

A Filtering Approach for Computation of Real-Time Dense Optical-flow for Robotic Applications

Juan David Adarve¹, David J. Austin², Robert Mahony^{1,*}

¹The Australian National University, ²MadJ Innovations

{juan.adarve, robert.mahony}@anu.edu.au, david@madjinnovations.com

Abstract

The present article presents an iterative filter approach for the computation of optical-flow. The filter is based on an update and propagation loop. The propagation stage takes the currently computed flow to predict the value at the next time iteration. The update stage takes this prediction, together with the stream of images, and corrects the optical-flow field. This leads to an incremental approach to build optical-flow. Regions of the image where no flow computation is possible are filled in by a diffusion term in the propagation step. Ground truth validation of the algorithm is provided by simulating a high-speed camera in a 3D scene. The local computations and convolution based implementation is well suited for real-time systems with high-speed and high-acuity cameras.

1 Introduction

The latest sensor technology and embedded computing systems enables the development of a new class of integrated vision sensing systems that we term *Rapid Embedded Vision* (REV) Systems: that is, systems with high frame rate (typically ten to a hundred times faster than the natural dynamics of the environment in which it is embedded), sufficient acuity to distinguish all objects of interest, and integrated inertial measurement systems and data processing capabilities. An example of such a system is the *FastVis* system developed by MadJ Innovations [FastVis, 2014] with a frame rate of 340 images per second and resolution of 2 Megapixels, that has the potential to provide REV system functionality for a wide

range of highly dynamic mobile robotic applications. A characteristic of such systems, however, is that the full-rate, full-resolution image data rate is so large, of the order of Gigabits per second, that it is impossible to consider transmitting the data off camera for external processing. Special algorithms are required that exploit the particular properties of the image sequence expected and embedded processing capabilities that are integrated into a REV system.

Dense optical-flow is a powerful cue for motion control of robotic vehicles. Optical-flow has been used for obstacle avoidance [Coombs *et al.*, 1998] and docking manoeuvres [Santos-Victor and Sandini, 1997; McCarthy and Barnes, 2004; McCarthy *et al.*, 2008]. It has been heavily studied by biologists investigating control strategies of animals and insects [Ruffier and Franceschini, 2005a; Srinivasan and Zhang, 2004; Chahl *et al.*, 2004; Srinivasan *et al.*, 2006]. Recently proposed methods for computing dense flow fields yield impressive results in terms of accuracy against ground-truth sequences such as the Middlebury dataset [Baker *et al.*, 2011] and the MPI Sintel dataset [Butler *et al.*, 2012]. The computational complexity of state-of-the-art algorithms, however, make them impossible to implement in real-time for any REV system, even with the most powerful integrated field-programmable gate array (FPGA) or graphics processing unit (GPU) computing facilities. Existing real-time optical-flow sensing systems [Honegger *et al.*, 2013; Plett *et al.*, 2012; Diaz *et al.*, 2006] for mobile robotic applications use either simple correlation based algorithms [Lucas and Kanade, 1981; Srinivasan, 1994], custom hardware such as mouse chips [Beyeler *et al.*, 2009], simple optical analogue sensors [Ruffier and Franceschini, 2005b], or neuromorphic chips [Benosman *et al.*, 2014]. These algorithms and hardware systems, to the authors knowledge, still use a classical optical-flow computational paradigm where only a few frames at a time are used to make a complete estimate of the optical-flow, and this information is not propagated to future estimates. For a REV system, hundreds of images are available every second, each frame very similar to the previ-

* Robert Mahony is with the ARC Centre of Excellence for Robotic Vision at the Australian National University. <http://www.roboticvision.org/>

^{1,2} This research was supported by the Australian Research Council and MadJInnovation through the Linkage grant LP110200768 “A high-speed light-weight embedded vision system for robotics and computer vision applications”.

ous one, and classical processing paradigm experiences significant limitations since there is insufficient variation in just a few sequential images to properly constrain the optical-flow estimate. Filtering type approaches for the computation of optical-flow [Black and Anandan, 1991; Heeger, 1988; Fleet and Langley, 1995] that exploit the stream of image data appear better suited for implementation on REV systems.

The present paper proposes a filter type iterative algorithm for the computation of optical-flow. An update-propagation loop architecture similar to the work by Black [Black, 1992] is employed. In the propagation step the present best estimate of flow field is propagated forward to a guess of the flow at the next time instant. This process is modelled as a transport PDE system [Thomas, 1995] where the optical flow field gets transported by itself. This allows us to define the propagation as a non-linear transform of the flow field, rather than using warping and re-sampling strategies. Following the propagation, the new image information is used to make an incremental update the flow estimate. This update is based on the constant brightness constraint [Barron *et al.*, 1994] plus a prior regularisation term. An averaging process introduces a diffusion into the propagation stage of the filter that allows the flow estimates to diffuse into regions of the image with poor or no texture. The proposed approach can easily be implemented as a series of convolutions and basic arithmetic operations suitable for implementation on an FPGA.

The paper is organised as follows: Section 2 develops the theoretical insight of the proposed filter. Section 3 handles the implementation details of the filter. Experimental validation of the algorithm against ground-truth data is provided in Section 4. The paper is closed with a discussion on the computational complexity of the algorithm in Section 5 and conclusions in Section 6.

