact. *Frontiers in neuroscience*, *8*(9). doi: 10.3389/fnins.2014.00009

Clady, X., Ieng, S.-H., & Benosman, R. (2015). Asynchronous event-based corner detection and matching. *Neural Networks*, *66*, 91–106. doi: 10.1016/j.neunet.2015.02.013

Conradt, J. (2015). On-Board Real-Time Optic-Flow for Miniature Event-Based Vision Sensors. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO).* Zhuhai, China. doi: 10.1109/ROBIO.2015.7419043

Conradt, J., Berner, R., Cook, M., & Delbruck, T. (2009). An embedded AER dynamic vision sensor for low-latency pole balancing. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops* (pp. 780–785). doi: 10.1109/ICCVW.2009.5457625

Conradt, J., Cook, M., Berner, R., Lichtsteiner, P., Douglas, R. J., & Delbruck, T. (2009). A pencil balancing robot using a pair of AER dynamic vision sensors. In *2009 IEEE International Symposium on Circuits and Systems* (pp. 781–784). doi: 10.1109/ISCAS.2009.5117867

De Croon, G. C. H. E. (2016). Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspiration & Biomimetics*, *11*(1), 1–18. doi: 10.1088/1748-3190/11/1/016004

De Croon, G. C. H. E., Alazard, D., & Izzo, D. (2015). Controlling spacecraft landings with constantly and exponentially decreasing time-to-contact. *IEEE Transactions on Aerospace and Electronic Systems*, *51*(2), 1241–1252.

De Croon, G. C. H. E., De Clercq, K. M. E., Ruijsink, R., Remes, B., & De Wagter, C. (2009). Design, aerodynamics, and vision-based control of the DelFly. *International Journal of Micro Air Vehicles*, *1*(2), 71–97. doi: 10.1260/175682909789498288

De Croon, G. C. H. E., Ho, H. W., De Wagter, C., Van Kampen, E., Remes, B., & Chu, Q. P. (2013). Optic-flow based slope estimation for autonomous landing. *International Journal of Micro Air Vehicles*, *5*(4), 287–297. doi: 10.1260/1756-8293.5.4.287

De Valois, R. L., Cottaris, N. P., Mahon, L. E., Elfar, S. D., & Wilson, J. A. (2000). Spatial and temporal receptive fields of geniculate and cortical cells and directional selectivity. *Vision research*, *40*(27), 3685–3702. doi: 10.1016/S0042-6989(00)00210-8

De Wagter, C., Tijmons, S., Remes, B. D. W., & De Croon, G. C. H. E. (2014). Autonomous flight of a 20-gram Flapping Wing MAV with a 4-gram onboard stereo vision system. In *Proceedings - 2014 IEEE International Conference on Robotics and Automation* (pp. 4982–4987). doi: 10.1109/ICRA.2014.6907589

Delbruck, T. (2007). *jAER Open Source Project.* Retrieved 2015-11-23, from `http://sourceforge.net/p/jaer/wiki/Home/`

Delbruck, T., & Lang, M. (2013). Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Frontiers in Neuroscience*, *7*(223), 1–7. doi: 10.3389/fnins.2013.00223

Delbruck, T., & Lichtsteiner, P. (2007). Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. In *2007 IEEE International Symposium on Circuits and Systems* (pp. 845–848). doi: 10.1109/ISCAS.2007.378038

Delbruck, T., Pfeiffer, M., Juston, R., Orchard, G., Muggler, E., Linares-Barranco, A., & Tilden, M. (2015). Human vs. computer slot car racing using an event and frame-based DAVIS vision sensor. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 2409 – 2412). doi: 10.1109/ISCAS.2015.7169170

Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, *58*, 1–35. doi: 10.1093/jxb/erm298

Drazen, D., Lichtsteiner, P., Häfliger, P., Delbrück, T., & Jensen, A. (2011). Toward real-time particle tracking using an event-based dynamic vision sensor. *Experiments in Fluids*, *51*(5), 1465–1469. doi: 10.1007/s00348-011-1207-y

