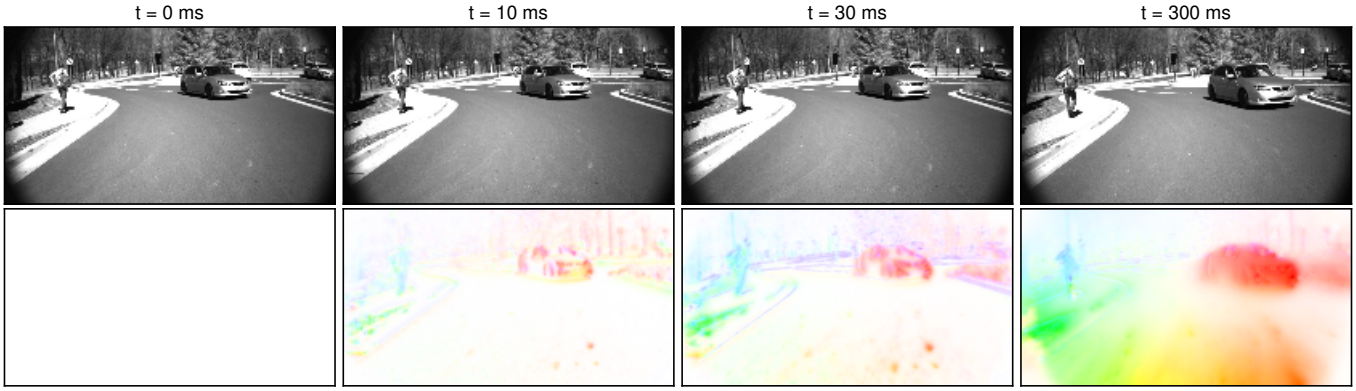
A Filter Formulation for Computing Real Time Optical Flow

Juan David Adarve and Robert Mahony



300 Hz incremental computation of optical flow. Image data is incorporated over time to create dense flow estimation.

***Abstract*—Dense optical flow is a crucial visual cue for obstacle avoidance and motion control for robotic systems functioning in complex unstructured environments. In order to compute image motion for modern highly dynamic robots, it is necessary to use high speed vision systems. To perceive small and thin objects such as tree branches, fence poles, and other similar objects, high resolution vision systems must be used. The data stream from a high-resolution, high-speed vision system will saturate the onboard data bus and overload the onboard processor of almost all existing robotic systems. To implement such a vision system, we believe it is necessary to process the data *in situ* at the camera using dedicated processing hardware, a vision processing paradigm that we term Rapid Embedded Vision (REV) systems. Even then, the task of processing multiple hundred hertz of dense high-resolution images is not possible using classical algorithms and classical computing hardware. A new processing paradigm that actively exploits the high frame rate of the image sequences, and high performance computing hardware such as GPU or FPGA must be employed. This paper proposes a filtering algorithm for the computation of dense optical flow fields in real time. The filter is designed as a pyramidal structure of update and propagation loops, where an optical flow state is constantly refined with new image data from the camera. The computational properties of the proposed algorithm makes it well suited for implementation in GPU and FPGA systems. We present results from a GPU implementation achieving frame rates in the order of 800 Hz at VGA resolution. Experimental validation and comparison is provided for both synthetic and real life high speed video sequences at frame rates of 300 Hz and 1016×544 pixel resolution.**

***Index Terms*—Visual Tracking; Visual-Based Navigation.**

Manuscript received: August, 31, 2015; Revised December 15, 2015; Accepted January 28, 2016.

This paper was recommended for publication by Editor Jana Kosecka upon evaluation of the Associate Editor and Reviewers' comments.

Juan David Adarve and Robert Mahony are with the Research School of Engineering, The Australian National University, 2601 Canberra, Australia, {juan.adarve. robert.mahony}@anu.edu.au and the Australian Centre for Robotic Vision (ACRV). http://www.roboticvision.org. Source code available at https://github.com/jadarve/optical-flow-filter

Digital Object Identifier (DOI): see top of this page.

I. Introduction

OPTICAL flow is a widely used sensor cue in Robotics. Applications span visual odometry [1], landing of aerial vehicles [2] and obstacle avoidance [3]. A common characteristic of these applications is the requirement of the underlying optical flow method to run at real-time frame rates. New advances in optical flow in the Computer Vision community, listed on evaluation datasets [4], [5], [6], typically trade off computational complexity for accuracy in the estimated flow fields, leading to slow algorithms not suitable for integration into real-time systems. For this reason, relatively simple algorithms [7], [8] are used in most robotic systems, due primarily to their reduced complexity and efficient implementations.

Typical optical flow algorithms compute a full dense flow estimate from each sequential pair of images. The early work of Lucas and Kanade [7] and Horn and Schunck [9] marked a distinction between local and global optimization methods. Real-time methods [10], [11], [12] typically follow a local minimization approach as it makes it possible to parallelize computations for each pixel using a small window of data. However, local methods are always highly dependent on texture available in the image, and where there is little or no texture they are highly inaccurate. When high frame rate image sequences are considered, then it is possible to use multiple frames of data and form temporal correlation from the data to compute optical flow [13], [14], [15]. In this case, algorithms exploit the intrinsic properties of high speed video data, that is, the relatively small changes in appearance between consecutive images and the approximately linear behavior of

the dynamics of the ego-motion of the vehicle as observed at high frame rates.

Alternatively, the optical flow estimate from one image can be considered as a *state* that is first propagated to the next time step and then updated based on the latest image information. This architecture is described in the work by Black [16]. In his work, the prediction stage is computed as an image warp followed by a resampling to create a regular grid, and the update stage uses robust statistical methods to remove outliers in the computations.

This paper presents a filtering algorithm for the computation of dense optical flow fields. We use a similar update-prediction architecture as Black [16]. However, in the proposed method we model the prediction stage of the algorithm by continuous time evolution of the image along the optical flow field. We derive a system of Partial Differential Equations representing a fluid transport process, for which we propose a fast numerical scheme based on finite difference methods. In the update stage, we use the standard brightness constancy assumption present in differential optical flow methods [17], and add a temporal regularization term based on the predicted flow to provide trade off between new information acquired from the latest image and propagated state information. To support large displacements, a pyramid of filter loops is constructed and information flows in a top-down fashion. Top level captures the large displacement optical flow while lower levels of the pyramid capture fine details of the flow field.

The computational structure of the algorithm is completely parallelizable and well suited for both GPU and FPGA systems. On commodity GPU hardware, we achieved frame rates in the order of 800 Hz at VGA resolution for flow fields up to 4 pixels. We compared our method to available GPU methods [18], [19] and [20] on both synthetic and real life high speed video at 300 Hz. At such frame rate, our method performs better in terms of average error and runtime performance.

The paper is organized as follows: Section II presents the filtering algorithm architecture. Section III develops the details on the numerical scheme for the propagation stage. Details on GPU implementation are provided in Section IV. Experimental validation is presented in Section V. Finally, Section VI provides future work and concluding remarks.

