Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 973475, 11 pages
doi:10.1155/2009/973475

Research Article

Parallel Processor for 3D Recovery from Optical Flow

Jose Hugo Barron-Zambrano, Fernando Martin del Campo-Ramirez, and Miguel Arias-Estrada

Computer Science Department, National Institute of Astrophysics, Optics and Electronics, 72840 Puebla, Mexico

Correspondence should be addressed to Jose Hugo Barron-Zambrano, jhbarronz@inaoep.mx

Received 16 March 2009; Revised 10 August 2009; Accepted 8 October 2009

Recommended by Cesar Torres

3D recovery from motion has received a major effort in computer vision systems in the recent years. The main problem lies in the number of operations and memory accesses to be performed by the majority of the existing techniques when translated to hardware or software implementations. This paper proposes a parallel processor for 3D recovery from optical flow. Its main feature is the maximum reuse of data and the low number of clock cycles to calculate the optical flow, along with the precision with which 3D recovery is achieved. The results of the proposed architecture as well as those from processor synthesis are presented.

1. Introduction

3D recovery from a sequence of frames has been one of the main efforts in computational vision. Unlike the case of a single static image, the information contained in a sequence is important because it allows the motion and structure recognition of the objects in a scene. The purpose of this technique is to build a 3D model from an object, using data extracted from the analysis of a 2D representation of such object. The implementation of a depth recovery algorithm depends highly on the application to develop, and the search of higher precision in 3D recovery has led to the implementation of more complex and computational demanding algorithms [1]. The main methods for 3D recovery are stereo vision, shading, and motion estimation. Stereo algorithms perform the calibration of two or more cameras to then triangulate the computing points of the scene [2]. Other methods try to recover the depth information from the motion and intensity of elements in the images. The methods based on correspondence try to perform pairing between images features and then estimate a single motion during the image sequence, to finally apply triangulation, as the stereo algorithms do [2]. These methods present good performance, but only when there is a correspondence between features, which recover only disperse surfaces.

Other techniques propose to perform a depth estimation using operations that manipulate the changes in the intensity of the image sequence and incorporate the information in a Kalman filter [3]. The problem in correspondence and intensity methods is that neither the features nor the intensity of the images is constant or continuous, decreasing the reliance of the results.

There are also other methods that compute the depth from known objects in the scene. Other techniques for depth estimation proposed to calculate the normal component of the flow [4]. For its correct operation it is necessary to know the trajectory followed between the camera and the scenery [3]. Most of the algorithms impose restrictions as the knowledge of the camera motion, its position with respect to the scenery or complicated calibration techniques.

In the case of the optical flow approach, recovery can be obtained by a camera without knowing its parameters. Neither multiple cameras aligning nor previous knowledge of the scene or the motion is necessary. All what is needed is the relative motion between the camera and the scene to be small.

The present paper introduces an FPGA-based processor for the 3D recovery from the optical flow under a static environment, so no object performs a movement in the scene. The motion of the camera along each image of the video sequence must be short, that is, a maximum of two

pixels per frame. The processor meets the constraint that it is capable of operating in near video rate time, that is, 22 frames per second (fps) for images with *Video Graphics Array* (VGA) resolution: 640 × 480 pixels.

The paper is divided as follows. Section 2 includes an analysis of related works in the field. Section 3 describes the theoretical bases for the development of the research. The functional description and interaction of the processor blocks are discussed in Section 4. While in Sections 5 and 6, performance analysis and results of the architecture are presented. Finally, conclusions and future work are presented in Section 7.

2. Background

There have been several efforts to solve the problem of depth recovery according to the characteristics and constraints of the applications to develop. Fife and Archibald [5] report an implementation for the navigation of autonomous vehicles, using feature correspondence. For each frame in the sequence, the system locates all the identified features in the previous frame, and then update the actual position estimation in the 3D space. This implementation was done by reconfigurable computing and the use of embedded processors. The performance obtained by this implementation is 30 fps with a resolution of 320 × 240 pixels. The main disadvantage of this architecture is that it only computes Z (the depth) for specific points in the video sequence.

In [6], Diaz et al. present an FPGA implementation that uses structured light. A code simplification is performed in this work by looking for redundant operations. Moreover, fixed point arithmetic is used under a format of 14 bits for the integer part and 5 or 6 bits for the fractional one. The processing time is around 20 seconds for images with a resolution of 640 × 480 pixels.

Zach et al. [7] present a work for dense 3D recovery in stereo images. Their implementation was built as a hardware-software codesign: the hardware part of the system is based on a pairing procedure to avoid the accuracy loss due to the limited resolution in 3D processors. Once the pairing is performed, depth recovery is obtained applying epipolar geometry. The software section only performs the flow control and the information transfer. A calculation of over 130,000 depth values per second running on a standard computer is reported.