Eichner, H., Joesch, M., Schnell, B., Reiff, D. F., & Borst, A. (2011). Internal Structure of the Fly Elementary Motion Detector. *Neuron*, *70*(6), 1155–1164. doi: 10.1016/j.neuron.2011.03.028

Evangelista, C., Kraft, P., Dacke, M., Reinhard, J., & Srinivasan, M. V. (2010). The moment before touchdown: landing manoeuvres of the honeybee Apis mellifera. *The Journal of experimental biology*, *213*(2), 262–270. doi: 10.1242/jeb.037465

Expert, F., & Ruffier, F. (2012). Controlling docking, altitude and speed in a circular high-roofed tunnel thanks to the optic flow. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Vol. 7, pp. 1125–1132). doi: 10.1109/IROS.2012.6385946

Expert, F., & Ruffier, F. (2015). Flying over uneven moving terrain based on optic-flow cues without any need for reference frames or accelerometers. *Bioinspiration & biomimetics*, *10*. doi: 10.1088/1748-3182/10/2/026003

Expert, F., Viollet, S., & Ruffier, F. (2011). A mouse sensor and a 2-pixel motion sensor exposed to continuous illuminance changes. *SENSORS, 2011 IEEE*(2), 974–977. doi: 10.1109/ICSENS.2011.6127002

Floreano, D., Pericet-Camara, R., Viollet, S., Ruffier, F., Brückner, A., Leitel, R., . . . Franceschini, N. (2013). Miniature curved artificial compound eyes. In *Proceedings of the National Academy of Sciences of the United States of America* (Vol. 110, pp. 9267–72). doi: 10.1073/pnas.1219068110

Floreano, D., & Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *Nature*, *521*(7553), 460–466. doi: 10.1038/nature14542

Franceschini, N., Riehle, A., & Le Nestour, A. (1989). Directionally selective motion detection by insect neurons. In *Facets of Vision* (pp. 360–390). Springer-Verlag Berlin.

Geyer, C., & Daniilidis, K. (2000). A unifying theory for central panoramic systems and practical implications. In David Vernon (Ed.), *Computer Vision-ECCV 2000* (pp. 445–461). Springer Berlin Heidelberg. doi: 10.1007/3-540-45053-X_29

Gibson, J. J. (1979). *The ecological approach to visual perception.* Boston: Houghton Mifflin.

Gil, P., García, G. J., Mateo, C. M., & Torres, F. (2014). Active visual features based on events to guide robot manipulators in tracking tasks. In *Preprints of the 19th World Congress of the International Federation of Automatic Control* (pp. 11890–11897).

Grabe, V., Bulthoff, H. H., Scaramuzza, D., & Giordano, P. R. (2015). Nonlinear ego-motion estimation from optical flow for online control of a quadrotor UAV. *International Journal of Robotics Research*, *34*, 1114–1135. doi: 10.1177/0278364915578646

Green, W., Oh, P., & Barrows, G. (2004). Flying insect inspired vision for autonomous aerial robot maneuvers in near-earth environments. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation* (pp. 2347–2352). doi: 10.1109/ROBOT.2004.1307412

Hassenstein, B., & Reichardt, W. (1956). Systemtheoretische analyse der zeit-, reihenfolgen- und vorzeichenauswertung bei der bewegungsperzeption des rüsselkäfers chlorophanus. *Zeitschrift für Naturforschung B*, *11*(9-10), 513–524.

Herissé, B., Hamel, T., Mahony, R., & Russotto, F.-X. (2012). Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow. *IEEE Transactions on Robotics*, *28*(1), 77–89. doi: 10.1109/TRO.2011.2163435

Ho, H. W., & De Croon, G. C. H. E. (2016). Characterization of Flow Field Divergence for MAVs Vertical Control Landing. In *AIAA Guidance, Navigation, and Control Conference* (pp. 1–13). doi: 10.2514/6.2016-0106

Ho, H. W., de Croon, G. C. H. E., van Kampen, E., Chu, Q. P., & Mulder, M. (2016). *Adaptive Control Strategy for Constant Optical Flow Divergence Landing.* Retrieved from `http://arxiv.org/abs/1609.06767`