II. Filter architecture

Denote the input image from the camera at time k by Y^{k}. Let H be the number of levels of the pyramidal structure, and $h = 0, \dots H-1$ the level index. An image pyramid $\left\{{}^{0}Y^{\mathrm{k}}, \dots, {}^{\mathrm{H\text{-}1}}Y^{\mathrm{k}}\right\}$ is constructed by successive low pass filtering and subsampling the input image by a factor of 2. At each level, and for each pixel p, we use raw image data ${}^{\mathrm{h}}Y^{\mathrm{k+1}}$ in a neighborhood Ω_p around p to fit a linear brightness model with parameters ${}^{\mathrm{h}}a_1^{\mathrm{k+1}}$ for the linear gradient vector and ${}^{\mathrm{h}}a_0^{\mathrm{k+1}}$ for the average intensity. Let $w(p,q)$ be a Gaussian weight function centered at p. The parameters $\left\{{}^{\mathrm{h}}a_1^{\mathrm{k+1}}, {}^{\mathrm{h}}a_0^{\mathrm{k+1}}\right\}$ are determined as the arg min of the cost function

$$ {}^{\mathrm{h}}\varepsilon_{\mathrm{I}} = \sum_{q\in\Omega_p} w(p,q) \left\| {}^{\mathrm{h}}Y_q^{\mathrm{k+1}} - {}^{\mathrm{h}}a_1^{\mathrm{k+1}}(p-q) - {}^{\mathrm{h}}a_0^{\mathrm{k+1}} \right\|^2 . \quad (1) $$

These parameters can efficiently been computed by applying a series of 1D convolutions in row and column directions. We used a support window of 5 pixels and weight vector $w = [1\ 4\ 6\ 4\ 1]/16$. The brightness parameters form the measurement data used in the update stage of the filter at each pyramid level.

The filter state $X^k = \left\{{}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k}}, {}^{\mathrm{H\text{-}2}}\Delta\Phi^{\mathrm{k}}, \dots, {}^{0}\Delta\Phi^{\mathrm{k}}\right\}$ is formed by a coarse estimate of optical flow ${}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k}} = ({}^{\mathrm{H\text{-}1}}u^{\mathrm{k}}, {}^{\mathrm{H\text{-}1}}v^{\mathrm{k}})$ at top level $H-1$, and successive refinements ${}^{\mathrm{k}}\Delta\Phi^{\mathrm{h}} = ({}^{\mathrm{h}}\Delta u^{\mathrm{k}}, {}^{\mathrm{h}}\Delta v^{\mathrm{k}})$ of flow for each lower level. The reconstruction of optical flow at lower levels is achieved by iterating Equation (2) for $h = H-2, \dots, 0$

$$ {}^{\mathrm{h}}\Phi^{\mathrm{k}} = 2\,{}^{\mathrm{h+1:h}}\Phi^{\mathrm{k}} + {}^{\mathrm{h}}\Delta\Phi^{\mathrm{k}} \quad (2) $$

where ${}^{\mathrm{h+1:h}}\Phi^{\mathrm{k}}$ denotes the upsampling of flow from level $h+1$ to h. The upsampling is achieved by linearly interpolating ${}^{\mathrm{h+1}}\Phi^{\mathrm{k}}$ to generate a flow field of the same resolution as the input image at level h. The interpolated field is then multiplied by 2 to consider the new pixel sampling.

The filter algorithm operates in a top-down fashion. At time k, the propagation stage takes place for all levels of the pyramid. In this stage, a prediction of the state variable for time $k+1$ is calculated based on current estimation. We denote the propagated state as $X^{k+} = \left\{{}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k+}}, {}^{\mathrm{H\text{-}2}}\Delta\Phi^{\mathrm{k+}}, \dots, {}^{0}\Delta\Phi^{\mathrm{k+}}\right\}$, where superscript $k+$ refers to the predicted state for time $k+1$ before the update stage takes place. Once the propagation is completed, the predicted state is combined with the newly computed brightness parameters at each level to create an updated state $X^{k+1} = \left\{{}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k+1}}, {}^{\mathrm{H\text{-}2}}\Delta\Phi^{\mathrm{k+1}}, \dots, {}^{0}\Delta\Phi^{\mathrm{k+1}}\right\}$.

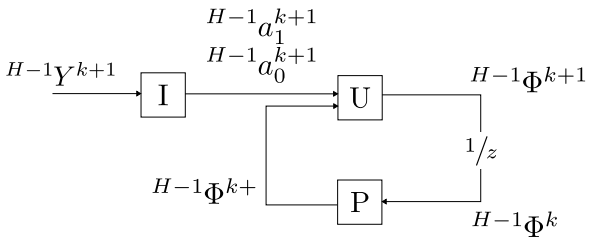
The filter pyramid consists of two types of filter loops. Figure 1a illustrates the top layer filter. Brightness model parameters $\left\{{}^{\mathrm{H\text{-}1}}a_1^{\mathrm{k+1}}, {}^{\mathrm{H\text{-}1}}a_0^{\mathrm{k+1}}\right\}$ are extracted from input image ${}^{\mathrm{H\text{-}1}}Y^{\mathrm{k+1}}$, and form the measurement data for the filter loop, formed by update and propagation stages. In the propagation stage, the old estimate of coarse optical flow ${}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k}}$ is propagated to create a prediction ${}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k+}}$ for time $k+1$. This prediction, together with the new brightness parameters are used in the update block to create new flow estimate ${}^{\mathrm{H\text{-}1}}\Phi^{\mathrm{k+1}}$.

The second type of filter loop is illustrated in Figure 1b, and is used for all lower levels of the pyramid $h = H-2, \dots, 0$. Brightness parameters $\left\{{}^{\mathrm{h}}a_1^{\mathrm{k+1}}, {}^{\mathrm{h}}a_0^{\mathrm{k+1}}\right\}$ are estimated from input image ${}^{\mathrm{h}}Y^{\mathrm{k+1}}$. State ${}^{\mathrm{h}}\Delta\Phi^{\mathrm{k}}$ is combined with optical flow from level $h+1$ to reconstruct the flow field ${}^{\mathrm{h}}\Phi^{\mathrm{k}}$ of this level following Equation (2). In the propagation, old optical flow ${}^{\mathrm{h}}\Phi^{\mathrm{k}}$ is used to propagate ${}^{\mathrm{h}}\Delta\Phi^{\mathrm{k}}$ and ${}^{\mathrm{h}}a_0^{\mathrm{k}}$ to produce predictions ${}^{\mathrm{h}}\Delta\Phi^{\mathrm{k+}}$ and ${}^{\mathrm{h}}a_0^{\mathrm{k+}}$. Notice that ${}^{\mathrm{h}}a_0^{\mathrm{k+}}$ represents a prediction of the image at $k+1$ given the old optical flow. Under the brightness constancy assumption, this prediction is expected to be close to the newly computed ${}^{\mathrm{h}}a_0^{\mathrm{k+1}}$ parameter. Thus, any difference between ${}^{\mathrm{h}}a_0^{\mathrm{k+}}$ and ${}^{\mathrm{h}}a_0^{\mathrm{k+1}}$ will be due to some small change, *i.e.*, ${}^{\mathrm{h}}\Delta\Phi^{\mathrm{k+1}}$, between new and old optical flow at this level. The purpose of the update stage is then to find the increment ${}^{\mathrm{h}}\Delta\Phi^{\mathrm{k+1}}$ that corrects the flow at this level.

