



HAL
open science

Depth from motion algorithm and hardware architecture for smart cameras

Abiel Aguilar-González, Miguel Arias-Estrada, François Berry

► **To cite this version:**

Abiel Aguilar-González, Miguel Arias-Estrada, François Berry. Depth from motion algorithm and hardware architecture for smart cameras. Sensors, 2018. hal-01964830

HAL Id: hal-01964830

<https://hal.science/hal-01964830v1>


Submitted on 23 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Article

Depth from motion algorithm and hardware architecture for smart cameras

Abiel Aguilar-González ^{1,2*} , Miguel Arias-Estrada ¹ and François Berry ²

¹ Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Mexico

² Institut Pascal, Université Clermont Auvergne (UCA), Clermont-Ferrand, France

* Correspondence: abiel@inaoep.mx, abiel.aguilar_gonzalez@etu.uca.fr

Academic Editor: name

Version November 26, 2018 submitted to Sensors

Abstract: Applications such as autonomous navigation, robot vision, autonomous flying, etc., require depth map information of the scene. Depth can be estimated by using a single moving camera (depth from motion). However, traditional depth from motion algorithms have low processing speed and high hardware requirements that limits the embedded capabilities. In this work, we propose a hardware architecture for depth from motion that consists of a flow/depth transformation and a new optical flow algorithm. Our optical flow formulation consists in an extension of the stereo matching problem. A pixel-parallel/window-parallel approach where a correlation function based in the Sum of Absolute Differences computes the optical flow is proposed. Further, in order to improve the Sum of Absolute Differences performance, the curl of the intensity gradient as preprocessing step is proposed. Experimental results demonstrated that it is possible to reach higher accuracy (90% of accuracy) compared with previous FPGA-based optical flow algorithms. For the depth estimation, our algorithm delivers dense maps with motion and depth information on all the image pixels, with a processing speed up to 128 times faster than previous works and making it possible to achieve high performance in the context of embedded applications.

Keywords: Depth estimation; monocular systems; optical flow; smart cameras; FPGA

1. Introduction

Smart cameras are machine vision systems which, in addition to image capture circuitry, are capable of extracting application-specific information from the captured images. For example, for video surveillance, image processing algorithms implemented inside the camera fabric could detect and track pedestrians [1], but for a robotic application, computer vision algorithms could estimate the system egomotion [2]. In recent years, advances in embedded vision systems such as progress in microprocessor power and FPGA technology allowed the creation of compact smart cameras with increased performance for real world applications [3–6]. As result, in current embedded applications, image processing algorithms inside the smart cameras fabric deliver an efficient on-board solution for: motion detection [7], object detection/tracking [8,9], inspection and surveillance [10], human behavior recognition [11], etc. Another algorithm that could be highly used by smart cameras are computer vision algorithms since they are the basis of several applications (automatic inspection, controlling processes, detecting events, modeling objects or environments, navigation and so on). Unfortunately, mathematical formulation of computer vision algorithms is not compliant with the hardware technologies (FPGA/CUDA) often used in smart cameras. In this work, we are interested in depth estimation from monocular sequences in the context of a smart camera because depth is the basis to obtain useful scene abstractions, for example: 3D reconstructions of the world and the camera egomotion.

34 1.1. *Depth estimation from monocular sequences*

35 In several applications, like autonomous navigation [12], robot vision and surveillance [1],
36 autonomous flying [13], etc., there is a need for determining the depth map of the scene. Depth
37 can be estimated by using stereo cameras [14], by changing focal length [15], or by employing a single
38 moving camera [16]. In this work we are interested in depth estimation from monocular sequences
39 by using a single moving camera (depth from motion). This choice is motivated because monocular
40 systems have higher efficiency compared with other approaches, simpler and more accurate than
41 defocus techniques and, cheaper/smaller compared with stereo-based techniques. In monocular
42 systems, depth information can be estimated based on two or multiple frames of a video sequence. For
43 two frames, image information may not provide sufficient information for accurate depth estimation.
44 The use of multiple frames improves the accuracy, reduces the influence of noise and allows the
45 extraction of additional information which cannot be recovered from just two frames, but the system
46 complexity and computational cost is increased. In this work, we use information from two consecutive
47 frames of the monocular sequence since our algorithm is focused for smart cameras and in this context
48 hardware resources are limited.

49 1.2. *Motivation and scope*

50 In the last decade several works have demonstrated that depth information is highly useful for
51 embedded robotic applications [1,12,13]. Unfortunately, depth information estimation is a relatively
52 complex task. In recent years, the most popular solution is the use of active vision to estimate depth
53 information from the scene [17–21], i.e., LIDAR sensors or RGBD cameras that can deliver accurate
54 depth maps in real time, however they increase the systems size and cost. In this work, we propose
55 a new algorithm and an FPGA hardware architecture for depth estimation. First, a new optical
56 flow algorithm estimates the motion (flow) at each point in the input image. Then, a flow/depth
57 transformation computes the depth in the scene. For the optical flow algorithm: an extension of the
58 stereo matching problem is proposed. A pixel-parallel/window-parallel approach where a Sum of
59 Absolute Differences computes the optical flow is implemented. Further, in order to improve the Sum
60 of Absolute Differences performance, we propose the curl of the intensity gradient as preprocessing
61 step. For the depth estimation proposes: we introduce a flow/depth transformation inspired in the
62 epipolar geometry.

63 2. **Related work**

64 In previous works, depth estimation is often estimated by using a single moving camera. This
65 approach is called depth from motion and consists in computing the depth from the pixel velocities
66 inside the scene (optical flow). i.e., optical flow is the basis for depth from motion.

67 2.1. *FPGA architectures for optical flow*

68 In [22], a hardware implementation of a high complexity algorithm to estimate the optical
69 flow from image sequences in real time is presented. In order to fulfil with the architectural
70 limitations, the original gradient-based optical flow was modified (using a smoothness constraint for
71 decreasing iterations). The developed architecture can estimate the optical flow in real time and can be
72 constructed with FPGA or ASIC devices. However, due to the mathematical limitations of the CPU
73 formulation (complex/iterative operations), speed processing is low, compared with other FPGA-based
74 architectures for real-time image processing [23,24]. In [25], a pipelined optical-flow processing system
75 that works as a virtual motion sensor is described. The proposed approach consists of several spatial
76 and temporal filters (Gaussian and gradient spatial filters and IIR temporal filter) implemented in
77 cascade. The proposed algorithm was implemented in an FPGA device, enabling the easy change
78 of the configuration parameters to adapt the sensor to different speeds, light conditions and other
79 environmental factors. This makes possible the implementation of an FPGA-based smart camera for

80 optical flow. In general, the proposed architecture reaches a reasonable hardware resources usage
 81 but accuracy and processing speed is low (lower than 7 fps for 640×480 image resolution). In [26], a
 82 tensor-based optical flow algorithm is presented. This algorithm was developed and implemented
 83 using FPGA technology. Experimental results demonstrated high accuracy compared with previously
 84 FPGA-based algorithms for optical flow. In addition, the proposed design can process 640×480 images
 85 at 64 fps with a relatively low resource requirement, making it easier to fit into small embedded
 86 systems. In [27], a highly parallel architecture for motion estimation is presented. The developed
 87 FPGA-architecture implements the Lucas and Kanade algorithm [28] with the multi-scale extension for
 88 the computation of large motion estimations in an FPGA. Although the proposed architecture reaches
 89 a low hardware requirement with a high processing speed, the use of huge external memory capacity
 90 is needed. Further, in order to fulfil with the hardware limitations, the accuracy is low (near 11% more
 91 error compared with the original CPU version of the Lukas and Kanade algorithm). Finally, in [29],
 92 an FPGA-based platform with the capability of calculating real-time optical flow at 127 frames per
 93 second for a 376×240 pixel resolution is presented. Radial undistortion, image rectification, disparity
 94 estimation and optical flow calculation tasks are performed on a single FPGA without the need of
 95 external memory. So, the platform is perfectly suited for mobile robots or embedded applications.
 96 Unfortunately, accuracy is low (qualitatively lower accuracy than CPU based approaches).

97 2.2. Optical flow methods based in learning techniques

98 There are some recent works that addresses the optical flow problem via learning techniques [30].
 99 In 2015 [31] proposed the use of convolutional neuronal networks (CNNs) as an alternative framework
 100 to solve the optical flow estimation problem. Two different architectures were proposed and compared:
 101 a generic architecture and another one including a layer that correlates feature vectors at different
 102 image locations. Experimental results demonstrated a competitive accuracy at frame rates of 5 to
 103 10 fps. On the other hand, in 2017 [32] developed a stacked architecture that includes warping of
 104 the search image with intermediate optical flow. Further, in order to achieve high accuracy on small
 105 displacements, authors introduced a sub-network specializing on small motions. Experimental results
 106 demonstrated that it is possible to reach more than 95% of accuracy, decreasing the estimation error by
 107 more than 50% compared with previous works.

108 3. The proposed algorithm

109 In Fig. 1 an overview of our algorithm is shown. First, given an imager as sensor, two consecutive
 110 frames ($f_t(x, y)$, $f_{t+1}(x, y)$) are stored in local memory. Then, an optical flow algorithm computes 2D
 111 pixel displacements between $f_t(x, y)$ and $f_{t+1}(x, y)$. A dynamic template based on the optical flow
 112 previously computed ($\Delta_{x,t-1}(x, y)$, $\Delta_{y,t-1}(x, y)$) computes the search region size for the current optical
 113 flow. Then, let the optical flow for the current frame be ($\Delta_x(x, y)$, $\Delta_y(x, y)$), the final step is depth
 114 estimation for all the pixels in the reference image $D(x, y)$. In the following subsections, details about
 115 the proposed algorithm are presented.