2 Problem Formulation

The goal of this paper is to develop an algorithm for the efficient computation of an optical-flow field Φ for a sequence of images. We consider the optical-flow computed at a given pixel at a given time as a sampled version of a spatio-temporal flow field $\Phi_{\text{pix/s}}(t, \zeta)$ in continuous time coordinate t and continuous spatial coordinate ζ . Let τ be a scaled time variable $\tau = \sigma t$ for σ the frame rate of the vision system. That is, $\tau = k$ corresponds to the time t at which frame k was sampled. Writing the flow in the new scaled time variable gives

$$\Phi(\tau, \zeta) = \Phi_{\text{pix/s}}(\tau/\sigma, \zeta) = \Phi_{\text{pix/s}}(t, \zeta)$$

where $\Phi(\tau, \zeta)$ is measured in pixels per frame. To obtain a computable algorithm we must use a finite dimensional representation of the optical flow field $\Phi(\tau, \zeta)$. We choose a grid of control points $\{\xi^r\}$ across the image indexed

by $r = 1, \dots, N_r$. The flow estimate at a control point ξ^r at time k will be denoted

$$\Phi_k^r := \Phi(k, \xi^r).$$

For dense optical-flow it is desirable to have a dense grid of points $\{\xi^r\}$ with sufficient resolution to represent the natural structure of the environment. However, since image texture is generally much higher resolution than environment structure, it is rarely necessary to have the same density of control points as pixels. We note in addition that it is not required that control points are located at pixel centres, although this is often simplest choice.

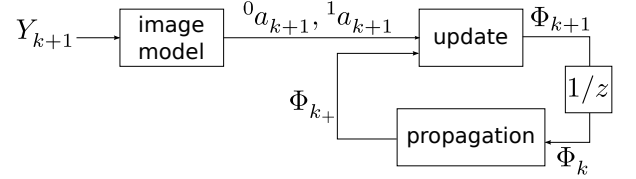


Figure 1: Proposed filter architecture. During the image acquisition process, the old flow vector field Φ_k is propagated to an estimate Φ_{k+} of the expected flow at time $k + 1$. New image data at time $k + 1$ is used to identify image parameters $({}^0a_{k+1}, {}^1a_{k+1})$, that are in turn used to update the propagated Φ_{k+} to obtain a new flow estimate Φ_{k+1} at time $k + 1$.

The data considered is a sequence of images Y_k from a Rapid Embedded Vision (REV) sensor system as described in the introduction. Analogous to the flow field, we will assume that the measured image is sampled from a spatio-temporal continuous “image” $Y(\tau, \zeta)$ where we use the same time scaling as introduced above. The measured image is given by a spatio-temporal average

$$Y_k(\xi) \approx \int_{k-\Delta_k}^{k+\Delta_k} \int_{\Lambda_\xi} Y(\tau, \zeta) d\zeta d\tau$$

over sample period $\tau \in (k - \Delta_k, k + \Delta_k)$ and field-of-view Λ_ξ of pixel ξ of the full continuous spatio-temporal image $Y(\tau, \zeta)$. The sampling process is physically implemented in the CMOS array of the image sensor.

The proposed approach is based on a filter architecture that aims to incrementally build an estimate of the optical-flow field by exploiting the spatio-temporal nature of the data stream. The algorithm architecture (Figure 1) consists of a propagation stage that takes the current estimated flow and estimates the new flow one time step in the future, and a measurement update in which new image data is used to correct the current optical-flow estimate. A preprocessing stage extracts model parameters from the image, which in turn are used as input data for the filter loop. Moreover, we employ a local processing paradigm. That is around every control

point we define a support region Ω^r , such that only the image and state data in Ω^r will contribute to the flow estimate computed at ξ^r . For example, the propagation step takes the flow field Φ_k^r from the local domain Ω^r around ξ^r and computes the new estimate of the propagated flow Φ_{k+}^r at the control point ξ^r . Similarly, the incoming image information Y_k in the domain Ω^r is used to produce image parameters ${}^0a_{k+1}^r$, and ${}^1a_{k+1}^r$ associated with estimates of the mean image intensity and mean image gradient at the control point ξ^r , and used in turn to update the flow estimate Φ_{k+}^r . In practice, the support domains $\{\Omega^r\}$ must overlap in order for the proposed algorithm to propagate information beyond the local neighbourhood of a single control point.

2.1 State propagation

The propagation stage takes the current estimate of optical-flow Φ_k^r in the domain Ω^r and uses this information to “propagate” the flow and predict the expected flow value Φ_{k+}^r at time $k+1$. We use the partial notation k_+ to indicate an estimate of the flow at time $k+1$ based only on past information. In the update step, described in §2.3, the new image information Y_{k+1} will update Φ_{k+}^r to an estimate Φ_{k+1}^r that is based on all available past and present information.

Consider the underlying smooth spatio-temporal vector field $\Phi(\tau, \zeta)$ in continuous variables τ and ζ . The vector field $\Phi(\tau, \zeta)$ defines a local flow on a domain Ω^r around time $\tau = k$

$$\frac{d}{d\tau}\zeta_k(\tau; \zeta) = \Phi(\tau, \zeta_k(\tau; \zeta)), \quad \zeta_k(k; \zeta) = \zeta \in \Omega^r. \quad (1)$$

Assuming that $\Phi(\tau, \zeta)$ is locally continuous then classical ODE theory ensures that there exists a $\delta_k > 0$ such that the solution $\zeta_k(\tau; \zeta)$ to (1) exists and is unique on $\tau \in (k - \delta_k, k + \delta_k)$.

The transport process is based on the assumption that the variable being propagated does not change independently with time [Thomas, 1995]. That is time change of the variable at a point in space is purely due to transport of the variable by the flow $\zeta_k(\tau; \zeta)$. Applying this assumption to the vector field $\Phi(\tau, \zeta)$ transporting itself we get the *transport identity*

$$\Phi(\tau, \zeta_k(\tau; \zeta)) = \Phi(k, \zeta). \quad (2)$$

for $\tau \in (k - \delta_k, k + \delta_k)$. That is, the propagated flow is (locally) just a parallel transport of the flow field defined at time $\tau = k$ along trajectories $\zeta_k(\tau; \zeta)$. In particular when $\zeta = \xi^r$ this provides an integral condition on the value of $\Phi(\tau, \zeta_k(\tau; \xi^r))$ along a flow curve passing through ξ^r at time $\tau = k$. Since $d/d\tau\Phi(k, \xi^r) \equiv 0$, the right-hand side of (2) does not depend on τ , then taking

the total time differential and evaluating at $\tau = k$ yields

$$0 = \frac{d}{d\tau}\Phi(\tau, \zeta_k(\tau; \xi^r))|_{\tau=k} \quad (3)$$

$$= \frac{\partial}{\partial \zeta}\Phi(k, \zeta)\Big|_{\zeta=\xi^r} \cdot \frac{d}{d\tau}\zeta_k(\tau; \xi^r)\Big|_{\tau=k} + \frac{\partial}{\partial \tau}\Phi(\tau, \xi^r)\Big|_{\tau=k}.$$