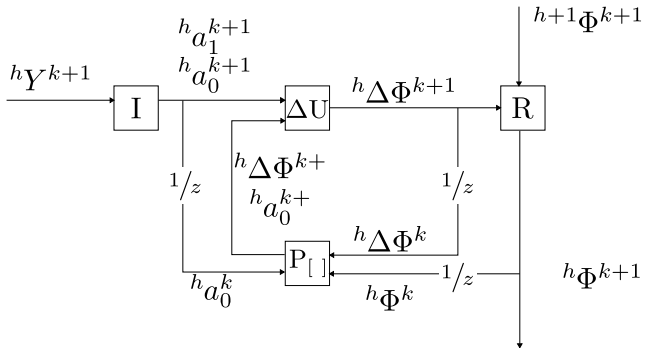
A. State propagation

Consider optical flow field $\Phi(\xi, t)$ in continuous variables $\xi \in \mathbb{R}^2$ and $t \in \mathbb{R}$ for space and time respectively, and let

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2532928, IEEE Robotics and Automation Letters



(a) Top level filter loop. In the image preprocessing block [I], brightness parameters are estimated from the new image and enter the filter loop. In the propagation block [P], a prediction of optical flow for the next time iteration is created. In the update [U], this prediction together with the brightness parameters create a new estimation of the flow field.



(b) Lower levels filter scheme. In the reconstruction block [R], optical flow for this level is reconstructed using flow from level above. Brightness parameters from the new image are computed at [I]. In the Propagation block [$P_{[]}$], old flow field is used to propagate the old state and the constant brightness parameter. These predictions are combined with new brightness parameters at the Update block [ΔU] to create a new state estimate.

Fig. 1: Pyramidal filter architecture.

$\xi(0)$ and $\Phi(\xi(0),0)$ be some known initial conditions. The evolution of $\xi(t)$ over time is given by

$$\dot{\xi}(t) = \Phi(\xi(t),t) \tag{3}$$

That is, at time t, the point moves with a velocity equal to the optical flow at $\xi(t)$. It is possible to apply the same principle to the optical flow field. In this case, the flow field is transported with a velocity equal to the optical flow itself. Assuming constant flow over time, the evolution of optical flow is defined as

$$\Phi(\xi(t),t) = \Phi(\xi(0),0) \tag{4}$$

That is, the value of optical flow at any given time is fully characterized by initial conditions $\Phi(\xi(0),0)$. Expressing Equation (4) as a total derivative, we obtain

$$\frac{d}{dt}\Phi(\xi(t),t) \approx 0 \tag{5}$$

and expanding Equation (5) to its partial terms, we obtain a partial differential equation describing the transport of optical flow

$$\frac{\partial\Phi}{\partial\xi}\Phi + \frac{\partial\Phi}{\partial t} \approx 0 \tag{6}$$

Equation (6) belongs to the family of non-linear hyperbolic PDE equations used in modeling several types of transport processes [21].

Notice that we used an approximation to zero in Equations (5) and (6). More precisely, there is an additional small source term due to the dependence of the optical flow on changing depth of the scene. For a camera recording video at high frame rates, this term is small and can be ignored.

Following a similar approach, it is possible to model the transport of an arbitrary scalar field $c(\xi(t),t) \in \mathbb{R}$ by optical flow, and derive a transport equation

$$\frac{\partial c}{\partial\xi}\Phi + \frac{\partial c}{\partial t} = 0 \tag{7}$$

Equation (7) has the same formulation as the brightness constancy assumption in differential methods for computing optical flow [17]. In this case, Φ is known and the goal is to find a prediction $c(t)$ for some time t given Φ.

Equations (6) and (7) are the fundamental PDEs that model the propagation of the filter state variables in our method. At top level, the propagation of the coarse optical flow $^{h}\Phi^{k}$ is governed by Equation (6)

$$\frac{\partial^{H-1}\Phi}{\partial\xi}{}^{H-1}\Phi + \frac{\partial^{H-1}\Phi}{\partial t} = 0 \tag{8}$$

Provided with initial conditions $^{H-1}\Phi(0) := {}^{H-1}\Phi^{k}$, we are interested in finding a solution $^{H-1}\Phi(1) =: {}^{H-1}\Phi^{k+}$ at time $t = 1$ corresponding to the propagated flow field for next time step.

For lower levels, where state $^{h}\Delta\Phi^{k}$ and old constant brightness parameter $^{h}a_0^{k}$ are transported by flow $^{h}\Phi^{k}$, the propagation is modeled by the system of PDEs

$$\frac{\partial^{h}\Delta\Phi}{\partial\xi}{}^{h}\Phi + \frac{\partial^{h}\Delta\Phi}{\partial t} = 0 \tag{9}$$

$$\frac{\partial^{h}a_0}{\partial\xi}{}^{h}\Phi + \frac{\partial^{h}a_0}{\partial t} = 0 \tag{10}$$

$$\frac{\partial^{h}\Phi}{\partial\xi}{}^{h}\Phi + \frac{\partial^{h}\Phi}{\partial t} = 0 \tag{11}$$

This system models the transport of $^{h}\Delta\Phi$ and $^{h}a_0^{k}$ by the flow $^{h}\Phi$, simultaneously being transported by itself. Initial conditions of the system are set to $^{h}\Delta\Phi(0) := {}^{h}\Delta\Phi^{k}$, $^{h}a_0(0) := {}^{h}a_0^{k}$ and $^{h}\Phi(0) := {}^{h}\Phi^{k}$ and it is solved at time $t = 1$ for $^{h}\Delta\Phi(1) =: {}^{h}\Delta\Phi^{k+}$ and $^{h}a_0(1) =: {}^{h}a_0^{k+}$. Notice that, while there is a solution $^{h}\Phi(1)$ for the optical flow, this solution never abandons the propagation block of the filter, and it can be considered as an internal state in the propagation. Details on the numerical solution to these equations are provided in Section III.