Huber, B. (2014). *High-Speed Pose Estimation using a Dynamic Vision Sensor* (MSc Thesis, University of Zurich). Retrieved from `http://www.kutter-fonds.ethz.ch/App_Themes/default/datalinks/BasilHuber_UniZ_MT2014.pdf`

IniLabs. (n.d.). *DVS Specifications.* Retrieved 2015-12-17, from `http://inilabs.com/products/dynamic-vision-sensors/specifications/`

Izzo, D., & De Croon, G. C. H. E. (2012). Landing with Time-to-Contact and Ventral Optic Flow Estimates. *Journal of Guidance, Control, and Dynamics*, *35*(4), 1362–1367. doi: 10.2514/1.56598

Kendoul, F. (2014). 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. doi: 10.1177/0278364913509496

Kendoul, F., Fantoni, I., & Nonami, K. (2009). Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, *57*(6-7), 591–602. doi: 10.1016/j.robot.2009.02.001

Klette, R. (2014). *Concise Computer Vision: An Introduction into Theory and Algorithms* (I. Mackie, Ed.). Springer-Verlag London. doi: 10.1007/978-1-4471-6320-6

Kramer, J. (2002). An integrated optical transient sensor. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *49*(9), 612–628. doi: 10.1109/TCSII.2002.807270

Lee, D. N. (1976). A theory of visual control of braking based on information about time-to-collision. *Perception*, *5*(4), 437–459. doi: 10.1068/p050437

Lee, D. N., Davies, M. N. O., Green, P. R., & Van der Weel, F. R. (1993). Visual Control of Velocity of Approach by Pigeons When Landing. *Journal of Experimental Biology*, *180*(1), 85–104.

Li, C., Brandli, C., Berner, R., Liu, H., Yang, M., Liu, S.-c., & Delbrück, T. (2015). Design of an RGBW color VGA rolling and global shutter dynamic and active-pixel vision sensor. In *IEEE International Symposium on Circuits and Systems* (pp. 718–721). doi: 10.1109/ISCAS.2015.7168734

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, *43*(2), 566–576.

Litzenberger, M., Posch, C., Bauer, D., Belbachir, A., Schon, P., Kohn, B., & Garn, H. (2006). Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor. In *2006 IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop* (pp. 173–178). doi: 10.1109/DSPWS.2006.265448

Longinotti, L. (2014). *cAER: A framework for event-based processing on embedded systems* (BSc Thesis, University of Zürich). Retrieved from `http://sourceforge.net/projects/jaer/files/cAER/`

Longuet-Higgins, H. C., & Prazdny, K. (1980). The interpretation of a moving retinal image. *Proceedings of the Royal Society of London, B: Biological Sciences*, *208*(1173), 385–397. doi: 10.1098/rspb.1980.0057

Lucas, B. D., & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence* (Vol. 81, pp. 674–679).

McCarthy, C., & Barnes, N. (2004). Performance of optical flow techniques for indoor navigation with a mobile robot. In *IEEE International Conference on Robotics and Automation* (pp. 5093–5098). doi: 10.1109/ROBOT.2004.1302525

McCarthy, C., Barnes, N., & Mahony, R. (2008). A Robust Docking Strategy for a Mobile Robot Using Flow Field Divergence. *IEEE Transactions on Robotics*, *24*(4), 832–842. doi: 10.1109/TRO.2008.926871

Mueggler, E., Huber, B., & Scaramuzza, D. (2014a). *Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers.* Retrieved 2016-04-26, from `https://www.youtube.com/watch?v=LauQ6LWTkxM`

Mueggler, E., Huber, B., & Scaramuzza, D. (2014b). Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2761–2768). doi: 10.1109/IROS.2014.6942940

Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Vol. 1, pp. I—-652). doi: 10.1109/CVPR.2004.1315094

Paz Gomes Verdugo, M. (2015). *Bio-inspired Optical Flow applied to MAV Landing* (MSc thesis). Delft University of Technology.