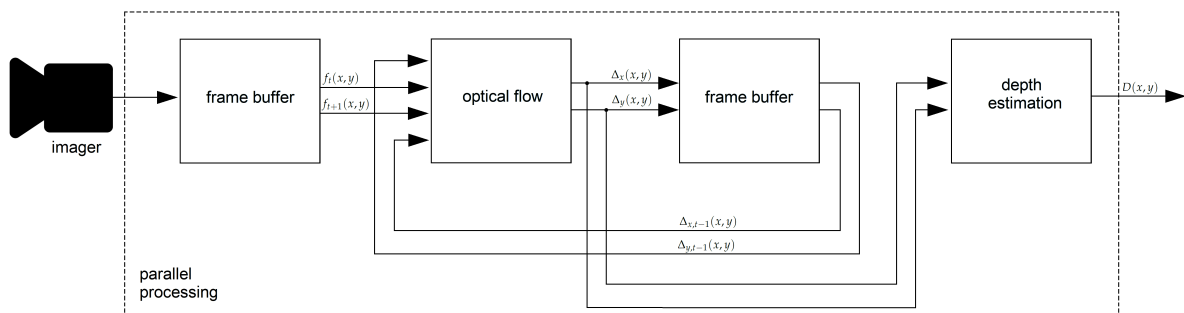


Figure 1. Block diagram of the proposed algorithm

116 3.1. Frame buffer

117 The first step in our mathematical formulation is image storage, considering that in most cases the
 118 imager provides data as a stream, some storage is required in order to have two consecutive frames
 119 available at the same time t . More information/details about the storage architecture are presented
 120 in **Section 4.1**. For mathematical formulation, we consider the first frame (frame at t time) as $f_t(x, y)$
 121 while the second frame (frame at $t + 1$ time) is $f_{t+1}(x, y)$.

122 3.2. Optical flow

123 In previous works, iterative algorithms, such as the Lucas Kanade [28] or the Horn–Schunck [33]
 124 algorithms have been used on order to compute optical flow across video sequences, then, given dense
 125 optical flow, geometric methods allow to compute the depth in the scene. However, these algorithms
 126 [28,33] have iterative operations that limit the performance for smart camera implementations. In
 127 order to avoid the iterative and convergence part of the traditional formulation we replace that with a
 128 correlation metric implemented inside a pixel-parallel/window-parallel formulation. In **Fig. 2** an
 129 overview of our optical flow algorithm is shown. Let $(f_t(x, y), f_{t+1}(x, y))$ be two consecutive frames
 130 from a video **sequence**, curl of the intensity gradient $\frac{df(x,y)}{dx}$ are computed, see **Eq. 1**, where ∇ is
 131 the **Del** operator. Let curl be a vector operator that describes the infinitesimal rotation, then, at
 132 every pixel the curl of that pixel is represented by a vector where attributes (length and direction)
 133 characterize the rotation at that point. In our case, we use only the norm of $\overline{\text{Curl}}(x, y)$, as shown in
 134 **Eq. 2** and, as illustrated in **Fig. 3**. This operation increases the robustness under image degradations
 135 (color/texture repetition, illumination changes, noise), therefore, simple similarity metrics [34] deliver
 136 accurate pixel tracking, simpler than previous tracking algorithms [28,33]. Given the curl images for
 137 two consecutive frames $(\overline{\text{Curl}}_t(x, y), \overline{\text{Curl}}_{t+1}(x, y))$, dense optical flow $(\Delta_x(x, y), \Delta_y(x, y))$, illustrated
 138 in **Fig. 4** in the reference image is computed as shown in **Fig. 5**. This process assumes that pixel
 139 displacements between frames is such as it exists an overlap on two successive "search regions". A
 140 search region is defined as a patch around a pixel to track. Considering that between f_t and f_{t+1} , the
 141 image degradation is low, any similarity-based metric have to provide good accuracy. In our case, this
 142 similarity is calculated by a SAD (Sum of Absolute Difference). This process is defined in **Eq. 3**; where
 143 r is the patch size (see **Fig. 5**). $(\overline{\text{Curl}}_t(x, y), \overline{\text{Curl}}_{t+1}(x, y))$ are curl images on two consecutive frames.
 144 x, y are the spatial coordinates of pixels in f_t and, a, b are the spatial coordinates within a search region
 145 constructed in f_{t+1} (see **Eq. 4** and **5**); where $\Delta'_{x(t-1)}, \Delta'_{y(t-1)}$ are a dynamic search template, computed
 146 as shown in **Section 3.3**. k is the search size and s is a sampling value defined by the user. Finally,
 147 optical flow **at the current time** $(\Delta_x(x, y), \Delta_y(x, y))$ is computed by **Eq. 6**.

148

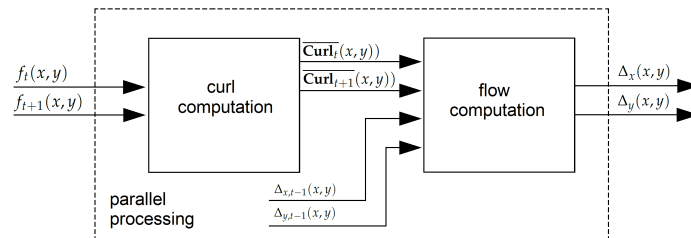


Figure 2. The optical flow step: first, curl images $(\overline{\text{Curl}}_t(x, y), \overline{\text{Curl}}_{t+1}(x, y))$ are computed. Then, given the curl images for two consecutive frames, pixels displacements $\Delta_x(x, y), \Delta_y(x, y)$ (optical flow for all pixels in the reference image) are computed using a dynamic template based on the optical flow previously computed $(\Delta_{x,t-1}(x, y), \Delta_{y,t-1}(x, y))$.

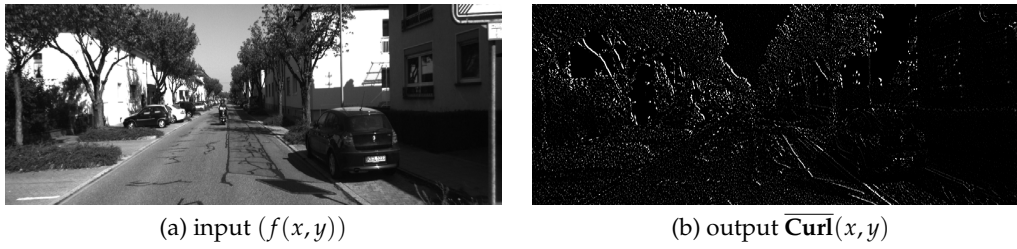


Figure 3. Curl computation example. Input image taken from the KITTI benchmark dataset [35]

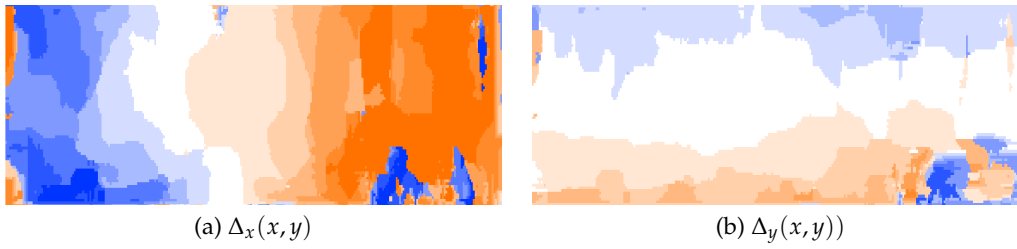


Figure 4. Optical flow example. Image codification as proposed in the Tsukuba benchmark dataset [36]

$$\mathbf{Curl}(x, y) = \nabla \times \frac{df(x, y)}{dx} = \frac{\partial}{\partial y} \frac{\partial f(x, y)}{\partial x} - \frac{\partial}{\partial x} \frac{\partial f(x, y)}{\partial y} \quad (1)$$

$$\overline{\mathbf{Curl}}(x, y) = \left| \frac{\partial}{\partial y} \frac{\partial f(x, y)}{\partial x} - \frac{\partial}{\partial x} \frac{\partial f(x, y)}{\partial y} \right| \quad (2)$$

149

where

$$\frac{\partial f(x, y)}{\partial x} = G_x(x, y) = f(x + 1, y) - f(x - 1, y)$$

$$\frac{\partial f(x, y)}{\partial y} = G_y(x, y) = f(x, y + 1) - f(x, y - 1)$$

$$\frac{\partial}{\partial y} \frac{\partial f(x, y)}{\partial x} = G_x(x, y + 1) - G_x(x, y - 1)$$

$$\frac{\partial}{\partial x} \frac{\partial f(x, y)}{\partial y} = G_y(x + 1, y) - G_y(x - 1, y)$$

150

$$\text{SAD}(a, b) = \sum_{u=-r}^{u=r} \sum_{v=-r}^{v=r} |\overline{\mathbf{Curl}}_t(x + u, y + v)| - |\overline{\mathbf{Curl}}_{t+1}(x + u + a, y + v + b)| \quad (3)$$

$$a = \Delta'_{x(t-1)}(x, y) - k : s : \Delta'_{x(t-1)}(x, y) + k \quad (4)$$

$$b = \Delta'_{y(t-1)}(x, y) - k : s : \Delta'_{y(t-1)}(x, y) + k \quad (5)$$

$$[\Delta_x(x, y), \Delta_y(x, y)] = \mathbf{arg\ min}_{(a, b)} \text{SAD}(a, b) \quad (6)$$

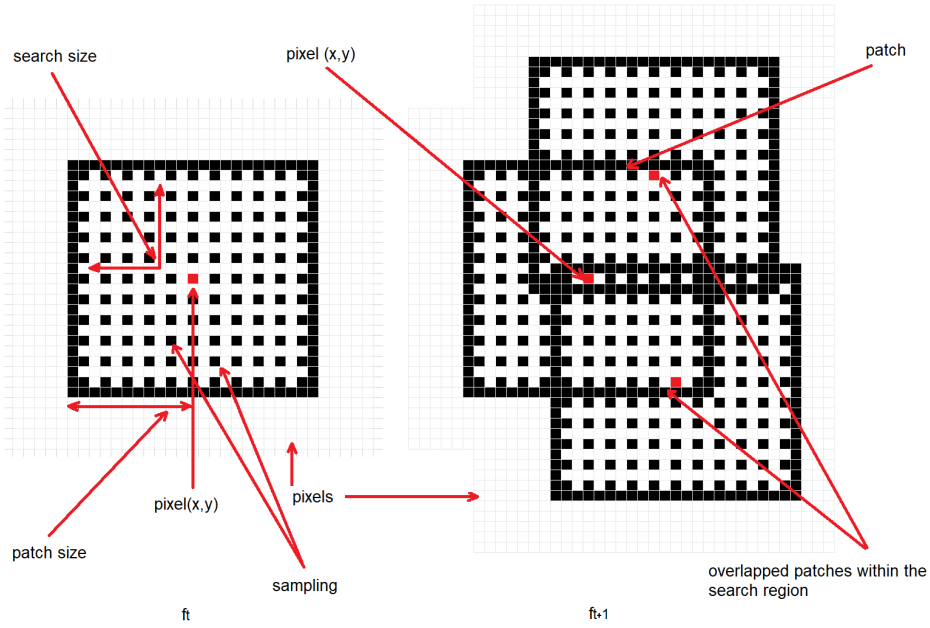


Figure 5. The proposed optical flow algorithm formulation: patch size = 10, search size = 10, sampling value = 2. For each pixel in the reference image f_t , n overlapped regions are constructed in f_{t+1} , n region center that minimizes or maximizes any similarity metric is the tracked position (flow) of the pixel (x, y) at f_{t+1} .