B. State update

In the update stage, a new state estimate X^{k+1} is produced by correcting the predicted state X^{k+} using new image data Y^{k+1} from the camera. First, at level $H-1$, the predicted optical flow $^{H-1}\Phi^{k+}$ is updated using brightness parameters $^{H-1}a_1^{k+1}$ and $^{H-1}a_0^{k+1}$. The update cost function

$$\begin{aligned} {}^{H-1}\varepsilon_U = &\left\| {}^{H-1}a_1^{k+1} \cdot {}^{H-1}\Phi^{k+1} + {}^{H-1}a_0^{k+1} - {}^{H-1}a_0^{k} \right\|^2 \\ &+ {}^{H-1}\gamma \left\| {}^{H-1}\Phi^{k+1} - {}^{H-1}\Phi^{k+} \right\|^2 \end{aligned} \tag{12}$$

consists of two terms. First, a data term relates the old image, characterized by $^{H-1}a_0^{k}$, and new image with brightness

parameters ${}^{\text{H-1}}a_1^{\text{k+1}}$ and ${}^{\text{H-1}}a_0^{\text{k+1}}$. This is similar to the brightness constancy constraint in differential optical flow techniques [17]. Second term is a regularization term that relates the propagated flow ${}^{\text{H-1}}\Phi^{\text{k+}}$ and the new estimate. Parameter ${}^{\text{H-1}}\gamma$ controls the weight of the regularization term relative to the data term. In textureless regions of the image, where ${}^{\text{H-1}}a_1^{\text{k+1}}$ is nearly zero, the predicted flow ${}^{\text{H-1}}\Phi^{\text{k+}}$ will act as best estimate of optical flow for time $k+1$.

With some algebraic manipulation, Equation (12) can be expressed as a linear system of the form

$$M\,{}^{\text{H-1}}\Phi^{\text{k+1}} = q \tag{13}$$

where

$$M = \begin{pmatrix} {}^{\text{H-1}}\gamma + ({}^{\text{H-1}}a_x^{\text{k+1}})^2 & {}^{\text{H-1}}a_x^{\text{k+1}}\,{}^{\text{H-1}}a_y^{\text{k+1}} \\ {}^{\text{H-1}}a_x^{\text{k+1}}\,{}^{\text{H-1}}a_y^{\text{k+1}} & {}^{\text{H-1}}\gamma + ({}^{\text{H-1}}a_y^{\text{k+1}})^2 \end{pmatrix} \tag{14}$$

$$q = \begin{pmatrix} {}^{\text{H-1}}\gamma\,{}^{\text{H-1}}u^{\text{k+}} + {}^{\text{H-1}}a_x^{\text{k+1}}[{}^{\text{H-1}}a_0^{\text{k}} - {}^{\text{H-1}}a_0^{\text{k+1}}] \\ {}^{\text{H-1}}\gamma\,{}^{\text{H-1}}v^{\text{k+}} + {}^{\text{H-1}}a_y^{\text{k+1}}[{}^{\text{H-1}}a_0^{\text{k}} - {}^{\text{H-1}}a_0^{\text{k+1}}] \end{pmatrix} \tag{15}$$

Given the small size of the linear system, it is advantageous to use the adjoint representation of the matrix inverse. Thus, solving the linear system yields

$${}^{\text{H-1}}\Phi^{\text{k+1}} = \frac{1}{\det(M)}\text{adj}(M)q \tag{16}$$

Notice that if ${}^{\text{H-1}}\gamma > 0$, then $\det(M) = {}^{\text{H-1}}\gamma\left({}^{\text{H-1}}\gamma + \|{}^{\text{H-1}}a_1^{\text{k+1}}\|\right)$ is non zero and hence, solution to Equation (16) is well conditioned for all pixels in the image. To diffuse the new flow estimates to textureless regions of the image, we smooth the optical flow using an average filter of size 5×5 pixels after the update stage is completed. The number of smooth iterations applied to the updated flow is a parameter of the algorithm.

For lower levels of the pyramid $h = H-2, \ldots, 0$, the update stage aims to correct state prediction ${}^{\text{h}}\Delta\Phi^{\text{k+}}$ using new brightness information. Recall from Section II that ${}^{\text{h}}\Delta\Phi^{\text{k+1}}$ represents an increment to the coarse optical flow ${}^{\text{h+1}}\Phi^{\text{k+1}}$, provided higher resolution image data at this level. Also, recall that ${}^{\text{h}}a_0^{\text{k}}$ was propagated using ${}^{\text{h}}\Phi^{\text{k}}$ (Equation (10)) and a prediction ${}^{\text{h}}a_0^{\text{k+}}$ exists for time $k+1$. This prediction is expected to be similar to ${}^{\text{h}}a_0^{\text{k+1}}$ computed from image data, and, under the brightness constancy principle, any difference between both parameters is due to some flow, corresponding to state ${}^{\text{h}}\Delta\Phi^{\text{k+1}}$.

The update cost function for ${}^{\text{h}}\Delta\Phi^{\text{k+1}}$ is defined as

$$\begin{aligned} {}^{\text{h}}\varepsilon_U = &\left\|{}^{\text{h}}a_1^{\text{k+1}} \cdot {}^{\text{h}}\Delta\Phi^{\text{k+1}} + {}^{\text{h}}a_0^{\text{k+1}} - {}^{\text{h}}a_0^{\text{k+}}\right\|^2 \\ &+ {}^{\text{h}}\gamma\left\|{}^{\text{h}}\Delta\Phi^{\text{k+1}} - {}^{\text{h}}\Delta\Phi^{\text{k+}}\right\|^2 \end{aligned} \tag{17}$$

Similar to the update at top level, Equation (17) consists of two terms: a data term to relate new brightness parameters with the predicted ${}^{\text{h}}a_0^{\text{k+}}$ to generate a new ${}^{\text{h}}\Delta\Phi^{\text{k+1}}$ estimate, and a temporal regularization term to relate the prediction ${}^{\text{h}}\Delta\Phi^{\text{k+}}$ with the new estimation. Solution to Equation (17) follows the same approach used to solve Equation (12).

III. Numerical propagation

For solving the different PDEs that make the filter propagation stage, we desire a numerical scheme that is robust to noise in the initial conditions and is computationally efficient for our real-time applications. Also, since the propagation only takes place between two consecutive image frames, long term accuracy of the numeric method is not a strict requirement. Numerical methods found in the literature [21], [22], [23] exploit different properties of the problem for designing a suitable numerical scheme. In this section, we propose a modification to the upwind finite difference method for solving the propagation equations.

Let u_{ij}^{n}, v_{ij}^{n} be a discretization of the x and y optical flow components indexed by pixel coordinates (i,j). Here, we omit the pyramid level index h in the different variables, as the numerical scheme works the same for all levels.

To guarantee stability, the numerical method needs to run for several time iterations to reach the solution at final time. These iterations occur between two image frames, and should not be confused with the camera frame rate. Parameter N denotes the number of iterations of the scheme and $n = 0, \ldots, N-1$ is the iteration index. Each iteration covers an equal time step $\Delta t = {}^1\!/_N$.