3. Theoretical Framework

3.1. Optical Flow. Optical Flow is the apparent motion of patterns of objects, surfaces, and edges caused by the relative change of position among the observer (a camera) and the scene [8, 9]. There are, in the literature, comparative works of hardware implementations for several optical flow algorithms [10–12]. Basically, the majority of the optical flow implementations are based on two types of algorithms: gradient-based algorithms and correlation-based algorithms.

The gradient-based algorithms calculate the optical flow with space-time derivatives of the intensity of the pixels in an image, or through the filtered versions of the images (using low-pass and high-pass filters) [13].

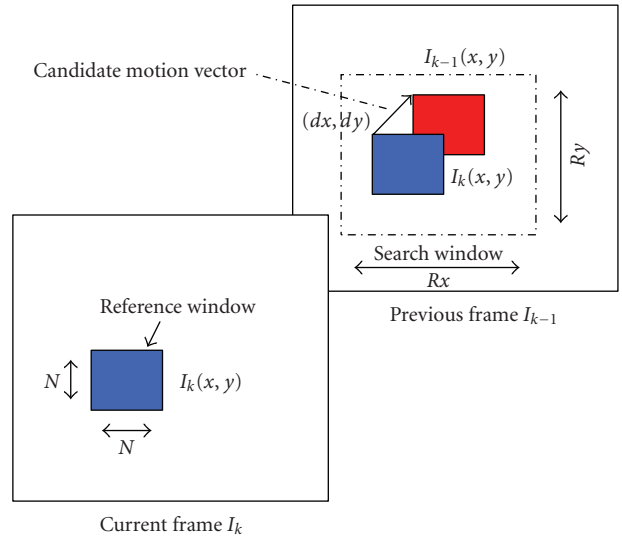

Figure 1: Basic operation of the correlation algorithms.

On the other hand, correlation-based algorithms work by comparing windows or blocks: two consecutive frames are taken from the video sequence, that are divided in periodic and equal size blocks. These blocks can present overlapping but always maintaining the same size. Given the regularity of the operations, correlation-based algorithms are better suited for hardware implementation. Figure 1 shows a graphical representation of the algorithms based on correlation.

One of the simplest correlation metrics found in the literature is the Sum of Absolute Differences (SAD) [14]. The main characteristics are its easy implementation and its reduced use of hardware resources

$$\mathrm{SAD} = \sum_{m=x}^{x+N-1} \sum_{n=y}^{y+N-1} \left| I_g(m,n) - I_{g-1}(m+dx, n+dy) \right|. \quad (1)$$

3.2. 3D Recovery from Optical Flow. This section discusses the equations that describe the relation among the depth estimation and the optical flow generated by the camera motion. The same notation found in [15] is used here. It is also assumed that the camera motion is through a static environment. The reference coordinate system is shown in Figure 2.

The coordinate system X, Y, is fixed with respect to the camera. The Z axis is located across the camera optical axis so any motion can be described by two variables: translation and rotation. $\vec{T}$ denotes the translational component of the camera, while $\vec{\omega}$ the angular velocity. Finally, the instant coordinates of the point P in the environment are (X, Y, Z) [15]. From these variables, (2) can be obtained, and from it the value of Z can be calculated

$$Z = \frac{\alpha^2 + \beta^2}{(u - u_r)\alpha + (v - v_r)\beta}, \quad (2)$$

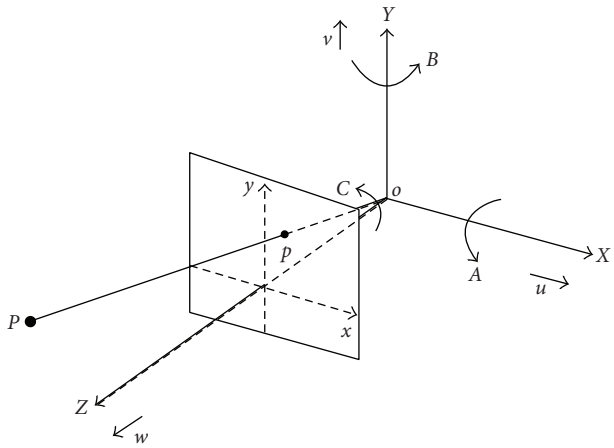

Figure 2: Reference coordinate system.

where α, β, u_r, and v_r are defined as

$$\begin{aligned} \alpha &= -U + xW, \\ \beta &= -V + yW, \\ u_r &= Axy - B(x^2 + 1) + Cy, \\ v_r &= A(y^2 + 1) - Bxy - Cx. \end{aligned} \tag{3}$$

Equation (2) calculates Z in terms of the parameters of the translation and rotation components. As there is no calibration on the camera, these parameters are still unknown. Nevertheless, their values are useful only to scale the value of Z and they do not affect the recovered structure. Therefore, it is possible to assume constant values for each of these parameters. The disadvantage of this consideration is that only a relative value of the depth information can be obtained.