Note that from (1)

$$\frac{d}{d\tau}\zeta_k(\tau; \xi^r)\Big|_{\tau=0} = \Phi_k^r. \quad (4)$$

We will introduce a notation for the Jacobian of the spatio-temporal flow field

$$J_k^r := \frac{\partial}{\partial \zeta}\Phi(k, \zeta)\Big|_{\zeta=\xi^r}. \quad (5)$$

That is the Jacobian $J_k^r \in \mathbb{R}^{2 \times 2}$ is a spatial partial differential of the flow field $\Phi(\tau, \zeta)$, at time $\tau = k$, and evaluated at the control point $\zeta = \xi^r$. Then rearranging (3) yields

$$\frac{\partial}{\partial \tau}\Phi(\tau, \xi^r)\Big|_{\tau=k} = -J_k^r \Phi_k^r. \quad (6)$$

Note that this relationship only holds at $\tau = k$ and $\xi = \xi^r$, however, by discretizing this relationship we will obtain a numerical formulae for the time evolution of Φ^r at control points $\{\xi^r\}$.

We approximate the partial time derivative of the flow in (6) by a first order forward Euler estimate

$$\frac{\partial}{\partial \tau}\Phi(\tau, \xi^r)\Big|_{\tau=k} \approx \Phi_{k+}^r - \Phi_k^r, \quad (7)$$

where Φ_{k+}^r is the forward estimate of the flow and we recall that unit time step in τ is a sample period. Substituting (7) into (6) and solving for Φ_{k+}^r one obtains

$$\Phi_{k+}^r = (I_2 - J_k^r)\Phi_k^r. \quad (8)$$

where I_2 is the 2×2 identity matrix. The previous flow estimate Φ_k^r is an internal state of the algorithm and the Jacobian J_k^r must be computed numerically from Φ_k^r in the neighbourhood Ω^r as we will discuss in Section 3.1. The choice of a forward Euler step for the time-derivative is the simplest and most naive choice of numerical implementation and more sophisticated numerical integration schemes could be considered depending on computational resources available.

2.2 Image Model

Consider a neighbourhood Ω^r of a reference pixel $\xi^r = (x^r, y^r)$. Within this neighbourhood, we approximate the image intensity by an affine function

$$\hat{Y}_k^r(\zeta) = {}^0a_k^r + {}^1a_k^r(\zeta - \xi^r), \quad \zeta \in \Omega^r \quad (9)$$

with ${}^0a_k^r \in \mathbb{R}$ an estimate of the average image intensity of the image over Ω^r , and ${}^1a_k^r = ({}^xa_k^r, {}^ya_k^r) \in \mathbb{R}^{1 \times 2}$ an estimate of the image intensity gradient. Note that the image model is expressed in the continuous variable ζ although it can be easily be computed at pixel locations $\zeta = \xi \in \Omega^r$ when required.

The image parameters $({}^0a_k^r, {}^1a_k^r)$ are characterised as the solution to a least squares cost optimisation problem. It is numerically advisable to introduce a weighting function $w(\xi; \xi^r)$ to the least squares cost in order to allow more flexibility in smoothing discontinuities in the image. A robust estimate of image model parameters can be achieved by solving the weighted cost function

$$\begin{aligned} \varepsilon_1({}^0a_k^r, {}^1a_k^r) &= \sum_{\xi \in \Omega^r} w(\xi; \xi^r) |\hat{Y}_k^r(\xi) - Y_k(\xi)|^2 \\ &= \sum_{\xi \in \Omega^r} w(\xi; \xi^r) |{}^0a_k^r + {}^1a_k^r(\xi - \xi^r) - Y_k(\xi)|^2 \end{aligned} \quad (10)$$

We will show how this least squares problem can be solved efficiently using convolutions in Section 3.2.

2.3 Measurement update

The update stage of the filter aims to correct the current estimate of optical-flow Φ_{k+}^r , obtained in the propagation stage, by using new available image data. This stage in the proposed algorithm is based on the constant brightness assumption of classical optical flow algorithms [Barron *et al.*, 1994]; an assumption that can also be characterised as a transport identity. The constant brightness assumption for the full spatio-temporal image can be written

$$Y(\tau, \zeta_{k+1}(\tau; \dot{\zeta})) = Y(k+1, \dot{\zeta}), \quad (11)$$

where $\zeta_{k+1}(\tau; \dot{\zeta})$ is given by (1) and $\tau \in (k+1-\delta_{k+1}, k+1+\delta_{k+1})$. That is the image intensity is the transport of the intensity at the point $\dot{\zeta}$ along the flow $\zeta_{k+1}(\tau; \dot{\zeta})$.

The full spatio-temporal image is not available, only the sampled version $Y_k(\xi)$ is measured. A continuous spatial approximate model of the image is provided by the image model $\hat{Y}_{k+1}^r(\zeta)$ defined in Section 2.2. We extend this model implicitly to a spatio-temporal model in the vicinity of a point $(k+1, \xi^r)$ using the analogy of the transport identity (11)

$$\hat{Y}_{k+1}^r(\tau, \zeta_{k+1}(\tau; \dot{\zeta})) := \hat{Y}_{k+1}^r(\dot{\zeta}).$$

This approximation of the “true” spatio-temporal image is used to compute the update stage of the algorithm.