151 3.3. Search template

152 In optical flow, the search window size defines the maximum allowed motion to be detected in the
 153 sequence, see Fig. 4. In general, let p be a pixel in the reference image (f_t), whose 2D spatial location is
 154 defined as (x_t, y_t) , the same pixel in the tracked image (f_{t+1}) has to satisfy $x_{t+1} \in x_t - k : 1 : x_t + k$,
 155 $y_{t+1} \in y_t - k : 1 : y_t + k$, where k is the search size for the tracking step. In practice, large search region
 156 sizes increase the tracking performance since feature tracking could be carried out in both slow and
 157 fast camera movements. However, large search sizes decrease the accuracy, i.e., if the search region size
 158 is equal to 1, then, $x_{t+1} \in x_t - 1 : 1 : x_t + 1$, $y_{t+1} \in y_t - 1 : 1 : y_t + 1$ so, there are 9 possible candidates
 159 for the tracking step and the mistake possibility is equal to 8, this considering that camera movement
 160 is slow and therefore pixel displacements between images are close to zero. In other scenario, if the
 161 search region size is equal to 10, then, $x_{t+1} \in x_t - 10 : 1 : x_t + 10$, $y_{t+1} \in y_t - 10 : 1 : y_t + 10$ so, there
 162 are 100 possible candidates for the tracking step and the mistake possibility is equal to 99. In our
 163 work, we propose to use the feedback of the previous optical flow step as a dynamic search size for
 164 the current step so, if camera movement in $t - 1$ is slow, small search sizes closer to the pixels being
 165 tracked (x_t, y_t) are used. On the other hand, given fast camera movements small search sizes far from
 166 the pixels being tracked are used. This makes the tracking step compute accurate results without
 167 outliers, furthermore, the use of small search sizes decreases the computational resources usage. For
 168 practical purposes we use a search region size equal to 10 since it provides a good tradeoff between
 169 robustness/accuracy and computational resources. So, let $\Delta_{x,t-1}(x, y)$, $\Delta_{y,t-1}(x, y)$ be the optical flow
 170 at time $t - 1$, the search template for the current time is computed as shown in Eq. 7 - 8, where k is the
 171 template size.

$$\Delta'_x(x + u, y + v) = \sum_{u=-k, v=-k}^{u=k, v=k} (\text{mean} \sum_{u=-k, v=-k}^{u=k, v=k} \Delta_{x,t-1}(x, y)) \quad (7)$$

$$\Delta'_y(x+u, y+v) = \sum_{u=-k, v=-k}^{u=k, v=k} (\text{mean} \sum_{u=-k, v=-k}^{u=k, v=k} \Delta_{y, t-1}(x, y)) \quad (8)$$

172 3.4. Depth estimation

173 In previous works it was demonstrated that monocular image sequences provide only partial
 174 information about the scene due to the computation of relative depth, unknown scale factor, etc. [37].
 175 In order to recover the depth in the scene it is necessary to have assumptions about the scene and its
 176 2-D images. In this work we assume that environment within the scene is rigid, then, given the optical
 177 flow of the scene (which represents pixel velocity across time), we suppose that depth in the scene is
 178 proportional to the pixel velocity. i.e., far objects have to be associated with a low velocity value while
 179 closer objects are associated with high velocity values. This could be considered as an extension of the
 180 epipolar geometry in which disparities values are proportional with the depth in the scene, as shown
 181 in Fig. 6.

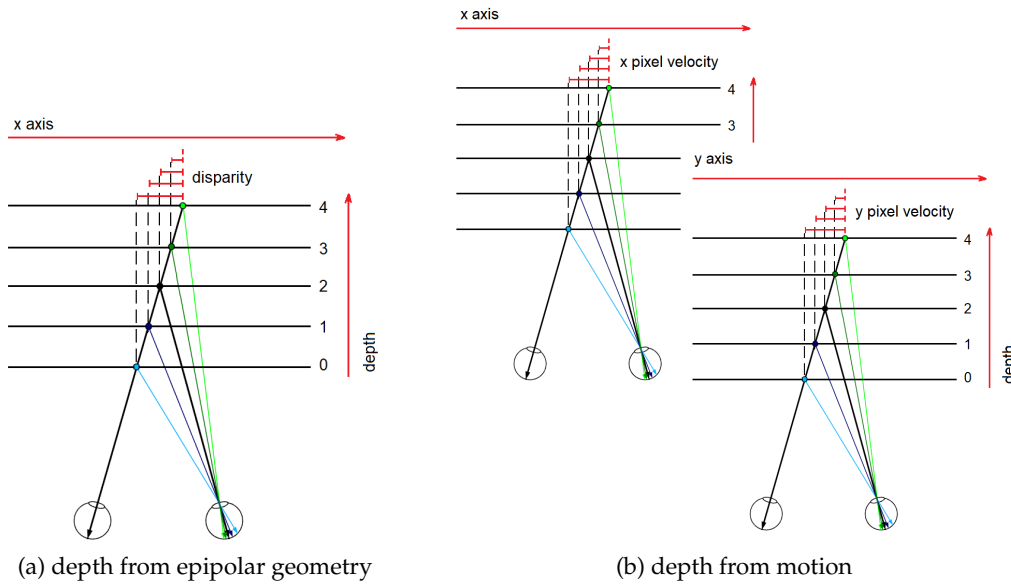


Figure 6. (a) Epipolar geometry: depth in the scene is proportional to the disparity value, i.e., far objects have low disparity values while closer objects are associated with high disparity values. To compute the disparity map (disparities for all pixels in the image) a stereo pair (two images with epipolar geometry) are needed. (b) Single moving camera: in this work we suppose that depth in the scene is proportional to the pixel velocity across the time. To compute the pixel velocity, optical flow across two consecutive frames has to be computed.

182 So, let $\Delta_x(x, y)$, $\Delta_y(x, y)$ be the optical flow (pixel velocity) at t time, depth in the scene $\mathbf{depth}(x, y)$
 183 is computed as proposed in Eq. 9, where $\mathbf{depth}(x, y)$ is the norm of the optical flow. In Fig. 7 an
 184 example of depth map computed by the proposed approach is shown.

$$\mathbf{depth}(x, y) = \|\Delta_x(x, y), \Delta_y(x, y)\| = \sqrt{\Delta_x(x, y)^2 + \Delta_y(x, y)^2} \quad (9)$$

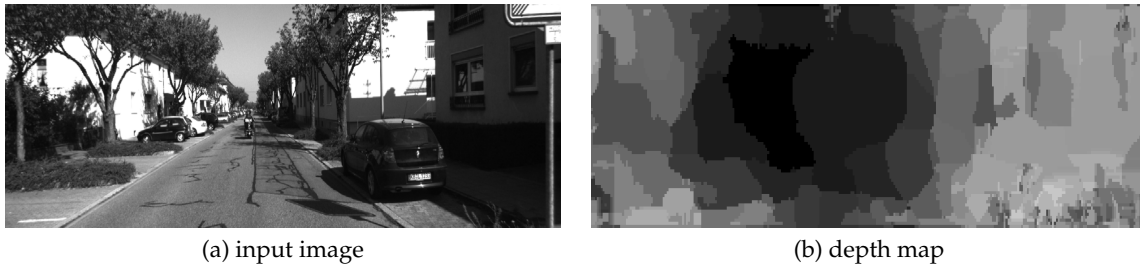


Figure 7. Depth estimation using the proposed algorithm

185 4. The FPGA architecture

186 In Fig. 8, an overview of the FPGA architecture for the proposed algorithm is shown. The
 187 architecture is centered on an FPGA implementation where all recursive/parallelizable operations
 188 are accelerated in the FPGA fabric. First, the "frame buffer" unit reads the pixel stream (pix [7:0])
 189 delivered by the imager. In this block, frames captured by the imager are feed to/from an external
 190 DRAM memory and delivers pixel streams for two consecutive frames in parallel (pix1 [7:0], pix2 [7:0]).
 191 "Circular buffers" implemented inside the "Optical flow" unit are used to hold local sections of the
 192 frames that are being processed and allow for local parallel access that facilitates parallel processing.
 193 Finally, optical flow streams (pix3 [7:0], pix4 [7:0]) are used to computed the depth in the scene (pix7
 194 [7:0]). In order to hold optical flow previously computed (which are used for the dynamic search
 195 template computation) a second "frame buffer" is used. In the following subsections details about the
 196 algorithm parallelization are shown.