4. Architecture

The proposed architecture has the purpose of recovering the 3D information from the optical flow found in a video sequence. The system presents a maximum reuse of data and is optimized for minimum hardware resources. Processing is achieved in a predictable time, that is, under real-time constrains. The architecture meets the following specifications: it works with images in 256 levels gray scale, and a resolution of 640 × 480 pixels. The image rate processing obtained 22 frames per second limited by the FPGA platform capacity, and maintaining a low *relative error* for the 3D recovery.

4.1. Description. The architecture is divided in three main functional blocks: the first one is for the calculation of the optical flow, the second one for the 3D recovery, and the last one is dedicated to data routing. The general operation of the design is as follows. The data corresponding to the pixels of the reference and search windows from a consecutive image pair in the sequence are sent to the system through a data bus.

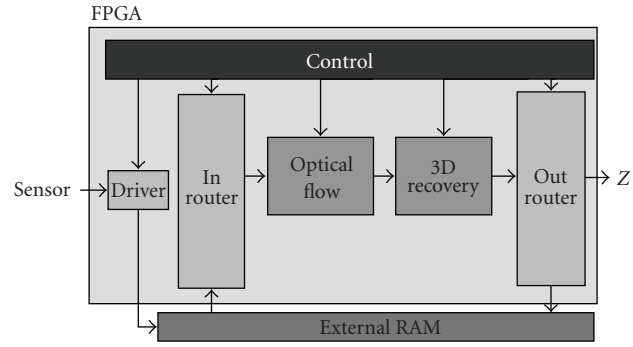

Figure 3: General block diagram of the architecture.

Subsequently, the optical flow processing is carried out. Here the motion vectors are obtained and then sent to the block for 3D recovery. Finally, the obtained values are presented to the real world, stored in external memories or sent to another process through an output data bus. The read and write addresses are generated by the routers that control the data flow. The signals that control the architectures operation are arranged in a control bus. The architecture is shown in Figure 3.

4.2. Optical Flow Module. The Optical Flow Module operates with a 4 × 4 pixels reference window and a 8 × 8 pixels search window. These values are usually used in the literature [14]. Due to the nature of (4), the Optical Flow Module can be formulated as a window-based operator considering that the coefficients of the window mask are variable and new windows are extracted from the first image to constitute the reference window. Once the processing in the search area has been completed, the window reference can be replaced with a new one, and the processing goes on the same way until all data is processed.

The number of operations per second (OPS) for the calculation of the motion estimation is given by

$$\text{OPS} = 3 * 2p * 2p * Nh * Nv * f, \tag{4}$$

where N_h and N_v are, respectively, the vertical and horizontal of the image in pixels, p is the size of the search window, and f represents the frames per second rate. For this particular work, $p = 8$, $f = 5$, $Nh = 640$, and $Nv = 480$, the result of (4) indicates a computational charge of 235,929,600 integer arithmetic operations per second. Thus, a sequential approach or the use of a general purpose processor is inadequate and insufficient for the motion estimation. The Optical Flow Module is composed by a series of basic blocks called Processing Elements (PEs). Each PE is in charge of the subtraction operations among pixels and the absolute value computation for the SAD equation. The block diagram of a PE is depicted in Figure 4.

A Window Processing Module (WPM) is assembled with 20 PEs working in parallel (Figure 4), where a set of operations are performed at the same time in a single clock cycle. The processing elements work in a systolic