The discrete counterpart of the optical flow transport in Equation (6) is written as

$$\frac{u_{ij}^{\text{n+1}} - u_{ij}^{\text{n}}}{\Delta t} + u_{ij}^{\text{n}}\delta_x u_{ij}^{\text{n}} + v_{ij}^{\text{n}}\delta_y u_{ij}^{\text{n}} = 0 \tag{18}$$

$$\frac{v_{ij}^{\text{n+1}} - v_{ij}^{\text{n}}}{\Delta t} + u_{ij}^{\text{n}}\delta_x v_{ij}^{\text{n}} + v_{ij}^{\text{n}}\delta_y v_{ij}^{\text{n}} = 0 \tag{19}$$

These equations are used for both the propagation of flow at top and lower levels of the pyramid, Equations (8) and (11), respectively.

Similarly, the discrete version of the scalar transport in Equation (7) is written as

$$\frac{c_{ij}^{\text{n+1}} - c_{ij}^{\text{n}}}{\Delta t} + u_{ij}^{\text{n}}\delta_x c_{ij}^{\text{n}} + v_{ij}^{\text{n}}\delta_y c_{ij}^{\text{n}} = 0 \tag{20}$$

This equation is used for propagating the state variables at lower levels of the filter (Equations (9) and (10)). One replica of Equation (20) is created for each scalar field $\{{}^{\text{h}}\Delta u, {}^{\text{h}}\Delta v, {}^{\text{h}}a_0\}$.

Operators δ_x and δ_y define differences in x and y directions, respectively. Such differences can be computed in any of the following ways

	$\delta_x u_{ij}$	$\delta_y u_{ij}$
backward	$\delta_{x\text{-}} = u_i - u_{i-1}$	$\delta_{y\text{-}} = u_j - u_{j-1}$
central	$\delta_{x0} = u_{i+1} - u_{i-1}$	$\delta_{y0} = u_{j+1} - u_{j-1}$
forward	$\delta_{x+} = u_{i+1} - u_i$	$\delta_{y+} = u_{j+1} - u_j$

For upwind numerical schemes, the choice of difference operator is conditioned on the velocity sign. This guarantees that the causality in the transport process is respected, *i.e.*, that only information in the upstream direction of flow is used in the calculations. If the flow is positive (moving to the right), then backward difference is used to consider information on the left side. Wrong choice of difference operator leads to unstable numerical schemes [22].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2532928, IEEE Robotics and Automation Letters

A. Linearization of flow transport

Direct application of upwind finite difference methods to non-linear Equations (18) and (19) do not generate the expected propagation of the flow field [23, p. 140]. Instead of using a non-linear PDE solver, which is computationally more expensive, we linearize the transport equations by making an approximation of the velocity at which each flow vector moves in the grid. We name this approximation as *dominant flow*. It is calculated as

$$\hat{u}_{ij}^{\mathrm{n}} = \begin{cases} \sigma_{x+} u_{ij}^{\mathrm{n}} & \text{if } \delta_{x0}|u_{ij}^{\mathrm{n}}| > 0 \\ \sigma_{x-} u_{ij}^{\mathrm{n}} & \text{if } \delta_{x0}|u_{ij}^{\mathrm{n}}| \leq 0 \end{cases} \tag{21}$$

$$\hat{v}_{ij}^{\mathrm{n}} = \begin{cases} \sigma_{y+} v_{ij}^{\mathrm{n}} & \text{if } \delta_{y0}|v_{ij}^{\mathrm{n}}| > 0 \\ \sigma_{y-} v_{ij}^{\mathrm{n}} & \text{if } \delta_{y0}|v_{ij}^{\mathrm{n}}| \leq 0 \end{cases} \tag{22}$$

where σ define shift operators in the image grid

	u_{ij}	v_{ij}
backward	$\sigma_{x-} u_i = u_{i-1}$	$\sigma_{y-} v_j = v_{j-1}$
forward	$\sigma_{x+} u_i = u_{i+1}$	$\sigma_{y+} v_j = v_{j+1}$

The *dominant flow* is the largest flow in x and y in the pixel's 4-neighborhood. Discontinuities in the optical flow field will be propagated in the direction of the dominant flow. As flow discontinuities typically occur in places of the scene where foreground objects occlude parts of the background, the propagation of such discontinuities by the foreground velocity, *i.e.*, the dominant flow, is a reasonable choice that allows use of a fast numerical scheme designed for linear PDE systems.

B. Iterative numerical scheme

Application of the upwind numerical scheme to the propagation equations is as follows. First, dominant flow $\hat{u}_{ij}^{\mathrm{n}}$ is computed for all pixels in the grid using Equation (21). Then, the optical flow and scalar fields are propagated in the x direction as

$$u_{ij}^{\mathrm{n}+\frac{1}{2}} = \begin{cases} u_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x-} u_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} \geq 0 \\ u_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x+} u_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} < 0 \end{cases} \tag{23}$$

$$v_{ij}^{\mathrm{n}+\frac{1}{2}} = \begin{cases} v_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x-} v_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} \geq 0 \\ v_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x+} v_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} < 0 \end{cases} \tag{24}$$

$$c_{ij}^{\mathrm{n}+\frac{1}{2}} = \begin{cases} c_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x-} c_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} \geq 0 \\ c_{ij}^{\mathrm{n}} - R\hat{u}_{ij}^{\mathrm{n}} \delta_{x+} c_{ij}^{\mathrm{n}} & \text{if } \hat{u}_{ij}^{\mathrm{n}} < 0 \end{cases} \tag{25}$$

After this, dominant flow $\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}}$ is computed using equation (22), and propagation takes place in the y direction

$$u_{ij}^{\mathrm{n}+1} = \begin{cases} u_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y-} u_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \geq 0 \\ u_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y+} u_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} < 0 \end{cases} \tag{26}$$

$$v_{ij}^{\mathrm{n}+1} = \begin{cases} v_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y-} v_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \geq 0 \\ v_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y+} v_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} < 0 \end{cases} \tag{27}$$

$$c_{ij}^{\mathrm{n}+1} = \begin{cases} c_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y-} c_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \geq 0 \\ c_{ij}^{\mathrm{n}+\frac{1}{2}} - R\hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} \delta_{y+} c_{ij}^{\mathrm{n}+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{\mathrm{n}+\frac{1}{2}} < 0 \end{cases} \tag{28}$$

The constant term

$$R := \frac{\Delta t}{\Delta x} = \frac{1}{N} \tag{29}$$

is the ratio between time and space step parameters. Grid spacing $\Delta x = 1$ is the same for all pyramid levels. Scaling of flow by pixel size is performed during the upsampling operation in Equation (2).