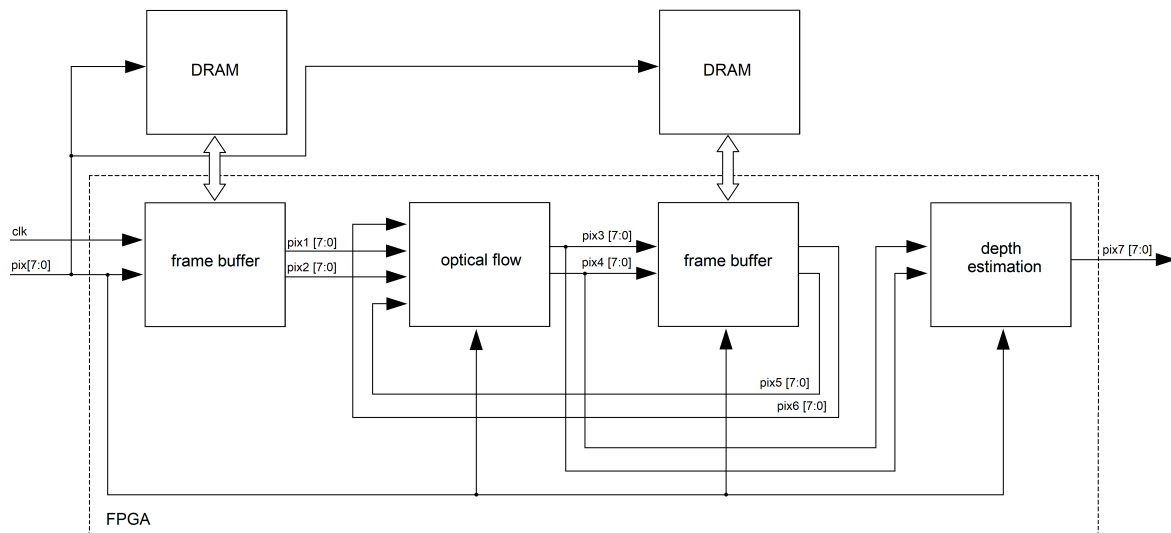


Figure 8. FPGA architecture for the proposed algorithm

197 4.1. Frame buffer

198 Images from the image sensor are stored in an external DRAM that holds an entire frame from the
 199 sequence, and later the DRAM data is read by the FPGA to cache pixel flow of the stored frame into
 200 circular buffers. In order to deliver two consecutive frames in parallel two DRAM chips in switching
 201 mode are used. i.e.:

- 202 1. t_1 : DRAM 1 in write mode (storing frame 1), DRAM 2 in read mode (invalid values), frame 1 at
 203 output 1, invalid values at output 2.
- 204 2. t_2 : DRAM 1 in read mode (reading frame 1), DRAM 2 in write mode (storing frame 1), frame 1 at
 205 output 2, frame 1 at output 2.

206 3. t_3 : DRAM 1 in **write** mode (storing frame 3), DRAM 2 in read mode (reading frame 2), frame 3 at
 207 output 2, frame 2 at output 2 and so on.

208 In **Fig. 9**, an overview of the "frame buffer" unit is shown. Current pixel stream (pix [7:0]) is
 209 mapped at output 1 (pix1 [7:0]) while output 2 (pix2 [7:0]) delivers pixel flow for a previous frame.
 210 For the external DRAM control, data [7:0] is mapped with the read/write pixel stream, address [31:0]
 211 manages the physical location inside the memory and the "we" and "re" signals enable the write/read
 212 process respectively, as shown in **Fig. 9**.

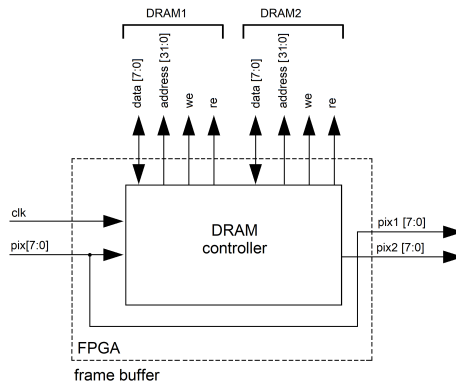


Figure 9. FPGA architecture for the "frame buffer" unit. Two external memories configured in switching mode makes possible to store the current frame (time t) into a DRAM configured in write mode while another DRAM (in read mode) deliver pixel flow for a previous frame (frame at time $t - 1$).

213 4.2. Optical flow

214 For the "Optical flow" unit, we consider that flow estimation problem can be a generalization
 215 of the dense matching problem. i.e., stereo matching algorithms track (searching on the horizontal
 216 axis around the search image), all pixels in the reference image. Optical flow aims to track all pixels
 217 between two consecutive frames from a video sequence (searching around spatial coordinates of the
 218 pixels in the search image). Then, it is possible to extend previous stereo matching FPGA architectures
 219 to fulfil with our application domain. In this work, we extended the FPGA architecture presented in
 220 [24], since it has low hardware requirements and high parallelism level. In **Fig. 10**, the developed
 221 architecture is shown. First, the "curl" units deliver curl images in parallel, see **Eq. 2**. More details
 222 about the FPGA architecture of this unit are shown in **Section 4.2.2**. The "circular buffer" units are
 223 responsible for data transfers in segments of the image (usually several rows of pixels). So, the core of
 224 the FPGA architecture are the circular buffers attached to the local processors that can hold temporarily
 225 as cache, for image sections from two frames, and that can deliver parallel data to the processors. More
 226 details about the FPGA architecture of this unit are shown in **Section 4.2.1**. Then, given optical flow
 227 previously computed, 121 search regions are constructed in parallel, see **Fig. 5** and **Eq. 4 - 5**. For our
 228 implementation, the search region size is equal to 10, therefore, the center of the search regions are all
 229 the sampled pixels within the reference region. Given the reference region in $f_t(x, y)$ and 121 search
 230 regions in $f_{t+1}(x, y)$, search regions are compared with the reference region (**Eq. 3**) in parallel. For
 231 that, a pixel-parallel/window-parallel scheme is implemented. Finally, in the "flow estimation" unit a
 232 multiplexer tree can determine the a, b indices that minimize **Eq. 3**, and therefore, the optical flow for
 233 all pixels in the reference image, using **Eq. 6**.

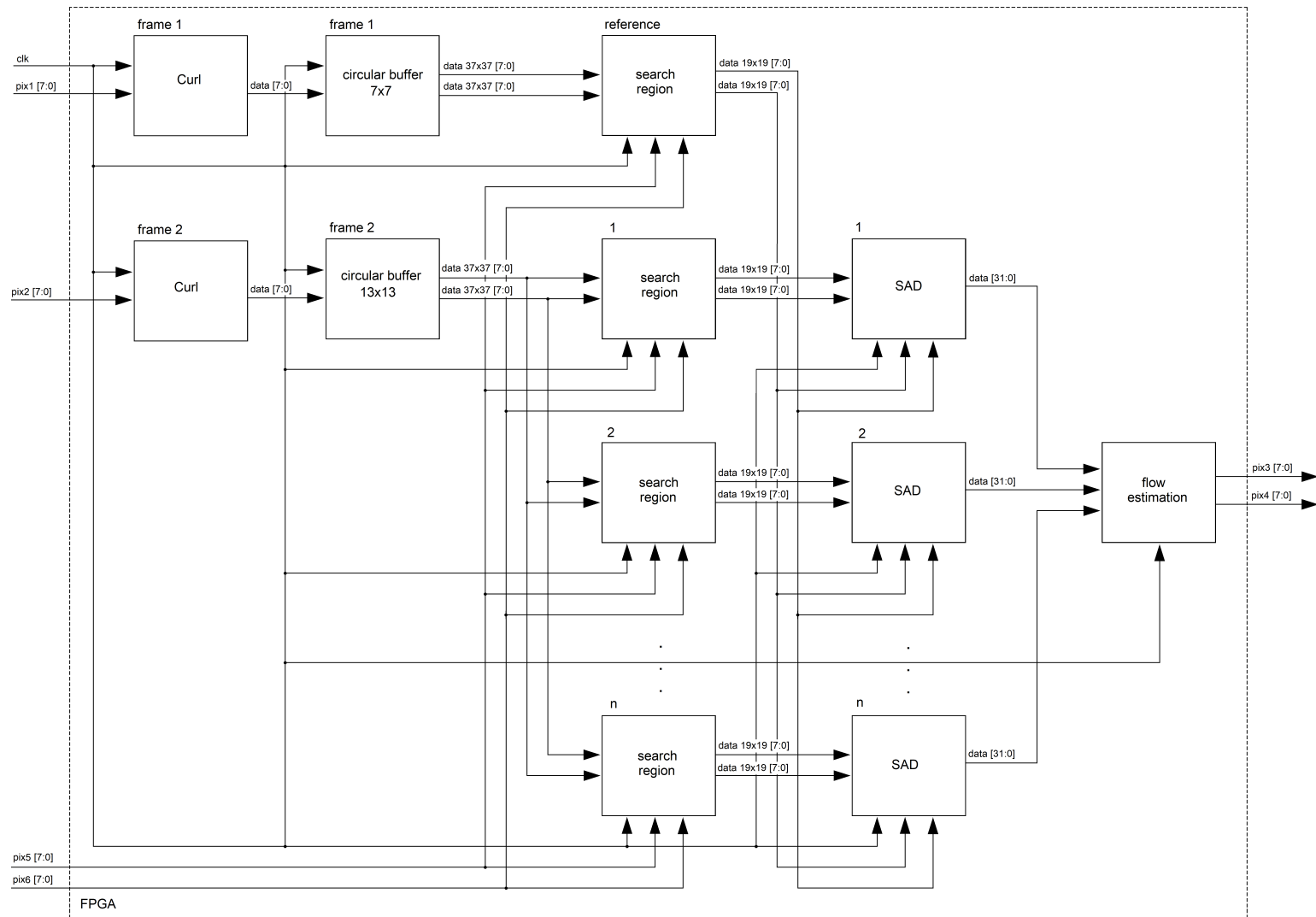
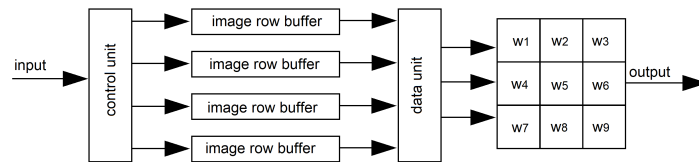