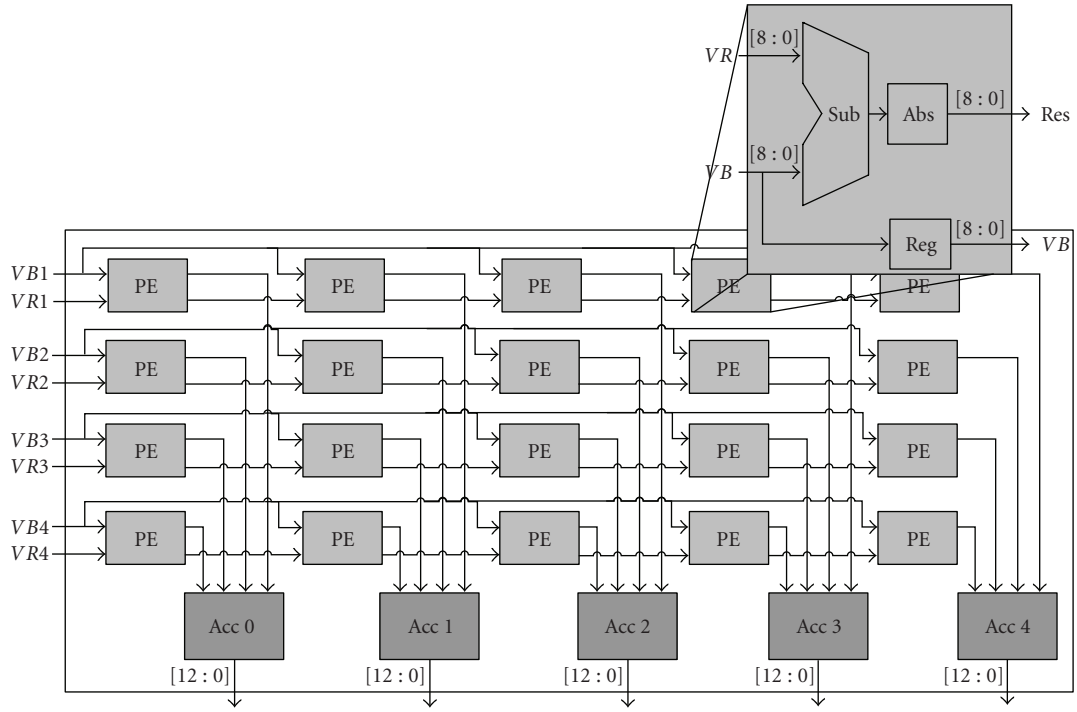

Figure 4: Block diagram of the Processing Element PE (basic building block). The Window Processing Module (WPM) integrates 20 PEs.

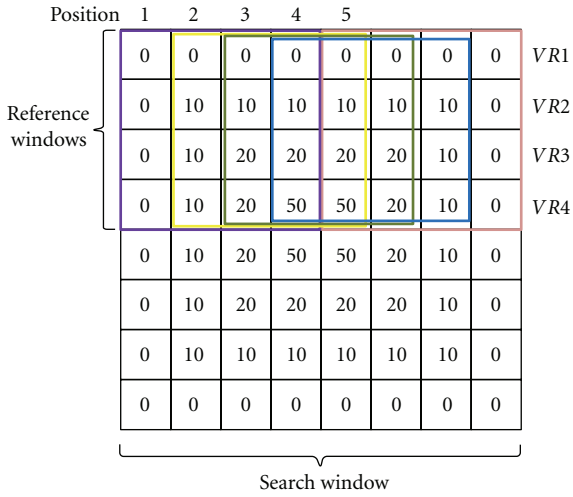

Figure 5: Computation of the correlation between the search and the reference window with a WPM.

pipeline. When the window data moves through the buffers to the next pixel location in the input image, several pixels are overlapped with the previous windows. Therefore, it is possible to use multiple instances of the WPM to compute incrementally at several consecutive pixel locations partial results of the window comparison operation. An accumulator adds together the partial results until all the data has been processed. Then, the accumulator is reset and a new window comparison operation is started. In the current implementation, the reference window is 4×4 pixels and the search window is 8×8 pixels.

The WPM performs in four clock cycles the computation of the correlation between the search and the reference window, but with the advantage that while calculating the correlation in the first position inside the search window, the correlations corresponding to the adjoining three positions of the reference window begin to be calculated (Figure 5). The design uses data recycling, taking advantage of the same information previously read from external memory more than once, to perform several calculations in the same WPM. The WPM is replicated 5 times to cover the whole search window.

The optical flow module presents a maximum data recycling, exploiting the vertical and horizontal overlap of the reference window (Figure 6). In addition, the processing time using this implementation is reduced 50 times with respect to the sequential approach.

A full search in a window is processed in 8 clock cycles and the full image in 2,386,432 clock cycles. The motion vectors are calculated pixel by pixel, contrary to other hardware implementations where the motion vectors are obtained only for each reference window. In Figure 7, an approximation of the performance, obtained experimentally, of the architecture for the calculation of the motion estimation through optical flow is shown. The necessary clock cycles for the processing of the reference and search windows can be seen in (a), while (b) shows the number of clock cycles that are necessary for the processing of a full image. Both quantities depend on the number of PE blocks used.