Using the initial condition $\dot{\zeta} = \xi^r$, taking the total time differential of (11), and evaluating at $\tau = k+1$

yields

$$\begin{aligned} 0 &= \frac{d}{d\tau} \hat{Y}_{k+1}^r(\tau; \zeta_{k+1}(\tau; \xi^r)) \Big|_{\tau=0} \\ &= \frac{\partial}{\partial \zeta} \hat{Y}_{k+1}^r(k+1; \zeta) \Big|_{\zeta=\xi^r} \cdot \Phi_{k+1}^r \\ &\quad + \frac{d}{d\tau} \hat{Y}_{k+1}^r(\tau; \xi^r) \Big|_{\tau=k+1} \end{aligned} \quad (12)$$

recalling (4).

From the image model (9) one may write

$$\begin{aligned} \frac{\partial}{\partial \zeta} \hat{Y}_{k+1}^r(k+1; \zeta) \Big|_{\zeta=\xi^r} &= \frac{\partial}{\partial \zeta} \hat{Y}_{k+1}^r(\zeta) \Big|_{\zeta=\xi^r} \\ &= {}^1a_{k+1}^r, \end{aligned} \quad (13)$$

Using a backward Euler discretization in the temporal variable along with the approximation that $\hat{Y}_{k+1}^r(k; \xi^r) = \hat{Y}_k^r(\xi^r)$ one obtains

$$\frac{\partial}{\partial \tau} \hat{Y}_{k+1}^r(\tau; \xi^r) \Big|_{\tau=k+1} \approx {}^0a_{k+1}^r - {}^0a_k^r, \quad (14)$$

Substituting into (12) yields

$${}^1a_{k+1}^r \Phi_{k+1}^r + {}^0a_{k+1}^r - {}^0a_k^r \approx 0 \quad (15)$$

where the approximate equality comes from the nature of the approximations made.

Equation (15) can be thought of as providing information about the flow Φ_{k+1}^r in the form of a constraint equation. The information is not complete, indeed, only the component ${}^1a_{k+1}^r \Phi_{k+1}^r$ of the flow in direction of the image gradient contributes to (15). In classical flow computation [Barron *et al.*, 1994] this property is known as the aperture problem and is a numerical issue for classical algorithms since the full flow estimate must be computed at every time instant. In the proposed algorithm, we trade off between the new information provided by (15) and the prior estimate Φ_{k+}^r using a least squares criterion.

Let $\gamma > 0$ be a suitable positive constant and consider the least squares cost function

$$\begin{aligned} \varepsilon_2(\Phi) &= |{}^1a_{k+1}^r \Phi + {}^0a_{k+1}^r - {}^0a_k^r|^2 \\ &\quad + \gamma \|\Phi - \Phi_{k+}^r\|^2 \end{aligned} \quad (16)$$

The updated flow estimate is chosen to be

$$\Phi_{k+1}^r := \arg \min \varepsilon_2(\Phi). \quad (17)$$

Details of the numerical implementation are given in Section 3.3. The relative scaling factor γ can be thought of as encoding the trade-off between propagated flow prior information and new information. If the image data is

noisy and poorly conditioned, then γ should be chosen to be large, while if the knowledge of the flow is poor then γ should be chosen to be small. It is important that γ is always chosen to be positive since the least squares cost is ill-conditioned for $\gamma = 0$, the classical aperture problem.

Note that when ${}^1a_{k+1}^r = 0$ then the left-hand term in (16) does not depend on Φ_{k+1}^r and the new flow estimate is equal to Φ_{k+}^r . That is if the image has zero gradient then Equation (15) does not contain any information on the optical-flow. In particular, the amount of information contained in (15) is proportional to the magnitude of the image-gradient. In practice, new information on the flow field is most available in regions of the image where the image gradient is high, i.e., at edges.

3 Implementation Details

The proposed algorithm admits a simple, computationally friendly, implementation in terms of image convolutions and simple matrix multiplications. This section discusses the details for computing the image model parameters and the propagation and measurement update stages of the filter.

3.1 Propagation

A finite-difference scheme for the propagation stage can be derived directly from equation (8). To avoid numerical issues, the partial derivatives that form the Jacobian matrix J_k^r must be computed with data from the direction the flow comes from (upwind direction), preserving data causality [Thomas, 1995]. To achieve this, partial derivatives in x and y are computed in parallel both using backward and forward kernels. The appropriate derivative output is selected according to the direction of the optical flow field.

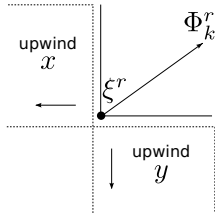


Figure 2: Upwind region for a flow vector. Data within this region can be used to compute partial derivatives.

To reduce the noise in the transport process, the scalar fields ${}^x\phi_k^r, {}^y\phi_k^r$ are smoothed with a low-pass kernel and then used as the input data for equation (8). The low pass smoothing of the flow-field is a crucial part of the algorithm in two ways. Firstly, it is important in reducing the noise in the Jacobian estimates. Secondly, and most crucially, it introduces a diffusion of information into the flow propagation. In particular, the update flow at a control point ξ^r will now depend on flow in the

$$\begin{aligned} \overleftarrow{\frac{\partial}{\partial x}} &= \begin{bmatrix} 0 & +1 & -1 \end{bmatrix} & \overrightarrow{\frac{\partial}{\partial x}} &= \begin{bmatrix} +1 & -1 & 0 \end{bmatrix} \\ \overleftarrow{\frac{\partial}{\partial y}} &= \begin{bmatrix} 0 \\ +1 \\ -1 \end{bmatrix} & \overrightarrow{\frac{\partial}{\partial y}} &= \begin{bmatrix} +1 \\ -1 \\ 0 \end{bmatrix} \end{aligned}$$

Figure 3: Partial derivative kernels (backward and forward) for the computation of Jacobian matrix of optical-flow.