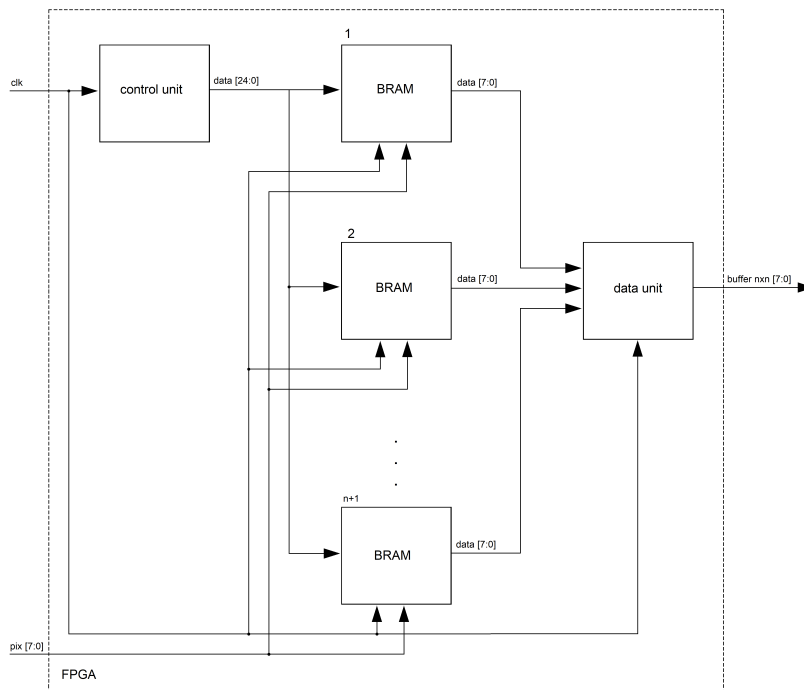
Figure 10. FPGA architecture for the optical flow estimation

234 4.2.1. Circular buffer

235 In [23] we proposed a circular buffer schema in which input data from the previous n rows of an
 236 image can be stored using memory buffers (block RAMs/BRAMs) until the moment when a $n \times n$
 237 neighborhood is scanned along subsequent rows. In this work, we follow a similar approach to achieve
 238 high data reuse and high level of parallelism. Then, our algorithm is processed in modules where all
 239 image patches can be read in parallel. First, a shift mechanism "control" unit manages the read/write
 240 addresses of $n + 1$ BRAMs, in this formulation n BRAMs are in read mode and one BRAM is in write
 241 mode in each clock cycle. Then, data inside the read mode BRAMs can be accessed in parallel and
 242 each pixel within a $n \times n$ region is delivered in parallel a $n \times n$ buffer, as shown in Fig. 11, where
 243 the "control" unit delivers control data (address and read/write enable) for the BRAM modules, one
 244 entire row is stored in each BRAM. Finally the "data" unit delivers $n \times n$ pixels in parallel. In our
 245 implementation, there is 1 circular buffer of 13×13 pixels/bytes, 1 circular buffer of 17×17 and 2
 246 circular buffers of 3×3 . For more details see [23].



(a) General formulation of a 3×3 circular buffer



(b) FPGA architecture for the circular buffers

Figure 11. The circular buffers architecture. For a $n \times n$ patch, a shift mechanism "control" unit manages the read/write addresses of $n + 1$ BRAMs. In this formulation n BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, the $n \times n$ buffer delivers logic registers with all pixels within the patch in parallel.

247 4.2.2. Curl estimation

248 In Fig. 12, the curl architecture is shown. First, one "circular buffer" holds 3 rows of the frame
 249 being processed and allows for local parallel access of a 3×3 patch that facilitates parallel processing.
 250 Then, image gradients ($\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial f(x,y)}{\partial y}$) are computed. Another "circular buffer" holds 3 rows of the
 251 gradient image previously computed and delivers a 3×3 patch for the next step. Second derivatives
 252 ($\frac{\partial}{\partial y} \frac{\partial f(x,y)}{\partial x}$, $\frac{\partial}{\partial x} \frac{\partial f(x,y)}{\partial y}$) are computed inside the "derivative" unit. Finally, the curl of the input image is
 253 computed by the "curl" unit.

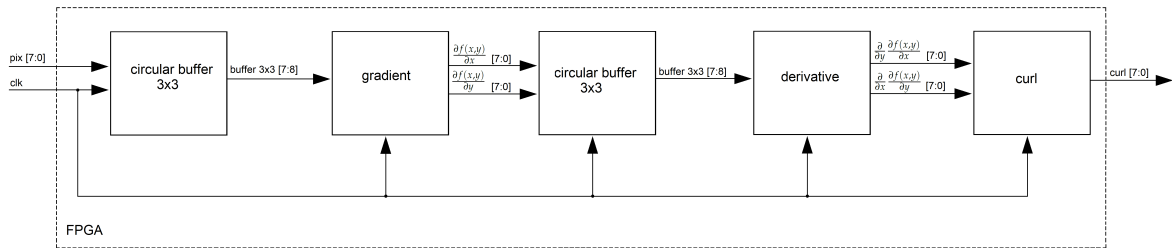


Figure 12. FPGA architecture for the "curl" unit

254 4.3. Depth estimation

255 In Fig. 13, the depth estimation architecture is shown. Let "pix1 [7:0]", "pix2 [7:0]" be the pixel
 256 stream for the optical flow at current frame (Eq. 6); first, the "multiplier" unit computes the square value
 257 of the input data. Then, the "adder" unit carries out the addition process for both components (Δ_x^2, Δ_y^2).
 258 Finally, the "sqrt" unit computes the depth in the scene, using Eq. 9. In order to achieve high efficiency
 259 in the square root computation, we adapted the architecture developed by Yamin Li and Wanming
 260 Chu [38]. This architecture uses a shift register mechanism and compares the more significant/less
 261 significant bits to achieving the root square operation without using embedded multipliers.

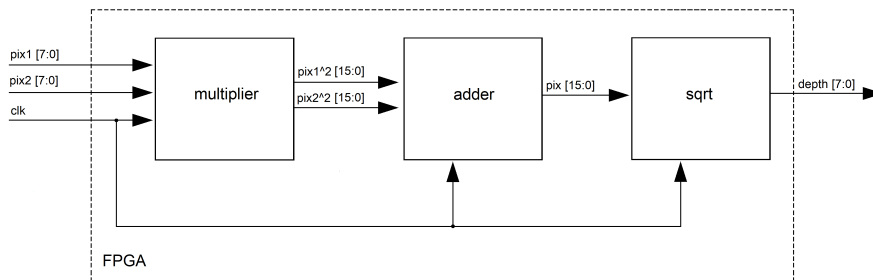


Figure 13. FPGA architecture for the "depth estimation" unit

262 5. Result and discussion

263 The developed FPGA architecture was implemented in an FPGA Cyclone IV EP4CGX150CF23C8
 264 of Altera. All modules were designed via Quartus II Web Edition version 10.1SP1 and, all modules
 265 were validated via post-synthesis simulations performed in ModelSim Altera. For all tests, we consider
 266 $k = 3$, $s = 2$ (Eq. 4 and 5) since these values provided a relatively "good" performance for real world
 267 scenarios. In practice, we recommend these values as reference. Higher $k = 3$, $s = 2$ values could
 268 provide higher accuracy, however, processing speed and hardware requirements can be increased. On
 269 the other hand, lower $k = 3$, $s = 2$ values should provide higher performance in terms of hardware
 270 requirements/processing speed but accuracy could decrease. The full hardware resource consumption
 271 of the architecture is shown in Table 1. Our algorithm formulation allows for a compact system
 272 design; it requires 66% of the total logic elements of the FPGA Cyclone IV EP4CGX150CF23C8. For

273 memory bits, our architecture uses 74% of the total resources, this represents 26 block RAMs consumed
 274 mainly in the circular buffers. These hardware utilization enables to target a relatively small FPGA
 275 device and therefore could be possible a small FPGA-based smart camera, suitable for real-time
 276 embedded applications. In the following subsections comparisons with previous work are presented.
 277 For optical flow, comparisons with previous FPGA-based optical flow algorithms are presented. **For**
 278 **depth estimation, we presented a detailed discussion about the performance and limitations of the**
 279 **proposed algorithm compared with the current state of the art.**

280 5.1. Performance for the optical flow algorithm

Table 1. Hardware resource consumption for the developed FPGA architecture.

Resource	Consumption/image resolution		
	640×480	320×240	256×256
Total logic elements	69,879 (59%)	37,059 (31%)	21,659 (18%)
Total pins	16 (3%)	16 (3%)	16 (3%)
Total Memory Bits	618,392 (15%)	163,122 (4%)	85,607 (2%)
Embedded multiplier elements	0 (0%)	0 (0%)	0 (0%)
Total PLLs	1 (25%)	1 (25%)	1 (25%)

281 In comparison with previous work, in **Table 2** we present hardware resource utilization between
 282 our FPGA architecture and previous FPGA-based optical flow algorithms. There are several works
 283 [22,25–27] whose FPGA implementations aims to parallelize all recursive operations in the original
 284 mathematical formulation. Unfortunately, most popular formulations such as those based in KTL
 285 [28] or Horn-Schunck [33], have iterative operations that are hard to parallelize. As result, most
 286 previous works have relatively high hardware occupancy/implementations compared with a full
 287 parallelizable design approach. Compared with previous works, our FPGA architecture outperform
 288 most previous works, for similar image resolution, less logic elements and memory bits than [25,29],
 289 and less logic elements and memory bits than [27]. [27] decreases the memory usage by a multiscale
 290 coding which makes possible to store only half of the original image, however, this reduction involves
 291 pixel interpolation for some cases and this increases the logic elements usage. For [22], the authors
 292 introduced an iterative-parallel approach; this makes possible to achieve low hardware requirements
 293 but processing speed is low. Finally, for [26], a filtering-based approach makes possible to achieve low
 294 hardware requirements with relatively high accuracy and high processing speed but the algorithmic
 295 formulation requires to store several entire frames, requiring large external memory (near 250 MB for
 296 store 3 entire frames), this increase the system size and cost.

Table 2. Hardware resource consumption comparisons

Method	Logic elements	Memory bits	Image resolution
Martín <i>et al.</i> [22] (2005)	11,520	147,456	256×256
Díaz <i>et al.</i> [25] (2006)	513,216	685,670	320×240
Wei <i>et al.</i> [26] (2007)	10,288	256 MB (DDR)	640×480
Barranco <i>et al.</i> [27] (2012)	82,526	573,440	640×480
Honegger <i>et al.</i> [29] (2012)	49,655	1,111,000	376×240
Our work*	69,879	624,244	640×480
Our work*	37,059	163,122	320×240
Our work*	21,659	85,607	256×256

*Operating frequency = 50 MHz

297 In **Table 3**, speed processing for different image resolutions is shown. We synthesized different
 298 versions of our FPGA architecture (**Fig. 8**), and we adapted the circular buffers in order to work with
 299 all tested image resolutions. Then, we carried out post-synthesis simulation in ModelSim Altera. In
 300 all cases, our FPGA architecture reached real-time processing. When compared with previous work
 301 (**Table 4**), our algorithm provided the highest speed processing, it outperforms several previous work
 302 [22,25–27,29], and for HD images, our algorithm reaches real-time processing: more than 60 fps for
 303 1280×1024 image resolution.