4.3. 3D Recovery Module. The implementation of equation (2) can be achieved in two ways: the first one is to implement

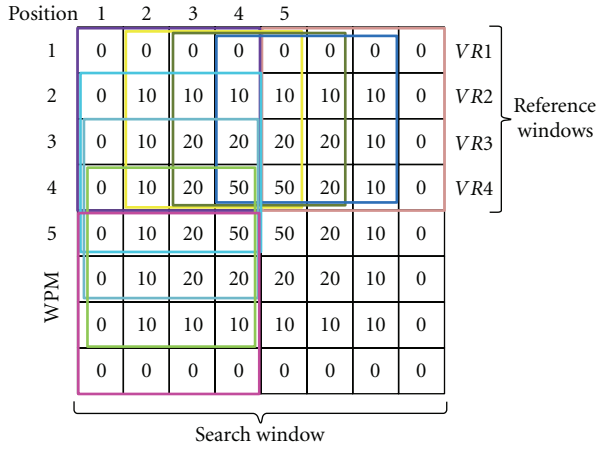

Figure 6: Computation of the correlation between the search and the reference window with 5 WPMs.

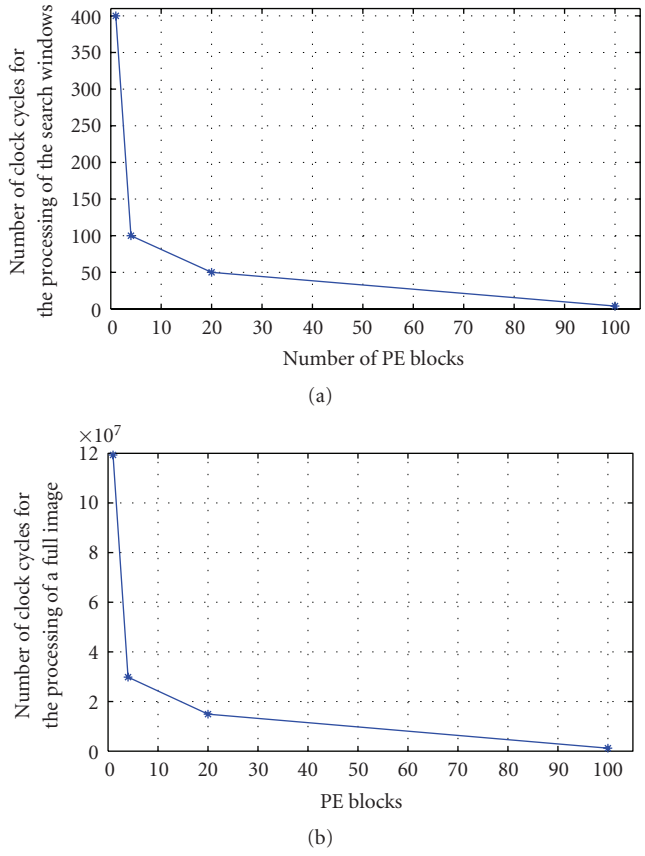

Figure 7: Performance analysis of the Optical Flow estimation for different PEs per WPM. The best area/performance compromise is around 20 PEs per WPM.

the equation with fully combinational elements. This option is inconvenient due to the complexity of the mathematical operations, which can lead to a significant degradation of the architecture performance. A more attractive option is the implementation using a *pipeline architecture* approach.

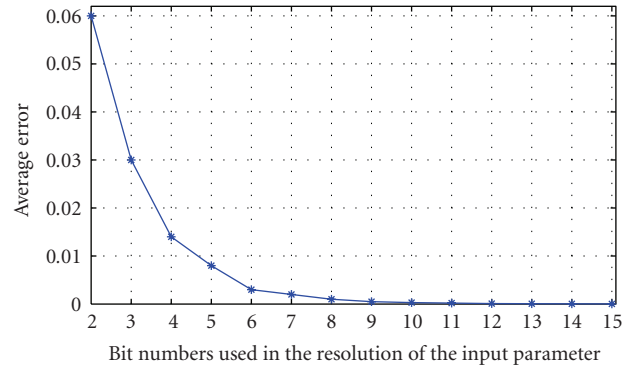

Figure 8: Average error as a function of the resolution used in A, B, C, U, V, y, W.

The 3D Recovery Module has A, B, C, U, V, W, x, y and motion vectors data as input. In the first pipeline stage, the values α and β are calculated: these variables depend only on the inputs. Part of the u_r and v_r values are calculated in this stage too. The square values of α and β and other part of u_r and v_r are calculated in the second stage. In the third stage, the variables u_r, v_r and the equation numerator are calculated. The fourth stage computes the denominator of the equation. Finally, in the last stage, Z is obtained.

Figure 9 shows the pipeline stages of the design with each of the intermediate modules. The operations in each stage are performed in fixed-point arithmetic, with different lengths for the integer and fractional part. Small errors are introduced due to the limited fixed point precision used for data representation in processing elements of the architecture. Currently a quantitative estimation of the error is being performed avoiding the excessive use of hardware resources.