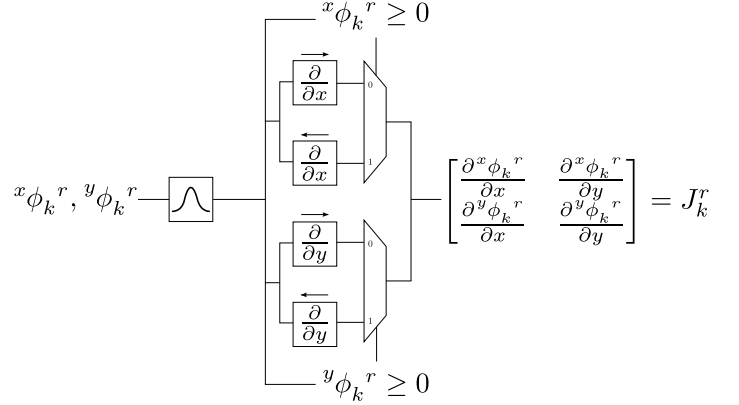


Figure 4: Computation of partial derivatives in the up-stream direction of the optical-flow field.

neighbourhood of the control point, as well as the flow at that particular point. This allows flow information to propagate outwards from a point and fill in flow in the neighbouring region. This is seen in the experimental verification of the algorithm in the way that strong edges in the image generate flow that then diffuses out across neighbouring regions in the image where there is insufficient intensity gradient to identify flow.

3.2 Image model

The support region Ω^r is chosen to be symmetric about each control point ξ^r . It follows that the solution to the least-squares cost function (10) for the image model parameters $\{{}^0a_k^r, {}^1a_k^r\}$ can be written as a 2D-convolution process. In general, the solution to the weighted least squares has the form

$${}^0a_k^r = \sum w(\xi; \xi^r) Y_k(\xi) \quad (18)$$

$${}^x a_k^r = \frac{\sum w(\xi; \xi^r) Y_k(\xi) (x - x_k^r)}{\sum w(\xi; \xi^r) (x - x_k^r)^2} \quad (19)$$

$${}^y a_k^r = \frac{\sum w(\xi; \xi^r) Y_k(\xi) (y - y_k^r)}{\sum w(\xi; \xi^r) (y - y_k^r)^2} \quad (20)$$

By choosing the weighting function $w(\xi; \xi^r)$ to be separable, these equations can be implemented as a series of 1D convolutions in x - and y - directions. The kernels for $\Omega^r = 5 \times 5$ and their inter-connection are depicted in Figures 5 and 6 respectively.

$$\begin{aligned} \boxed{\mathcal{G}_x} &= \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \\ \boxed{\mathcal{G}_a} &= \frac{1}{8} \begin{bmatrix} +1 & +2 & 0 & -2 & -1 \end{bmatrix} \\ \boxed{\mathcal{G}_y} &= \frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \quad \boxed{\mathcal{G}_a} = \frac{1}{8} \begin{bmatrix} +1 \\ +2 \\ 0 \\ -2 \\ -1 \end{bmatrix} \end{aligned}$$

Figure 5: Image model convolution kernels.

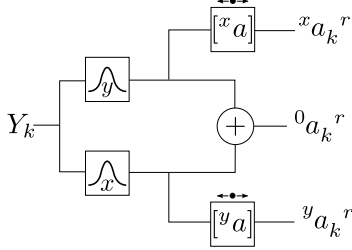


Figure 6: Image model computation.

3.3 Flow Update

The solution to the least-squares update cost function in equation (17) can be expressed in matrix form as

$$M\Phi_{k+1}^r = q \quad (21)$$

with

$$M = \begin{pmatrix} \gamma + (x_{k+1}^r)^2 & x_{k+1}^r y_{k+1}^r \\ x_{k+1}^r y_{k+1}^r & \gamma + (y_{k+1}^r)^2 \end{pmatrix} \quad (22)$$

$$q = \begin{pmatrix} \gamma \phi_{k+}^r + x_{k+1}^r [a_k^r - a_{k+1}^r] \\ \gamma \phi_{k+}^r + y_{k+1}^r [a_k^r - a_{k+1}^r] \end{pmatrix} \quad (23)$$

Given the small size of the linear system it is advantageous to use the adjugate matrix $A = \text{adj}(M)$ representation of the matrix inverse. Thus, solving the linear system yields

$$\Phi_{k+1}^r = \frac{1}{\det(M)} Aq. \quad (24)$$

For the particular matrix M it is easily verified that

$$\det(M) = \gamma(\gamma + \|a_{k+1}^r\|^2). \quad (25)$$

The solution for the optical-flow update is then

$$\Phi_{k+1}^r = \frac{1}{\det(M)} A \left(\gamma \Phi_{k+}^r + [a_k^r - a_{k+1}^r] a_{k+1}^{r\top} \right) \quad (26)$$

Note that the solution to the update cost function only uses the local image model parameters and the propagated flow vector at ξ^r , requiring basic arithmetic operations. The only numerical operation that is costly to implement on an FPGA is the single scalar inverse $1/\det(M)$.

4 Experimental Validation

The proposed filtering approach requires a considerable number of images in order to get smooth optical-flow fields. This is not an issue for a REV system with a high frame rate image sensor. However, it poses a problem to validate the algorithm against existing ground truth data sets. Optical-flow data sets such as the Middlebury data set [Baker *et al.*, 2011] offers very few frames and cannot be used. The MPI Sintel data set [Butler *et al.*, 2012] has considerably more frames per sequence, however, the large motion between frames makes it inappropriate for our assumption on high-frame rate, and hence relatively small optical-flow.

To validate the proposed algorithm we created a 3D environment in Blender with real-world dimensions and realistic textures*. Camera motion and intrinsic parameters are known and the frame rate is set to 300frames/second. True optical-flow field can be easily computed from the relative depth map and camera motion. As error metrics, we employed the end-point and angular error metrics defined in the Middlebury dataset.

The video sequence simulates the camera motion for 6 seconds (a total of 1800 images). The camera moves with a constant velocity of 1m/s (3.6km/hr or walking speed) in the x direction for the first two seconds (frames 0-600). The next two seconds (600-1200), the camera performs a circular trajectory with constant tangential velocity of 1m/s. The last two seconds (frames 1200-1800), it returns to a constant velocity of 1m/s in the x direction.