Table 3. Processing speed for different image resolutions

Resolution	Frames/s	Pixels/s
1280×1024	68	90,129,200
640×480	297	91,238,400
320×240	1,209	92,880,000
256×256	1,417	92,876,430

*Operating frequency = 50 MHz

Table 4. Processing speed comparisons

Method	Resolution	Frames/s	Pixels/s
Martín <i>et al.</i> [22]	256×256	60	3,932,160
Díaz <i>et al.</i> [25]	320×240	30	2,304,000
Wei <i>et al.</i> [26]	640×480	64	19,550,800
Barranco <i>et al.</i> [27]	640×480	31	9,523,200
Honegger <i>et al.</i> [29]	376×240	127	11,460,480
Our work	640×480	297	91,238,400

304 In **Fig. 14**, qualitative results for this work compared with previous work are shown. In a **first**
 305 **experiment** we used the “Garden” dataset since previous work [22,25,26] used this dataset as reference.
 306 When compared with previous work (**Fig. 14**), our algorithm provides high performance under
 307 real world scenarios, it outperforms several previous work [22,25,26], quantitatively closer to the
 308 ground truth (error near to 9%) compared with other FPGA-based approaches. In a **second experiment**
 309 quantitative and qualitative results for the KITTI dataset [39], are shown. In all cases our algorithm
 310 provides high performance, it reaches an error near to 10% with several test sequences, as shown in
 311 **Fig. 15**. In both experiments we compute the error by comparing the ground truth $\Omega_x(x, y)$, $\Omega_y(x, y)$
 312 (provided with the dataset) with the computed optical flow $\Delta_x(x, y)$, $\Delta_y(x, y)$. First, we compute the
 313 local error (the error magnitude at each point of the input image) as defined in **Eq. 10**; where i, j is the
 314 input image resolution. Then, a global error (Ξ) can be computed as shown in **Eq. 11**; where i, j is the
 315 input image resolution. $\zeta(x, y)$ is the local error at each pixel in the reference image and the global
 316 error (Ξ) is the percentage of pixels in the reference image in which local error is higher to zero.

$$\zeta(x, y) = \sum_{x=1}^{x=i} \sum_{y=1}^{y=j} \sqrt{\Omega_x(x, y)^2 + \Omega_y(x, y)^2} - \sqrt{\Delta_x(x, y)^2 + \Delta_y(x, y)^2} \quad (10)$$

$$\Xi = \frac{100\%}{i \cdot j} \cdot \sum_{x=1}^{x=i} \sum_{y=1}^{y=j} \begin{cases} 1 & \text{if } \zeta(x, y) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

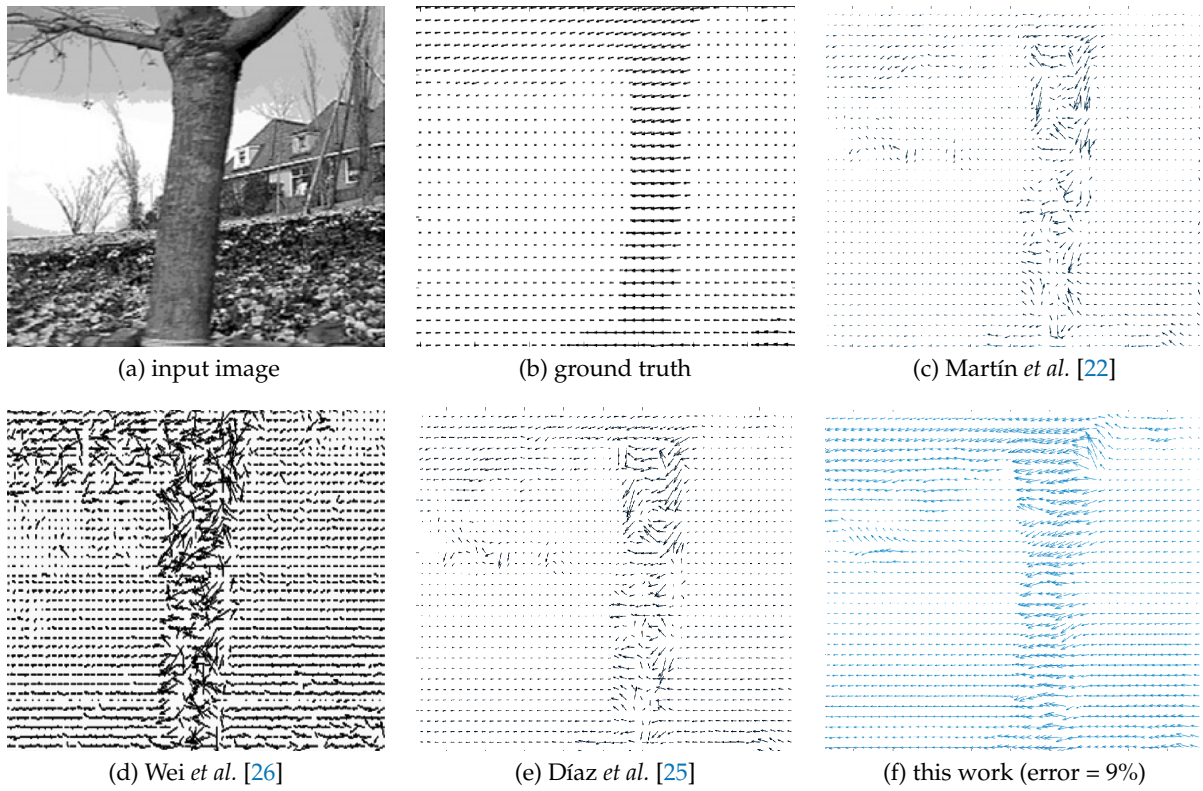


Figure 14. Accuracy performance for different FPGA-based optical flow algorithms.

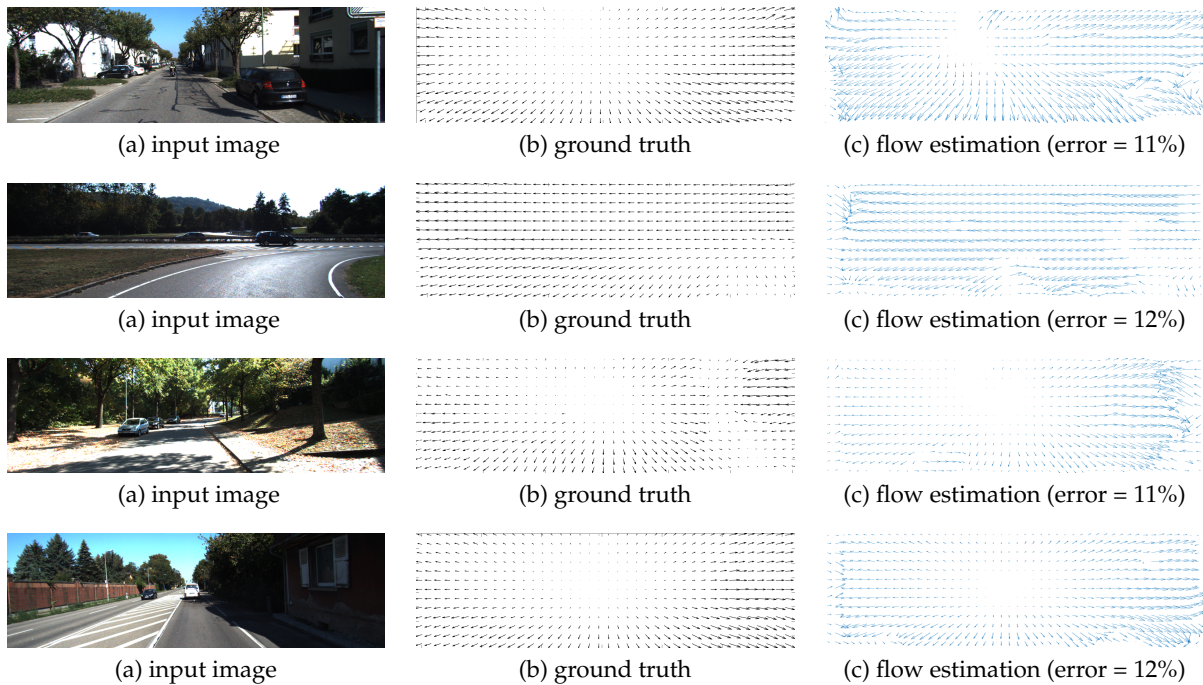


Figure 15. Optical flow: quantitative/qualitative results for the KITTI dataset

318 5.2. Performance for the depth estimation step

319 In Fig. 15, quantitative and qualitative results for the KITTI dataset [39], are shown. In all cases
 320 our algorithm provides rough depth maps compared with stereo-based or deep learning approaches
 321 [40,41] but with real-time processing and with the capability to be implemented in embedded hardware,
 322 suitable for smart cameras. To our knowledge, previous FPGA-based approaches are limited; there are
 323 several GPU-based approaches but in these cases most of the effort was for accuracy improvements and
 324 real-time processing or embedded capabilities were not considered so, in several cases, details about
 325 the hardware requirements or the processing speed are not provided [42–44]. In Table 5 quantitative
 326 comparisons between our algorithm and the current state of the art are presented. For previous
 327 works, the RMS error, hardware specifications and processing speed were obtained from the published
 328 manuscripts while for our algorithm we computed the RMS error as indicated by the KITTI dataset,
 329 [45]. For accuracy comparisons, most previous works [42–44,46–48] outperform our algorithm (near
 330 15% more accurate than ours); however, our algorithm outperform all of them in terms of processing
 331 speed (a processing speed up to 128 times faster than previous works) and with embedded capabilities
 332 (making it possible to develop a smart camera/sensor suitable for embedded applications).

Table 5. Depth estimation process in the literature: performance and limitations for the KITTI dataset.