Piatkowska, E., Belbachir, A. N., Schraml, S., & Gelautz, M. (2012). Spatiotemporal multiple persons tracking using Dynamic Vision Sensor. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (pp. 35–40). doi: 10.1109/CVPRW.2012.6238892

Posch, C., Matolin, D., & Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, *46*(1), 259–275. doi: 10.1109/JSSC.2010.2085952

Posch, C., Serrano-Gotarredona, T., Linares-barranco, B., & Delbrück, T. (2014). Retinomorphic Event-Based Vision Sensors : Bioinspired Cameras With Spiking Output. *Proceedings of the IEEE*, *102*(10), 1470–1484.

Richter, C., Rohrbein, F., & Conradt, J. (2014). *Bio- inspired optic flow detection using neuromorphic hardware.* Retrieved 2015-11-12, from `http://mediatum.ub.tum.de/doc/1226041/600345.pdf`

Rosten, E., Porter, R., & Drummond, T. (2008). Faster and better: a machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(1), 105–19. Retrieved from `http://arxiv.org/pdf/0810.2434` doi: 10.1109/TPAMI.2008.275

Ruckauer, B., & Delbruck, T. (2016). Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in Neuroscience*, *10*(176). doi: 10.3389/fnins.2016.00176

Ruffier, F., & Franceschini, N. (2005). Optic flow regulation: The key to aircraft automatic guidance. *Robotics and Autonomous Systems*, *50*(4), 177–194. doi: 10.1016/j.robot.2004.09.016

Ruffier, F., & Franceschini, N. (2014). Optic Flow Regulation in Unsteady Environments: A Tethered MAV Achieves Terrain Following and Targeted Landing Over a Moving Platform. *Journal of Intelligent & Robotic Systems*, *79*, 275–293. doi: 10.1007/s10846-014-0062-5

Serrano-Gotarredona, T., Leñero-Bardallo, J. A., & Linares-Barranco, B. (2011). A Bioinspired 128x128 Pixel Dynamic-Vision-Sensor. In *26th Conference on Design of Circuits and Integrated Systems.*

Serrano-Gotarredona, T., & Linares-Barranco, B. (2013). A 128, 128 1.5% contrast sensitivity 0.9% FPN 3 μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE Journal of Solid-State Circuits*, *48*(3), 827–838. doi: 10.1109/JSSC.2012.2230553

Srinivasan, M., Zhang, S., Lehrer, M., & Collett, T. (1996). Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, *199*(1), 237–244. doi: 10.1006/anbe.1998.0897

Srinivasan, M. V. (1994). An image-interpolation technique for the computation of optic flow and egomotion. *Biological Cybernetics*, *71*(5), 401–415. doi: 10.1007/BF00198917

Tschechne, S., Sailer, R., & Neumann, H. (2014). Bio-Inspired Optic Flow from Event-Based Neuromorphic Sensor Input. *Artificial Neural Networks in Pattern Recognition*, *8774*, 171–182.

Van Breugel, F., Morgansen, K., & Dickinson, M. H. (2014). Monocular distance estimation from optic flow during active landing maneuvers. *Bioinspiration & Biomimetics*, *9*. doi:

10.1088/1748-3182/9/2/025002

Yang, M., Liu, S.-C., & Delbruck, T. (2015). A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding. *IEEE Journal of Solid-State Circuits*, *50*(9), 2149–2160. doi: 10.1109/JSSC.2015.2425886

Zaghloul, K. a., & Boahen, K. (2004). Optic Nerve Signals in a Neuromorphic Chip II: Testing and Results. *IEEE Transactions on Biomedical Engineering*, *51*(4), 667–675. doi: 10.1109/TBME.2003.821040

Zufferey, J.-C., Beyeler, A., & Floreano, D. (2010). Autonomous flight at low altitude using light sensors and little computational power. *International Journal of Micro Air Vehicles*, *2*(2), 107–117. doi: 10.1260/1756-8293.2.2.107

Zufferey, J.-C., & Floreano, D. (2006). Fly-Inspired Visual Steering of an Ultralight Indoor Aircraft. *IEEE Transactions on Robotics*, *22*(1), 137–146.