We employed a support region $\Omega^r = 5 \times 5$ for the computation of the image model, to balance the computations required for the filter and rejection of noise. For the propagation, we used a 5×5 average kernel as a low-pass filter for the optical-flow field. Additionally, we performed the propagation process in two time steps with $\Delta t = 0.5$ rather than in a single unit step. This helps to reduce the large displacements of the flow field into smaller and more numerically stable steps. For the update stage, γ has been set to 100. The video result for this experiment is available at <http://cecs.anu.edu.au/~jadarve/acra2014.html>

Figure 8 shows the estimated optical-flow at certain frames. The first 4 rows illustrate the warm-up process of the filter. At frame 10, flow estimate is primarily updating at image discontinuities where the high image gradient provides significant information to the update stage. As the camera continues to move (frames 60 and 200), flat regions in the image are filled in with flow estimate diffused from nearby image edges. This process is best visualised in the decrease of the end-point error

*One or more textures on this 3D model have been created with images from CGTextures.com. These images may not be redistributed by default, please visit www.cgtextures.com for more information

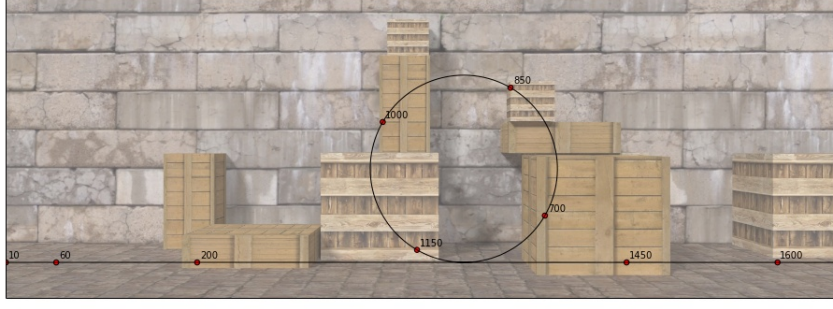


Figure 7: Overview of the 3D synthetic environment. Textured boxes are located at different distances from the camera. Sampled frames of the camera trajectory are used in Figure 8 for displaying the estimated optical-flow field.

(column 4).

The next 4 rows (frames 700, 850, 100, 1150) show the estimated flow during the circle motion of the camera. Note that this is a challenging manoeuvre in order to track optical flow, since the flow changes quickly, and varies significantly in direction. For the proposed algorithm, horizontal flow generated by vertical edge is difficult to “forget” during vertical motion, since the horizontal edge no longer provides information on the flow. Despite these difficulties, the observed end-point error of the flow is of the order of 0.5 pixels throughout the motion (except in the shadowed regions). Note how the angular error is significantly lower at image discontinuities where there is more information available in the update step. The last rows (frames 1450 and 1600) shows how the flow stabilises during constant velocity motion.

5 Discussion

The proposed algorithm is able to build dense optical-flow with relatively simple computations by exploiting the temporal nature of image data. Each control point ξ^r only requires data contained on its support domain Ω^r . As mentioned above, a support $\Omega^r = 5 \times 5$ pixels is a sensible compromise between computational load and robustness in the characterisation of the different state variables.

Since the algorithm is largely deterministic, it is easy to determine the number of calculations required for computing flow at a single control point ξ^r . Let $\Omega^r = N \times N$ be the size of the support domain for both the image model estimation and smoothing kernel in the propagation stage, and n_a the number of propagation steps. For the calculation, we will assume separable kernels (e.g. Gaussian) for the smoothing blocks of the filter. Table 1 shows the required computations for each filter stage.

Note that the number of computations in the update stage are independent of the size of Ω^r . Also, in practice,

	Add	Multiply	Div.
Image model			
- 1D Gaussians	$2(N - 1)$	$2N$	0
- ${}^x a^r, {}^y a^r$ kernels	$2(N - 2)$	$2(N - 1)$	0
- ${}^0 a^r$	1	0	0
Propagation			
- Smoothing ($\times 2$)	$4n_a(N - 1)$	$4n_a N$	0
- Jacobian elements	$8n_a$	0	0
- Transport (eq (8))	$6n_a$	$4n_a$	0
Update			
- Matrix M	2	3	0
- $\det(M)$	2	3	0
- Φ_{k+1}^r (eq (26))	5	10	1

Table 1: Arithmetic operations required for the proposed filter per reference pixel ξ^r .

the multiplications can be simplified by using smoothing weighting coefficients that are powers of 2 (as in Figure 5 for the image model parameters), or by using an average kernel, which does not require multiplications at all.

The local nature of the algorithm makes it suitable to be implemented in a REV platform using either a GPU or an FPGA computing device. In particular, the fixed convolutions for the image model parameters and the propagation of the flow field, makes the filter a perfect candidate for a digital circuit implementation, such as in an FPGA. Here we will consider the specifications of the FastVis REV system [FastVis, 2014] as the target system. Table 2 provides part of the specification of the platform. Note that the bandwidth of the camera data is too large to be transmitted to a desktop machine, and cannot be processed by a normal, micro-controller based, embedded platform. If an application is to take advantage of the high-resolution and high-frame rate of the image sensor, the processing has to be performed in situ. Currently, the simplest embedded technology capable of handling such bandwidth is an FPGA.