Method	Error (RMS)	Speed	Image resolution	Approach	
Zhou <i>et al.</i> [42](2017)	6.8%	-	128×416	DfM-based*	-
Yang <i>et al.</i> [46](2017)	6.5%	5 fps	128×416	CNN-based*	GTX 1080 (GPU)
Mahjourian <i>et al.</i> [47](2018)	6.2%	100 fps	128×416	DfM-based*	Titan X (GPU)
Yang <i>et al.</i> [43](2018)	6.2%	-	830×254	DfM-based*	Titan X (GPU)
Godard <i>et al.</i> [44](2018)	5.6%	-	192×640	CNN-based*	-
Zou <i>et al.</i> [48](2018)	5.6%	1.25 fps	576×160	DfM-based*	Tesla K80 (GPU)
Our work	21.5%	192 fps	1241×376	DfM-based	Cyclone IV (FPGA)

*DfM: Depth from Motion, CNN: Convolutional Neural Network

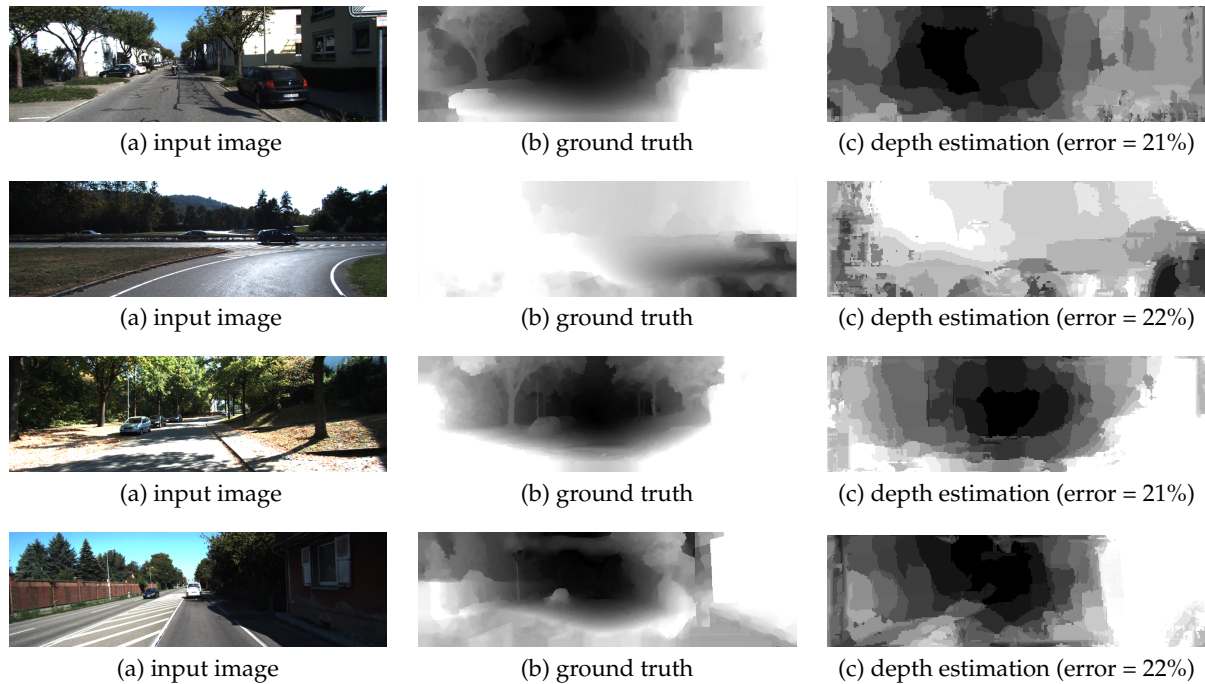
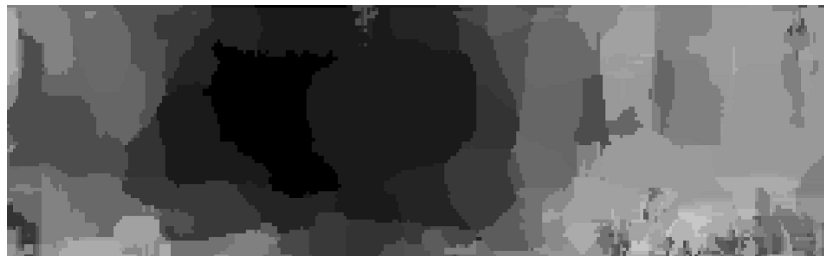


Figure 16. Depth estimation: quantitative/qualitative results for the KITTI dataset

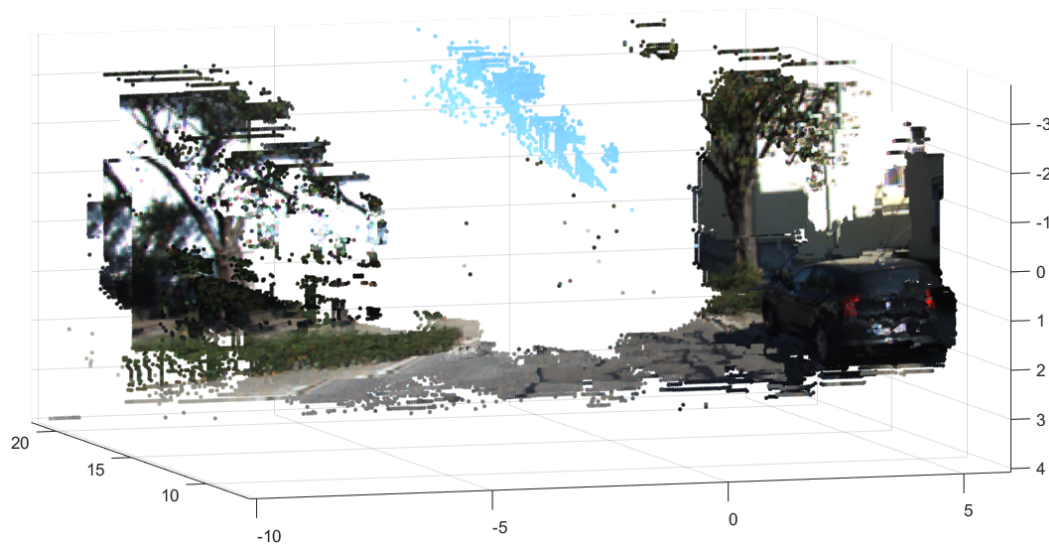
333 Finally, in Fig. 17 an example of 3D reconstruction using our approach is shown. Our depth maps
334 allow for a real-time dense 3D reconstruction. Previous works like the ORB-SLAM [49] or LSD-SLAM
335 [50] compute motion and depth in 2 to 7% of all image pixels, while ours compute 80% of the image
336 pixels. Then, our algorithm improves by around 15 times the current state of the art, making possible
337 real-time dense 3D reconstructions and with the capability to be implemented inside FPGA devices,
338 suitable for smart cameras.
339



(a) Input image



(b) Depth map



(c) 3D reconstruction

Figure 17. The KITTI dataset: Sequence 00; 3D reconstruction by the proposed approach. Our algorithm provides rough depth maps (lower accuracy compared with previous algorithms) but with real-time processing and with the capability to be implemented in embedded hardware; as result, real-time dense 3D reconstructions can be obtained and, these can be exploited by several real world applications such as, augmented reality, robot vision and surveillance, autonomous flying, etc.

6. Conclusions

Depth from Motion is the problem of depth estimation using information from a single moving camera. Although several Depth from Motion algorithms were developed, previous works have low processing speed and high hardware requirements that limits the embedded capabilities. In order to solve these limitations in this work we have proposed a new depth estimation algorithm whose FPGA implementation deliver high efficiency in terms of algorithmic parallelization. Unlike previous works, depth information is estimated in real time inside a compact FPGA device, making our mathematical formulation suitable for smart embedded applications.

Compared with the current state of the art, previous algorithms outperform our algorithm in terms of accuracy but our algorithm outperforms all previous approaches in terms of processing speed and hardware requirements; these characteristics makes our approach a promising solutions for the current embedded systems. We believed that several real world applications such as augmented reality, robot vision and surveillance, autonomous flying, etc., can take advantages by applying our algorithm since it delivers real-time depth maps that can be exploited to create dense 3D reconstructions or other abstractions useful for the scene understanding.

Author Contributions: Conceptualization, Abiel Aguilar-González, Miguel Arias-Estrada and François Berry. Investigation, Validation and Writing—Original Draft Preparation: Abiel Aguilar-González. Supervision and Writing—Review & Editing: Miguel Arias-Estrada and François Berry.

Funding: This research received no external funding.

Acknowledgments: This work has been sponsored by the French government research program "Investissements d'avenir" through the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the program Regional competitiveness and employment, and by the Auvergne region. This work has been sponsored by Campus France through the scholarship program "bourses d'excellence EIFFEL", dossier No. MX17-00063 and by the National Council for Science and Technology (CONACyT), Mexico, through the scholarship No. 567804..

Conflicts of Interest: The authors declare no conflict of interest.

- Hengstler, S.; Prashanth, D.; Fong, S.; Aghajan, H. MeshEye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. Proceedings of the 6th international conference on Information processing in sensor networks. ACM, 2007, pp. 360–369.
- Aguilar-González, A.; Arias-Estrada, M. Towards a smart camera for monocular SLAM. Proceedings of the 10th International Conference on Distributed Smart Camera. ACM, 2016, pp. 128–135.
- Carey, S.J.; Barr, D.R.; Dudek, P. Low power high-performance smart camera system based on SCAMP vision sensor. *Journal of Systems Architecture* **2013**, *59*, 889–899.
- Birem, M.; Berry, F. DreamCam: A modular FPGA-based smart camera architecture. *Journal of Systems Architecture* **2014**, *60*, 519–527.
- Bourrasset, C.; Maggiani, L.; Sérot, J.; Berry, F.; Pagano, P. Distributed FPGA-based smart camera architecture for computer vision applications. Distributed Smart Cameras (ICDSC), 2013 Seventh International Conference on. IEEE, 2013, pp. 1–2.
- Bravo, I.; Baliñas, J.; Gardel, A.; Lázaro, J.L.; Espinosa, F.; García, J. Efficient smart cmos camera based on fpgas oriented to embedded image processing. *Sensors* **2011**, *11*, 2282–2303.
- Köhler, T.; Röchter, F.; Lindemann, J.P.; Möller, R. Bio-inspired motion detection in an FPGA-based smart camera module. *Bioinspiration & biomimetics* **2009**, *4*, 015008.
- Olson, T.; Brill, F. Moving object detection and event recognition algorithms for smart cameras. Proc. DARPA Image Understanding Workshop, 1997, Vol. 20, pp. 205–208.
- Norouznezhad, E.; Bigdeli, A.; Postula, A.; Lovell, B.C. Object tracking on FPGA-based smart cameras using local oriented energy and phase features. Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras. ACM, 2010, pp. 33–40.
- Fularz, M.; Kraft, M.; Schmidt, A.; Kasiński, A. The architecture of an embedded smart camera for intelligent inspection and surveillance. In *Progress in Automation, Robotics and Measuring Techniques*; Springer, 2015; pp. 43–52.