Stability of the scheme is given by the Courant-Friedrichs-Lewy condition [22]

$$R \max_{n,i,j} \left\{ |\hat{u}_{ij}^{\mathrm{n}}|, |\hat{v}_{ij}^{\mathrm{n}}| \right\} \leq 1 \tag{30}$$

Given the number of iterations N as a parameter of the algorithm, it can be seen from the stability condition that the components of the optical flow field must satisfy inequality $|u_{ij}^{\mathrm{n}}|, |v_{ij}^{\mathrm{n}}| \leq N$ for all pixels and time iterations. Considering that the pyramid is constructed with a downsampling factor of 2, if ${}^{\mathrm{h}}N$ denotes the number of iterations at level h, then the number of iterations required at level $h+1$ to satisfy the stability condition can be computed as ${}^{\mathrm{h+1}}N = {}^{\mathrm{h}}N/2$.

The numerical solution to the propagation equations posses several properties that can be exploited in a parallel or streamed implementation. In particular, calculations for each pixel are independent of each other, allowing for implementation on GPU hardware. Additionally, since each pixel only requires its 4-neighborhood to complete one iteration of the numerical scheme, it is possible to build a pipeline of propagation blocks, where each block only needs to buffer 3 rows of image and flow data. Since the number of iterations can be fixed, it is possible to create a digital circuit that implements the propagation pipeline, maximizing pixel throughput while minimizing external memory access.

IV. GPU IMPLEMENTATION

Computations within different stages of the filter are independent for each pixel, and require a fixed and local support around each pixel. Moreover, memory access pattern is deterministic and independent of actual memory content. These properties makes the algorithm a good candidate for implementation on parallel platforms. Currently, the proposed flow filter algorithm is implemented on GPU platform using Nvidia CUDA. For each pyramid level, the different filter stages are pipelined. At each stage, data is read from previous stage output buffer and computation results are written into a separate memory buffer, avoiding read-write memory racing. Images are stored in pitched memory arrays. Read operations are performed through the GPU texture units to exploit cache buffering of neighboring pixels, eliminating the need of manually buffering data using shared memory arrays. Coalesced write operations write the results directly to pitched global memory buffers.

V. EXPERIMENTAL EVALUATION

A. Setup

We ran all the algorithms for the experimental validation on a Desktop computer with an Intel i7-4790K processor and a Nvidia GTX 780 with 2304 CUDA cores and 288 GB/s of memory bandwidth. In the evaluation, we included other GPU optical flow methods suitable for real-time applications: Pyramidal Lucas-Kanade (Pyr-LK) [18], Farneback [19] and Brox [20]. These algorithms were parametrized such that their running frequency was close to the frame rate of the camera.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2532928, IEEE Robotics and Automation Letters

Other GPU methods such as eFOLKI [10] and Flowlib [24] were not considered in the evaluation, as either their code were not available or were outdated with respect to current version of CUDA. Computing time is recorded from the moment each algorithm start computations. Data transfer between CPU and GPU memory spaces is not included in the runtime measurements, as such transfers can be scheduled in parallel to the compute threads.

B. Ground truth error analysis

To analyze the error performance of the proposed algorithm, we require image datasets of cameras operating at high frame rates, reproducing the operating conditions on which the optical flow filter is expected to be deployed. Conventional evaluation datasets [4], [5], [6] do not provide either enough images, nor the required frame rate.

We created a synthetic image sequence in Blender[1]. This sequence, named Bunny, simulates a high speed camera working at VGA resolution (640×480 pixels) at 300 Hz. The camera moves at a constant linear velocity of $0.5^{\mathrm{m}}/_{\mathrm{s}}$ in x direction for 1 second, thus generating a total of 300 images. With this sequence, we studied the converge rate of the algorithm in presence of constant camera velocity. The optical flow filter is configured with two pyramid levels, $[4, 2]$ propagation iterations, $[2, 4]$ smooth iterations and gains $\gamma = [50, 5]$ at bottom and top levels respectively. This parametrization enables the filtering algorithm, in particular the propagation blocks, to handle up to $4^{\mathrm{pix}}/_{\mathrm{frame}}$ of optical flow between consecutive frames.

Figure 2 shows the computed flow fields for a selection of image frames using Middlebury colorwheel encoding [4], while Figure 3 (top) shows the recorded runtime in milliseconds for each frame. Figure 3 (second row) plots the average Endpoint Error (avg-EPE) for each algorithm in the evaluation. For the first 20 frames, the estimated optical flow goes from a zero initial condition to a dense flow estimation in the correct direction of camera motion. The average endpoint error converges to a constant value lower than the evaluated algorithms.

We performed a second set of experiments on this sequence to analyze the trade off between runtime and error performance in the comparison algorithms. For this, we chose images 150-151 as reference frames. The parameters of our algorithm are unchanged and equal to the values mentioned above. The estimated optical flow by our method at frame 150 and the average runtime are used for reference in this analysis.

To measure the trade off between error and runtime performance, we varied the number of iterations each method used to compute optical flow. Figure 3 (third row) displays avg-EPE as function of number of iterations. Avg-EPE reduces significantly for Brox variational method, giving comparable error performance to our method after 25 iterations. Figure 3 (fourth row) plots the effect of number of iterations on the achieved frame rate of each algorithm. In this case, only Pyr-LK algorithm is capable of reaching comparable frame rates as our method. Finally, fifth row of Figure 3 compares

[1]https://www.blender.org/

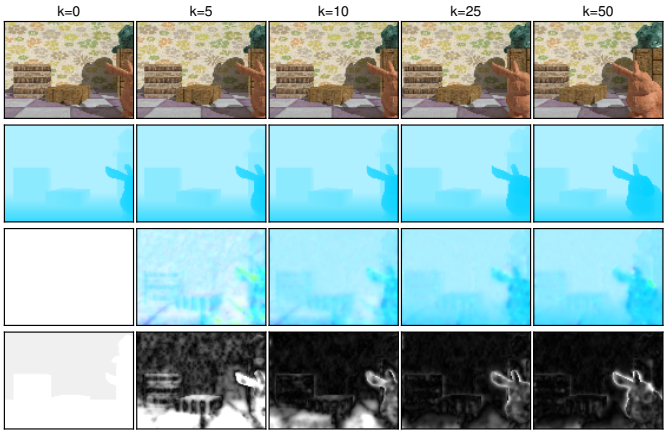


Fig. 2: Results on Bunny sequence. From top to bottom: input image, ground truth flow, estimated flow and EPE (0 to 1 pixel).