The depth value is obtained with a 9-bit representation. Z uses 0 bits for the integer and 9 bits for the fractional part. The graphic in Figure 8 shows the average error when representing fractional values, using variables with different resolutions. For each case, all the variables have the same resolution, and their representation is always in fixed-point arithmetic. All the bits in this representation are used for the fractional part of the variables. The values shown were obtained through hardware implementation experimentation.

In Figure 8, it can be appreciated that the average error drops as the resolution of the input variables is incremented. This reduction is not linear, and the graphic shows a point where such reduction is not significant, no matter the increment in the number of bits of the input variables. 9 bits were chosen as a good compromise between area and average error.

Table 1 shows the calculation of depth in each of the *pipeline* stages. Once the motion vectors have been computed, the process in which the value of the depth is obtained begins. The input values A, B, C, U, V, and W simulate a translational motion of an object in the direction of the X axis. The motion of the object is of one pixel per image in the simulation.

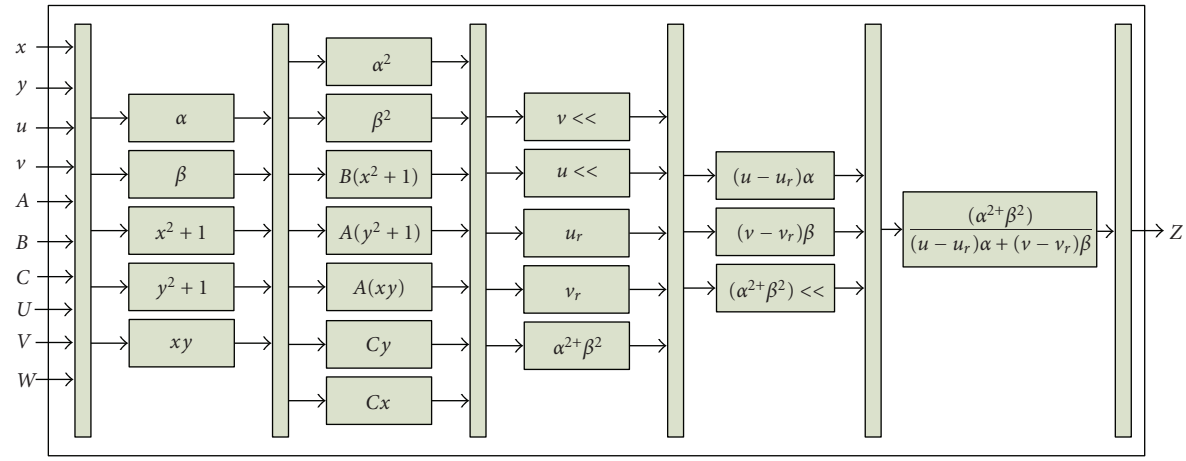

Figure 9: Architecture for the calculation of Z.

Table 1: Calculation of the depth in the different stages of the *pipeline*.

Parameters	Pipeline cycle	Stage 1		Stage 2		Stage 3		Stage 4		Stage 5	
$A = 0.00$		α	0.50								
$B = 0.00$		β	0.00								
$C = 0.00$	1	$y^2 + 1$	2.00								
$U = 0.50$		$x^2 + 1$	2.00								
$V = 0.00$		y^*x	1.00								
$W = 0.00$				α^2	0.25						
$x = 1.00$				β^2	0.00						
$y = 1.00$				$A^*(y^2 + 1)$	0.00						
$u = 1.00$	2			$B^*(x^2 + 1)$	0.00						
$v = 0.00$				$A^*(x^*y)$	0.00						
				$B^*(x^*y)$	0.00						
				C^*x	0.00						
				C^*y	0.00						
	3					u_r	0.00				
						v_r	0.00				
	4							$(u - u_r)\alpha$	0.50		
								$(v - v_r)\beta$	0.00		
										$\alpha^2 + \beta^2$	0.25
	5									$(u - u_r)\alpha+(v - v_r)\beta$	0.50
										z	0.50

4.4. Routers. The function of the Router units is to generate the addresses for the data read and write operations. To avoid a huge number of memory accesses, the routers store several rows from the images before the execution of any other action regarding the external memory. The *In-Router* module is composed by 12 buffers that store the rows of the images.

The block works as follows: eight rows from the current image (frame 1) are read and stored (search window). Then, four rows from the previous image (frame 0) are also read and stored (reference window). These pixel rows are stored in independent internal RAM memories (buffers). The router feeds 12 pixels in parallel to the optical flow module. When a full search window has been processed, a new pixel is read from the external memory and then stored in the last buffer. Each pixel of the actual address is transferred to the past buffer. The generation of the new read addresses is performed at the same time. The functional block diagram with input and output variables can be seen in Figure 10.