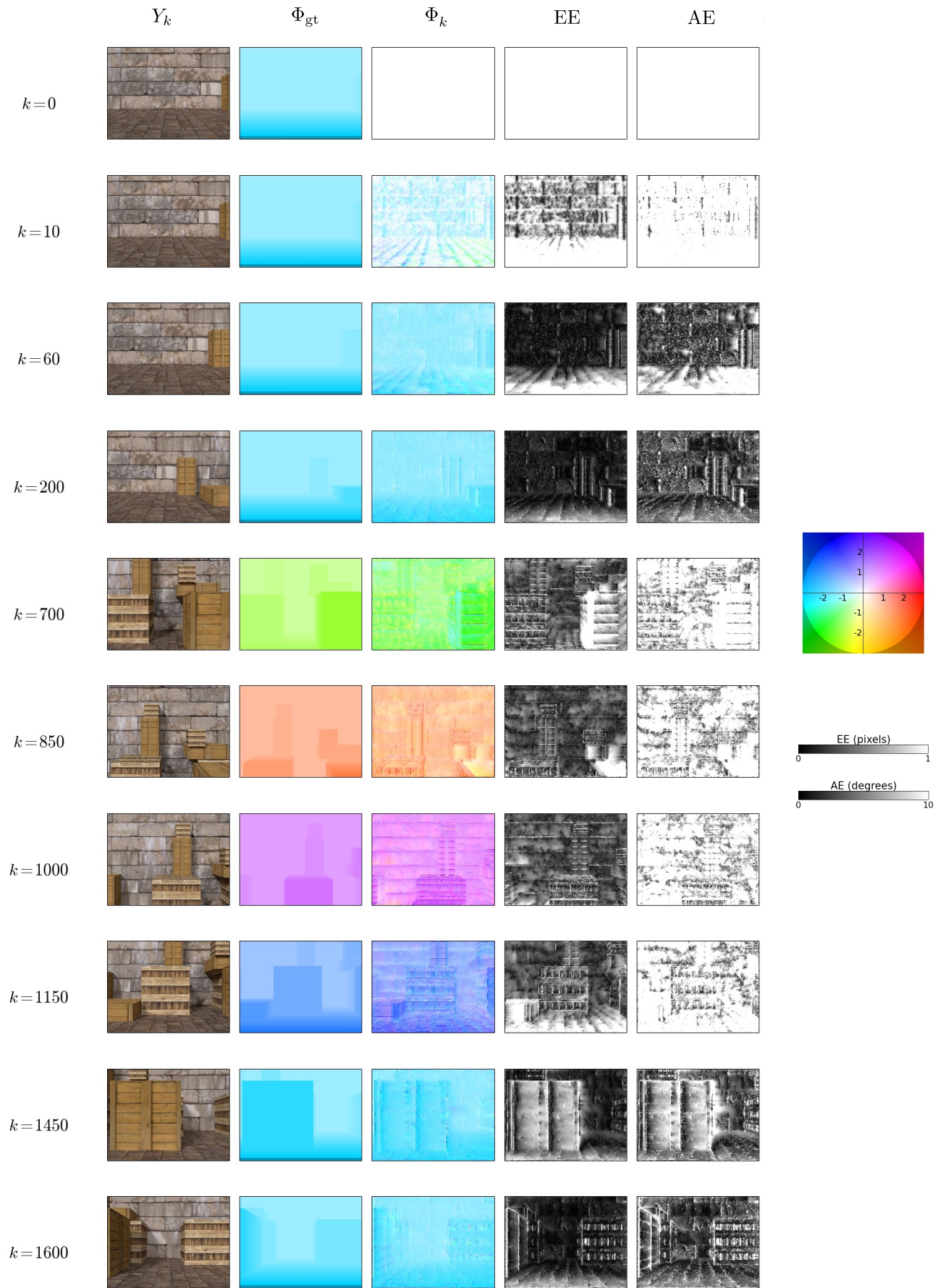


Figure 8: Ground truth validation. The camera (300 frames/second) moves with a known velocity for 6 seconds. Columns 2 and 3 show the color encoded ground-truth Φ_{gt} and estimated optical-flow Φ_k . End-point (EE, in pixels) and angular error (AE, in degrees) are computed (cols 4-5). A video result is available at <http://cecs.anu.edu.au/~jadarve/acra2014.html>

Sensor	CMV2000
Image resolution	2048x1088pixels
Frame rate	340 max
Input image bandwidth	7.68Gbps
Computing device	FPGA
Logic elements	149,000
FPGA memory	6.8 Mb
DSP Slices	160
Per-pixel operations	
Additions (32 bits)	~5,000
Multiplications (18 bits)	~140
Memory	SDRAM x 3
Total memory bandwidth	15Gbps
Output channel	USB 3.0
Output bandwidth	3.2 Gbps

Table 2: Specifications of the FastVis REV system.

However, designing algorithms for a FPGA platform is a difficult task. Typically, proposed systems [Diaz *et al.*, 2006; Aubépart and Franceschini, 2007; Plett *et al.*, 2012] use algorithms that use a low level of abstraction. The filter algorithm proposed here is well suited for a digital implementation for various reasons. The convolutions for the image model parameters connect directly to the input data stream from the image sensor, requiring only buffering of N rows of image intensity data. The propagation stage consumes data from memory and computes the backward and forward partial derivatives all in parallel, selecting the appropriate output from the direction of the flow field. Finally, the measurement update consumes the output streams from the image model and propagation blocks and updates the flow estimate, with the updated flow written back into the memory.

Consider the resources used for the proposed algorithm based on the FastVis system parameters. Consider a 1024×544 grid of control points $\{\xi^r\}$, corresponding to sub-sampling the image resolution by a factor of two in both x - and y - directions. We use a support $\Omega^r = 5 \times 5$ centered on control points, providing a four-times overlap of pixels, every pixel contributes to at least four support regions. We use a Gaussian smoothing kernel with coefficients given in Figure 5 and two propagation steps. For each $\{\xi^r\}$, we set $\{^0a^r, ^x\phi^r, ^y\phi^r\}$ to 8 bits resolution. The image model estimation requires a total of 15 additions and 1 multiplications per control point. The 2 steps propagation stage requires a total of 60 additions and 16 multiplications to be computed. Finally, the update stage requires 9 additions, 16 multiplications and 1 division operations per control point. The total number of computations is then 84 additions, 33 multiplications and 1 division per control point. The filter only requires about 6% of the total number of multipliers. The total required storage per frame is 1.5 MB. Since each mem-

ory chip has its own dedicated channel, we propose to use two of the memory chips in a double buffer design for the update and propagation stages of the algorithm to minimise read/write switching and conflicts. With a memory bandwidth of 596 MB/s per memory chip, the filter requires 90% of the available bandwidth of two memory chips at full frame rate 340fps. The third memory chip is held in reserve for post processing of the computed flow field and other memory read-write requirements.