- 390 11. Haritaoglu, I.; Harwood, D.; Davis, L.S. W 4: Real-time surveillance of people and their activities. *IEEE*
391 *Transactions on pattern analysis and machine intelligence* **2000**, *22*, 809–830.
- 392 12. Biswas, J.; Veloso, M. Depth camera based indoor mobile robot localization and navigation. *Robotics and*
393 *Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1697–1702.
- 394 13. Stowers, J.; Hayes, M.; Bainbridge-Smith, A. Altitude control of a quadrotor helicopter using depth map
395 from Microsoft Kinect sensor. *Mechatronics (ICM), 2011 IEEE International Conference on*. IEEE, 2011, pp.
396 358–362.
- 397 14. Scharstein, D.; Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence
398 algorithms. *International journal of computer vision* **2002**, *47*, 7–42.
- 399 15. Subbarao, M.; Surya, G. Depth from defocus: a spatial domain approach. *International Journal of Computer*
400 *Vision* **1994**, *13*, 271–294.
- 401 16. Chen, Y.; Alain, M.; Smolic, A. Fast and accurate optical flow based depth map estimation from light fields.
402 *Irish Machine Vision and Image Processing Conference (IMVIP), 2017*.
- 403 17. Zhang, J.; Singh, S. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *Robotics and*
404 *Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2174–2181.
- 405 18. Schubert, S.; Neubert, P.; Protzel, P. Towards camera based navigation in 3d maps by synthesizing depth
406 images. *Conference Towards Autonomous Robotic Systems*. Springer, 2017, pp. 601–616.
- 407 19. Maddern, W.; Newman, P. Real-time probabilistic fusion of sparse 3d lidar and dense stereo. *Intelligent*
408 *Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2181–2188.
- 409 20. Dai, A.; Chang, A.X.; Savva, M.; Halber, M.; Funkhouser, T.A.; Nießner, M. ScanNet: Richly-Annotated 3D
410 Reconstructions of Indoor Scenes. **2017**. *2*, 10.
- 411 21. Liu, H.; Li, C.; Chen, G.; Zhang, G.; Kaess, M.; Bao, H. Robust Keyframe-based Dense SLAM with an
412 RGB-D Camera. *arXiv preprint arXiv:1711.05166* **2017**.
- 413 22. Martín, J.L.; Zuloaga, A.; Cuadrado, C.; Lázaro, J.; Bidarte, U. Hardware implementation of optical flow
414 constraint equation using FPGAs. *Computer Vision and Image Understanding* **2005**, *98*, 462–490.
- 415 23. Aguilar-González, A.; Arias-Estrada, M.; Pérez-Patricio, M.; Camas-Anzueto, J. An FPGA 2D-convolution
416 unit based on the CAPH language. *Journal of Real-Time Image Processing* **2015**, pp. 1–15.
- 417 24. Pérez-Patricio, M.; Aguilar-González, A.; Arias-Estrada, M.; Hernandez-de Leon, H.R.; Camas-Anzueto,
418 J.L.; de Jesús Osuna-Coutiño, J. An fpga stereo matching unit based on fuzzy logic. *Microprocessors and*
419 *Microsystems* **2016**, *42*, 87–99.
- 420 25. Díaz, J.; Ros, E.; Pelayo, F.; Ortigosa, E.M.; Mota, S. FPGA-based real-time optical-flow system. *IEEE*
421 *transactions on circuits and systems for video technology* **2006**, *16*, 274–279.
- 422 26. Wei, Z.; Lee, D.J.; Nelson, B.E. FPGA-based Real-time Optical Flow Algorithm Design and Implementation.
423 *Journal of Multimedia* **2007**, *2*.
- 424 27. Barranco, F.; Tomasi, M.; Diaz, J.; Vanegas, M.; Ros, E. Parallel architecture for hierarchical optical
425 flow estimation based on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2012**,
426 *20*, 1058–1067.
- 427 28. Tomasi, C.; Kanade, T. Detection and tracking of point features. *School of Computer Science, Carnegie Mellon*
428 *Univ. Pittsburgh* **1991**.
- 429 29. Honegger, D.; Greisen, P.; Meier, L.; Tanskanen, P.; Pollefeys, M. Real-time velocity estimation based on
430 optical flow and disparity matching. *IROS, 2012*, pp. 5177–5182.
- 431 30. Chao, H.; Gu, Y.; Napolitano, M. A survey of optical flow techniques for robotics navigation applications.
432 *Journal of Intelligent & Robotic Systems* **2014**, *73*, 361–372.
- 433 31. Dosovitskiy, A.; Fischer, P.; Ilg, E.; Hausser, P.; Hazirbas, C.; Golkov, V.; Van Der Smagt, P.; Cremers, D.;
434 Brox, T. FlowNet: Learning optical flow with convolutional networks. *Proceedings of the IEEE International*
435 *Conference on Computer Vision, 2015*, pp. 2758–2766.
- 436 32. Ilg, E.; Mayer, N.; Saikia, T.; Keuper, M.; Dosovitskiy, A.; Brox, T. FlowNet 2.0: Evolution of optical flow
437 estimation with deep networks. *IEEE conference on computer vision and pattern recognition (CVPR),*
438 *2017, Vol. 2*, p. 6.
- 439 33. Horn, B.K.; Schunck, B.G. Determining optical flow. *Artificial intelligence* **1981**, *17*, 185–203.
- 440 34. Khaleghi, B.; Shahabi, S.M.A.; Bidabadi, A. Performace evaluation of similarity metrics for stereo
441 corresponce problem. *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*.
442 *IEEE, 2007*, pp. 1476–1478.

- 443 35. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *The International*
444 *Journal of Robotics Research* **2013**, *32*, 1231–1237.
- 445 36. Baker, S.; Scharstein, D.; Lewis, J.; Roth, S.; Black, M.J.; Szeliski, R. A database and evaluation methodology
446 for optical flow. *International Journal of Computer Vision* **2011**, *92*, 1–31.
- 447 37. Fortun, D.; Bouthemy, P.; Kervrann, C. Optical flow modeling and computation: a survey. *Computer Vision*
448 *and Image Understanding* **2015**, *134*, 1–21.
- 449 38. Li, Y.; Chu, W. A new non-restoring square root algorithm and its VLSI implementations. Computer Design:
450 VLSI in Computers and Processors, 1996. ICCD'96. Proceedings., 1996 IEEE International Conference on.
451 IEEE, 1996, pp. 538–544.
- 452 39. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *The International*
453 *Journal of Robotics Research* **2013**, *32*, 1231–1237.
- 454 40. Fu, H.; Gong, M.; Wang, C.; Batmanghelich, K.; Tao, D. Deep Ordinal Regression Network for Monocular
455 Depth Estimation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR); , 2018.
- 456 41. Bo Li, Yuchao Dai, M.H. Monocular Depth Estimation with Hierarchical Fusion of Dilated CNNs and
457 Soft-Weighted-Sum Inference **2018**.
- 458 42. Zhou, T.; Brown, M.; Snavely, N.; Lowe, D.G. Unsupervised learning of depth and ego-motion from video.
459 CVPR, 2017, Vol. 2, p. 7.
- 460 43. Yang, Z.; Wang, P.; Wang, Y.; Xu, W.; Nevatia, R. LEGO: Learning Edge with Geometry all at Once by
461 Watching Videos. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018,
462 pp. 225–234.
- 463 44. Godard, C.; Mac Aodha, O.; Brostow, G. Digging Into Self-Supervised Monocular Depth Estimation. *arXiv*
464 *preprint arXiv:1806.01260* **2018**.
- 465 45. Uhrig, J.; Schneider, N.; Schneider, L.; Franke, U.; Brox, T.; Geiger, A. Depth Prediction
466 Evaluation. [http://www.cvlibs.net/datasets/kitti/eval_odometry_detail.php?&result=](http://www.cvlibs.net/datasets/kitti/eval_odometry_detail.php?&result=fee1ecc5afe08bc002f093b48e9ba98a295a79ed)
467 [fee1ecc5afe08bc002f093b48e9ba98a295a79ed](http://www.cvlibs.net/datasets/kitti/eval_odometry_detail.php?&result=fee1ecc5afe08bc002f093b48e9ba98a295a79ed), 2017.
- 468 46. Yang, Z.; Wang, P.; Xu, W.; Zhao, L.; Nevatia, R. Unsupervised Learning of Geometry with Edge-aware
469 Depth-Normal Consistency. *arXiv preprint arXiv:1711.03665* **2017**.
- 470 47. Mahjourian, R.; Wicke, M.; Angelova, A. Unsupervised Learning of Depth and Ego-Motion from Monocular
471 Video Using 3D Geometric Constraints. Proceedings of the IEEE Conference on Computer Vision and
472 Pattern Recognition, 2018, pp. 5667–5675.
- 473 48. Zou, Y.; Luo, Z.; Huang, J.B. Df-net: Unsupervised joint learning of depth and flow using cross-task
474 consistency. European Conference on Computer Vision. Springer, 2018, pp. 38–55.
- 475 49. Mur-Artal, R.; Montiel, J.M.M.; Tardos, J.D. ORB-SLAM: a versatile and accurate monocular SLAM system.
476 *IEEE Transactions on Robotics* **2015**, *31*, 1147–1163.
- 477 50. Engel, J.; Schöps, T.; Cremers, D. LSD-SLAM: Large-scale direct monocular SLAM. European Conference
478 on Computer Vision. Springer, 2014, pp. 834–849.

479 **Sample Availability:** Samples of the compounds are available from the authors.

480 © 2018 by the authors. Submitted to *Sensors* for possible open access publication under the terms and conditions
481 of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).