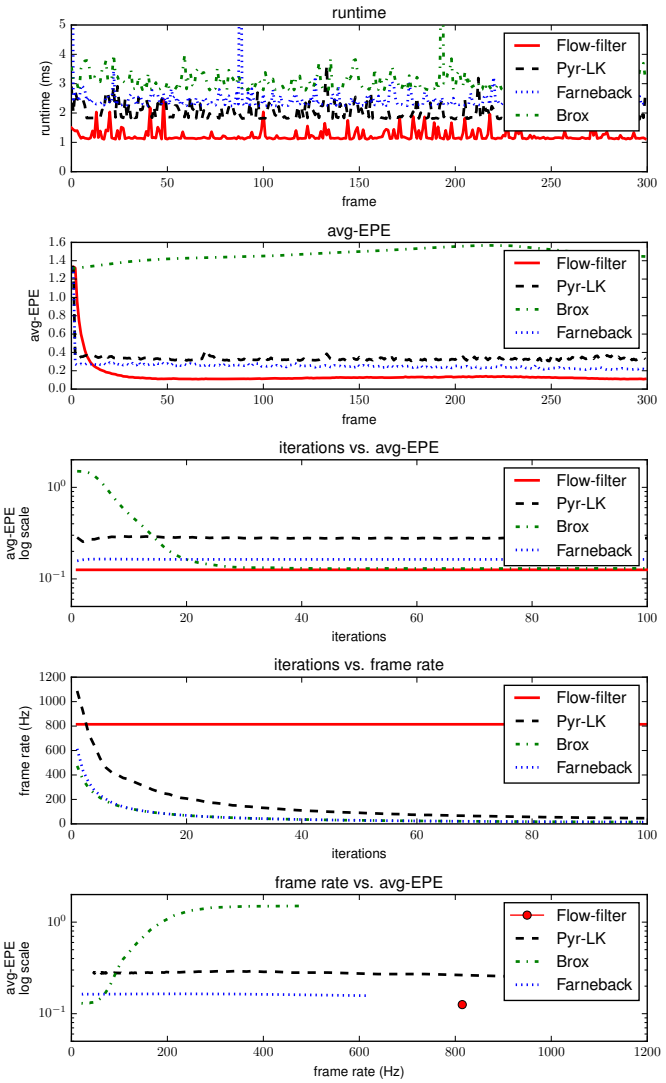


Fig. 3: Ground truth error and runtime performance for the Bunny sequence.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2532928, IEEE Robotics and Automation Letters

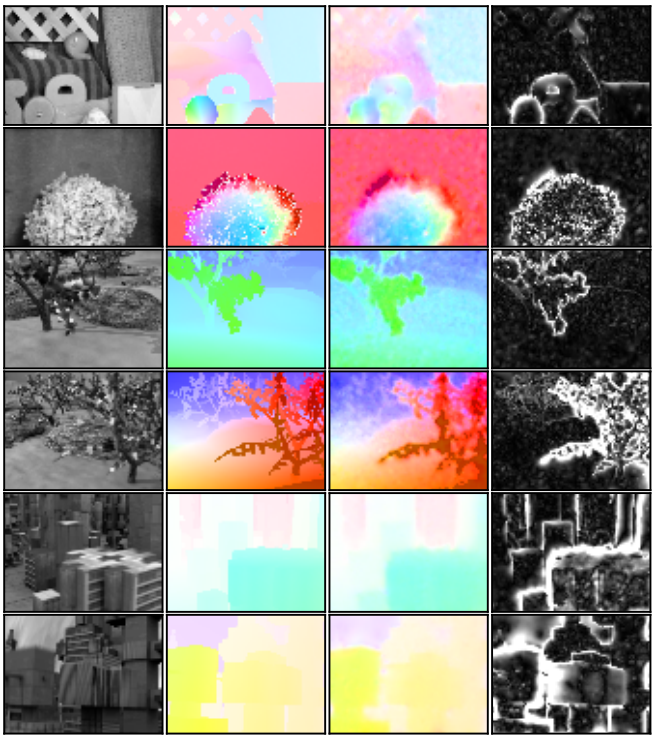


Fig. 4: Results on Middlebury test sequences. From left to right: input image, ground truth, estimated optical flow and EPE error (0 to 1 pixel).

the obtained avg-EPE and the achieved frame rate of each algorithm. Notice that our method achieved an average frame rate of 814 Hz with the same error performance than Brox method working approximately at 35 Hz. The achieved frame rate by our method is equivalent to pixel throughtputs in order of 250 Mpix/s.

C. Evaluation on Middlebury dataset

We performed an extra error analysis using the Middlebury training dataset [4]. We created additional images for each sequence using the provided ground truth optical flow. For this, we used the propagation equations of our filter as interpolation mechanism. Initial conditions for the propagation are set to the ground truth optical flow and reference image. We ran the propagation both in backwards and forward time direction for 50 iterations with time step $\Delta t = {}^1/_{50}$, thus generating 100 images total. These images are sufficient for our algorithm to reach stable state, enabling error comparison against provided ground truth. Figure 4 shows the estimated optical flow using the interpolated images. The filter is configured with two pyramid levels, $[4, 2]$ propagation iterations, $[2, 4]$ smooth iterations and gains $\gamma = [50, 5]$ at bottom and top levels respectively.

Average endpoint error, runtime and throughput for the evaluated algorithms are reported in Table I. For Pyr-LK, Farneback and Brox methods, optical flow is computed using the original images provided in the dataset. Average endpoint error is comparable to the evaluated algorithms. For the Urban2 and Urban3 case, our method yields better results as the interpolated images used to compute flow posses small displacements between frames compared to the original sequences. In terms of runtime performance, our method achieves 2x performance gain compared to Pyr-LK and 25x gain with respect to Brox method, achieving frame rates in the order of 870 Hz and pixel throughput of 244 Mpix/s.

	Flow-filter	Pyr-LK	Farneback	Brox
avg. EPE (pix.)				
RubberWhale	0.316	0.413	0.288	0.185
Hydrangea	0.584	0.932	0.606	0.499
Grove2	0.345	0.467	0.585	0.278
Grove3	0.919	1.562	1.245	1.187
Urban2	0.886	7.623	5.612	6.085
Urban3	0.999	7.038	6.336	4.281
avg. runtime (ms)	**1.140**	**2.657**	**4.128**	**52.419**
throughput (Mpix/s)	**244.863**	**105.210**	**67.949**	**5.328**

TABLE I: Average endpoint error, runtime and throughput for the Middlebury test dataset.

D. Evaluation on high speed road video

We tested the optical flow filter on high speed video data recorded with a vehicle driving around ANU campus. We used a Basler acA2000-165um monochrome USB 3.0 camera capable of recording 1016×544 video at 300 Hz. The camera was placed in the front bumper and recorded approximately 60000 images at 300 Hz, or about 3.3 minutes of travel. Figure 5 shows a qualitatively comparison between our filtering algorithm and Pyramidal-LK method. Both algorithms run with 2 pyramid levels and are parametrized such that runtime is close to the camera frame rate. The average frame rate for our method was 507 Hz, while Pyr-LK achieved 279 Hz

In general, our flow-filter method is capable of capturing the optical flow of elements in the scene that are relevant to driving applications, such as road, transit signs, pedestrians and other vehicles. Thanks to the temporal regularization term in the update stage, and the spatial diffusion of flow information from highly textured regions, the estimated optical flow is temporally and spatially consistent. In contrast, the results computed using Lucas-Kanade method show high levels of noise in regions of the image with poor texture information. Since this method is a 2-frames based approach, temporal consistency is not guaranteed.