It is interesting to note that the real bottle-neck in the implementation lies in the memory bandwidth. A dedicated FPGA design, with careful memory management avoiding switching between read/write modes on the SDRAM memory makes the proposed design viable. The computational resources available (over 90%) of the FPGA multipliers and a separate dedicated memory buffer with 596MB/s bandwidth, makes it a straightforward matter to add a post processing stage to compute direct control cues for mobile robotics applications. Objects that can easily be computed are, divergence of the flow field (trace of J_k^r), flow averages in regions, cross-correlation of flow with desired flow profile, thresholding and identification of high flow areas, etc. These control cues can be streamed to the USB output channel and used in control of robotic vehicles.

6 Conclusions

This article developed a filter type approach for the computation of optical-flow well suited for real-time dense optical-flow computation. The filter is based on an update and propagation loop that incrementally builds the optical-flow field using the stream of images and previously computed flow. The propagation stage takes the current flow field and uses a transport process to predict its next time value. The update stage in turn, takes the predicted flow and corrects it using new image data. The filter loop naturally fills in regions of the image where computing flow is not possible. Ground truth validation is provided by simulating a high-speed camera moving in a 3D environment. In terms of implementation, the proposed algorithm is well suited for implementation on a Rapid Embedded Vision (REV) system.

References

- [Aubépart and Franceschini, 2007] F. Aubépart and N. Franceschini. Bio-inspired optic flow sensors based on fpga: Application to micro-air-vehicles. *Microprocessors and Microsystems*, 31(6):408 – 419, 2007. Special Issue on Sensor Systems.
- [Baker *et al.*, 2011] Simon Baker, Daniel Scharstein, J. Lewis, Stefan Roth, Michael Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92:1–31, 2011.

- [Barron *et al.*, 1994] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.
- [Benosman *et al.*, 2014] R. Benosman, C. Clercq, X. Lagorce, Sio-Hoi Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):407–417, Feb 2014.
- [Beyeler *et al.*, 2009] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 27(3):201–219, 2009.
- [Black and Anandan, 1991] M.J. Black and P. Anandan. Robust dynamic motion estimation over time. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1991. CVPR '91.*, pages 296–302, Jun 1991.
- [Black, 1992] M.J. Black. *Robust Incremental Optical Flow*. PhD thesis, Yale University, 1992.
- [Butler *et al.*, 2012] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conf. on Computer Vision (ECCV)*, pages 611–625, 2012.
- [Chahl *et al.*, 2004] J. S. Chahl, M. V. Srinivasan, and S. W. Zhang. Landing strategies in honeybees and applications to uninhabited airborne vehicles. *The International Journal of Robotics Research*, 23:101–110, Feb 2004.
- [Coombs *et al.*, 1998] D. Coombs, M. Herman, T. Hong, and M. Nashman. Real-time obstacle avoidance using central flow divergence and peripheral flow. *IEEE Transactions on Robotics and Automation*, 14(1):49–59, 1998.
- [Diaz *et al.*, 2006] J. Diaz, E. Ros, F. Pelayo, E.M. Ortigosa, and S. Mota. Fpga-based real-time optical-flow system. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(2):274–279, Feb 2006.
- [FastVis, 2014] FastVis. Fastvis company web page. <http://fastvis.com/>, 2014. last visited November 2014.
- [Fleet and Langley, 1995] D.J. Fleet and K. Langley. Recursive filters for optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):61–67, jan 1995.
- [Heeger, 1988] David J. Heeger. Optical flow using spatiotemporal filters. *International Journal of Computer Vision*, 1:279–302, 1988.
- [Honegger *et al.*, 2013] Dominik Honegger, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [Lucas and Kanade, 1981] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Imaging Understanding Workshop. Proceedings.*, pages 121–130, 1981.
- [McCarthy and Barnes, 2004] C. McCarthy and N. Barnes. Performance of optical flow techniques for indoor navigation with a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5093–5098, 2004.
- [McCarthy *et al.*, 2008] C. McCarthy, N. Barnes, and R. Mahony. A robust docking strategy for a mobile robot using flow field divergence. *IEEE Transactions on Robotics*, 24(4):832–842, Aug. 2008.
- [Plett *et al.*, 2012] Johannes Plett, Armin Bahl, Martin Buss, Kolja Kühnlenz, and Alexander Borst. Bio-inspired visual ego-rotation sensor for mavs. *Biological Cybernetics*, 106:51–63, 2012.
- [Ruffier and Franceschini, 2005a] F. Ruffier and N. Franceschini. Optic flow regulation: The key to aircraft automatic guidance. *Robotics and Autonomous Systems*, 50:177–194, 2005.
- [Ruffier and Franceschini, 2005b] Franck Ruffier and Nicolas Franceschini. Optic flow regulation: the key to aircraft automatic guidance. *Robotics and Autonomous Systems*, 50(4):177 – 194, 2005.
- [Santos-Victor and Sandini, 1997] J. Santos-Victor and G. Sandini. Visual behaviours for docking. *Computer Vision and Image Understanding*, 67(3):223–238, September 1997.
- [Srinivasan and Zhang, 2004] M.V. Srinivasan and S. Zhang. Visual motor computations in insects. *Annual Review of Neuroscience*, 27:679–696, July 2004.
- [Srinivasan *et al.*, 2006] M.V. Srinivasan, S. Thurgood, and D. Soccol. An optical system for guidance of terrain following in uavs. In *Proceedings of the IEEE International Conference on Video and Signal Based Surveillance, 2006. AVSS '06.*, pages 51–51, Nov. 2006.
- [Srinivasan, 1994] M. V. Srinivasan. An image interpolation technique for the computation of optical flow and egomotion. *Biological Cybernetics*, 71:401–415, 1994.
- [Thomas, 1995] James William Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer, 1995.