The *OUT-Router* (Figure 11) performs the writing of the architecture results to the external RAM memory. This block is simpler than the *In-Router* and is composed by an address generator and a register.

The *Gen_Addr_Esc* module controls the storage address of the datum corresponding to the depth, obtained in the *Depth calculation* module. The Register module put together 4 depth values calculated by the architecture in order to align them for memory write. This concatenation has the

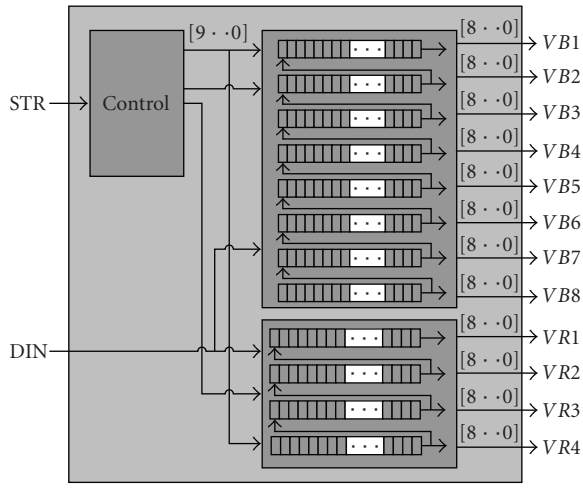

Figure 10: *IN-Router* block diagram.

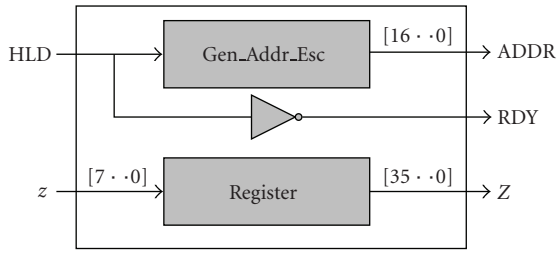

Figure 11: Component of the *OUT-Router* module.

purpose of storing a 36-bit value in each of the RAM locations.

4.5. Final Architecture. The final version of the architecture works as follows: in the first step, a pair of images from a sensor or another process is stored in the external memory. Next, a data array of the two images is read by the *In-Router* module: 8 rows of the previous image and 4 rows of the actual one. Once this is done, the data stored in the module are addressed to the WPM elements blocks, for the motion estimation. After being calculated, the motion vectors are passed to the next module for the depth calculation. In a parallel fashion, a new datum of the actual image and another of the reference image are acquired, and the process of the motion estimation is started.

After the computation of the depth has been performed, the result is stored in the external memory, where the system waits for the motion vectors for performing the process once again. This is repeated until the two images have been completely processed. When this is finished, a new pair of images is stored in the external memory.

5. Performance Analysis

From the general description of the architecture, an estimation of the performance of the architecture can be obtained. The processing speed of the architecture can be estimated as a function of the size of the image, the number of necessary cycles for the processing of a reference window of $n \times n$, the number of PEs operating in parallel, and the number of times the WPM are instantiated in the complete architecture. The analysis is based on the critical path which is the slowest module (the *Optical Flow Calculation*).

The number of clock cycles that are necessary to process a row of the search window of $m \times m$ pixels with the reference window of $n \times n$ pixels is given by the number of cycles required to process a row of the reference window plus the number of cycles that would take to process the positions that the reference window occupies over the search window, in a horizontal way. In order to compute the number of cycles required to process a row of the search window, the following is used:

$$\text{cycles}_{\times\text{row}} = \text{cycles}_{\times Vr} + \frac{\left(\text{cycles}_{\times Vr}\right)\left(V_{r\_\text{pos\_hor}}\right)}{V_{r\_\text{proc\_par}}}, \quad (5)$$

where $\text{cycles}_{\times\text{row}}$ is the number of cycles to process a row of the search window, $\text{cycles}_{\times Vr}$ is the number of cycles to process a row of the reference window, $V_{r\_\text{pos\_hor}}$ is the number of positions that the reference window occupies over the search window, in the horizontal direction, and $V_{r\_\text{proc\_par}}$ is the number of windows processed in parallel.