In terms of quality of the resulting optical flow, our method gives flow fields that might be considered as blurred according to modern flow methods coming from Computer Vision. On the other hand, the real time capability of our method, as well as its temporal and spatial consistency, are enabling features for creating a robust low level optical flow sensor to be used for high level control tasks in robotic applications.

VI. Conclusions and Future Work

We presented a novel optical flow algorithm designed for high speed real time applications. The algorithm consists of a pyramid of filtering loops that incrementally creates a dense estimation of optical flow over time. The computational structure of the algorithm makes it suitable for implementation on GPU and FPGA systems. Currently, a GPU implementation of the algorithm is available. We compared our method

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2532928, IEEE Robotics and Automation Letters

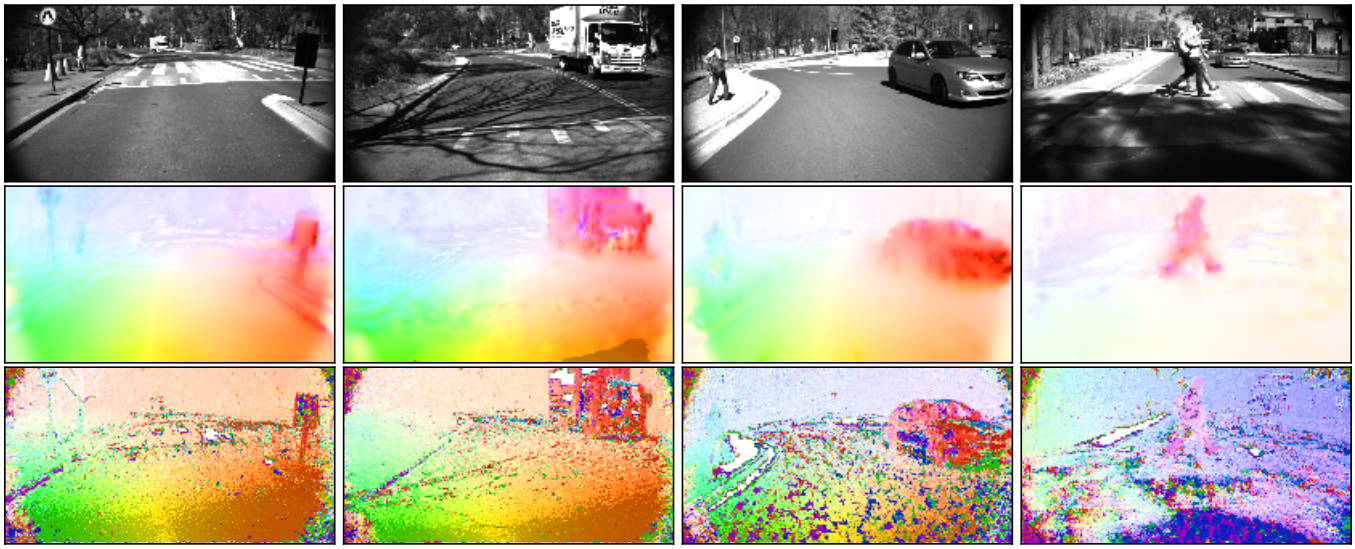


Fig. 5: Images captured with a 300 Hz camera and the estimated optical flow. From top to bottom: input image, flow-filter estimated optical flow (507 Hz avg frame rate), pyramidal Lucas-Kanade optical flow (279 Hz avg frame rate).

with other available GPU optical flow methods in terms of error and runtime performance on different high speed video sequences. Evaluation of our method on real life high speed video shows possible applications on driving systems. Future lines of work include the improvement our method in terms of error performance. This can be achieved by designing more robust optical flow update methods as well as better numerical schemes for state propagation. Additionally, we will investigate the deployment of the method on embedded hardware such as mobile SoC with embedded GPU and low-power FPGA hardware.

REFERENCES

[1] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys, "An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 1736–1741.

[2] B. Herisse, T. Hamel, R. Mahony, and F.-X. Russotto, "Landing a vtol unmanned aerial vehicle on a moving platform using optical flow," *IEEE. Trans. Robotics*, vol. 28, no. 1, pp. 77–89, 2012.

[3] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "Mav navigation through indoor corridors using optical flow," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 3361–3368.

[4] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *Int. J. Comput. Vision*, vol. 92, pp. 1–31, 2011.

[5] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *Proc. European Conf. Comput. Vision*, 2012, pp. 611–625.

[6] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *Int. J. Robotics Reseach*, Aug. 2013.

[7] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. 7th Int. Joint Conf. Artif. Intell.*, Vancouver, BC, Canada, 1981, pp. 674–679.

[8] M. V. Srinivasan, "An image interpolation technique for the computation of optical flow and egomotion," *Biological Cybernetics*, vol. 71, pp. 401–415, 1994.

[9] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185 – 203, 1981.

[10] A. Plyer, G. Le Besnerais, and F. Champagnat, "Massively parallel lucas kanade optical flow for real-time video processing applications," *J. Real-Time Img. Processing*, pp. 1–18, 2014.

[11] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. Van Hulle, "A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features," *IEEE Trans. Computers*, vol. 61, no. 7, pp. 999–1012, July 2012.

[12] M. Tao, J. Bai, P. Kohli, and S. Paris, "Simpleflow: A non-iterative, sublinear optical flow algorithm," *Comput. Graphics Forum*, vol. 31, no. 2pt1, pp. 345–353, 2012.

[13] D. J. Heeger, "Optical flow using spatiotemporal filters," *Int. J. Comput. Vision*, vol. 1, pp. 279–302, 1988.

[14] D. Fleet and K. Langley, "Recursive filters for optical flow," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 1, pp. 61 –67, jan 1995.

[15] J. D. Adarve, D. Austin, and R. Mahony, "A filtering approach for the computation of real-time dense optical-flow for robotic applications," in *Proc. Australasian Conf. Robot. Autom.*, 2014.

[16] M. J. Black, "Robust incremental optical flow," Ph.D. dissertation, Yale university, 1992.

[17] J. Barron, "System and experiment. performance of optical flow techniques," *Int. J. Comput. Vision*, vol. 12:1, pp. 43–77, 1994.

[18] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm," Intel Corporation, Tech. Rep., 2001.

[19] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, 2003, pp. 363–370.

[20] T. Brox, A. Brhun, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *Proc. European Conf. Comput. Vision*, vol. 4, 2004, pp. 25–36.

[21] R. J. LeVeque, *Finite volume methods for hyperbolic problems*. Cambridge university press, 2002, vol. 31.

[22] J. W. Thomas, *Numerical partial differential equations: finite difference methods*. Springer, 1995, vol. 22.

[23] ——, *Numerical partial differential equations: conservation laws and elliptic equations*. Springer Berlin, 1999, vol. 3.

[24] M. Werlberger, "Convex approaches for high performance video processing," Ph.D. dissertation, Institute for Computer Graphics and Vision, Graz University of Technology, 2012.