The number of cycles that are necessary to process a search window is the size of the reference window n, multiplied by the number of cycles required in processing a row of the search window by the number of positions occupied by the reference window above the search window in the vertical direction, divided by the number of processors that work in parallel and the number of times that the PE blocks array is repeated. Once the processing of the search window is done, two data are read from the external memory, so this will add 2 more clock cycles. Equation (6) allows the calculation of the number of cycles necessary to process the search window:

$$\text{cycles}_{\times V_b} = \frac{n\left(\text{cycles}_{\times\text{row}}\right)\left(V_{r\_\text{pos\_ver}}\right)}{\left(\text{PE}_{\text{par}}\right)\left(\text{WPM}_{\text{blocks}}\right)} + \text{cycles}_{\text{read}}, \quad (6)$$

where $\text{cycles}_{\times V_b}$ is the number of cycles required to process a search window, n is the size of the search window, $\text{cycles}_{\times\text{row}}$ is the number of cycles required to process a row of the reference window, $V_{r\_\text{pos\_ver}}$ is the number of positions of the reference window over the search window, PE_{par} is the number of processing elements working in parallel, and $\text{WPM}_{\text{blocks}}$ is the number of times that the PE array is repeated.

Finally, (7) represents the total number of cycles necessary to process a full image. This total is calculated multiplying the number of cycles required to process the search window by the number of search windows present in the image, both in a horizontal and in a vertical way,

$$\begin{aligned}\text{cycles}_{\times\text{Imag}} = {} & \left(\text{cycles}_{\times V_b}\right)\left(\text{Img}_{\text{hor}} - m + 1\right) \\ & \times \left(\text{Img}_{\text{ver}} - m + 1\right).\end{aligned} \quad (7)$$

Table 2: Synthesis results for the *Optical Flow Calculation* module.

Resources	Usage
FFs	1,986 of 10,944
LUTs	5,192 of 10,944
Slices	3,519 of 5,472
Max. Operating Freq.	70 MHz

Table 3: Synthesis results for the *Depth Calculation* module.

Resources	Usage
FFs	68 of 10,944
LUTs	1,882 of 10,944
Slices	983 of 5,472
Max. Operating Freq.	100 MHz

Table 4: Synthesis results for the full architecture.

Resources	Usage
FFs	2,177 of 10,944
LUTs	8,024 of 10,944
Slices	4,739 of 5,472
Block Rams	12 of 36
Max. Operating Freq.	66 MHz

For validation in this work, the following values were used:

(i) $n = 4$,
(ii) $m = 8$,
(iii) $\text{cycles}_{\times Vr} = 4$ cycles,
(iv) $V_{r\_\text{pos\_hor}} = m - n = 8 - 4 = 4$ positions,
(v) $V_{r\_\text{proc\_par}} = 4$ windows,
(vi) $V_{r\_\text{pos\_ver}} = m - n + 1 = 8 - 4 + 1 = 5$ positions,
(vii) $\text{PE}_{\text{par}} = 4$ PE blocks working in parallel,
(viii) $\text{WPM}_{\text{blocks}} = 5$ blocks of 20 PEs,
(ix) $\text{Img}_{\text{hor}} = 640$,
(x) $\text{Img}_{\text{ver}} = 480$.

Replacing the values of $\text{cycles}_{\times Vr}$, $V_{r\_\text{pos\_hor}}$, and $V_{r\_\text{proc\_par}}$ in (5), the following equation is obtained:

$$\text{cycles}_{\times V_{\text{row}}} = 4 + \frac{(4)(4)}{4} = 8\,\text{cycles}_{\times \text{row}}. \tag{8}$$

Now the values of n, $\text{cycles}_{\times V_b}$, $V_{r\_\text{pos\_ver}}$, $\text{WPM}_{\text{blocks}}$, and PE_{par} are replaced in (6) to obtain the number of cycles necessary to process the search window

$$\text{cycles}_{\times V_b} = \frac{4(8)(5)}{(4)(5)} + 2 = 10\,\text{cycles}\ p/Vb. \tag{9}$$

To obtain the total number of cycles necessary to process an image, the values of m, $\text{cycles}_{\times V_b}$, Img_{hor}, and Img_{ver} are replaced in (7)

$$\begin{aligned}\text{cycles}_{\times \text{Imag}} &= (10)(640 - 8)(480 - 8) \\ &= 2,994,090\,\text{cycles}\ p/\text{frame}.\end{aligned} \tag{10}$$

Table 5: Percentage of consumed resources of the FPGA device, by the modules of the proposed architecture.

Module	% resources
Optical Flow Calculation module	47
Depth Calculation Module	17
Routers and logic	9
Complete Architecture	73

Figure 12: Image from the used sequence.

For example, with a clock frequency of 50 MHz, the architecture could process 16 fps for a 640 × 480 pixels resolution image stream.

6. Implementation and Results

6.1. Architecture Implementation and Synthesis. The hardware design was described in Handel-C 4, the design was simulated in MatLab 7, and the synthesis design was carried on with Xilinx ISE Project Navigator 9.1. The board used for testing was an ML403 from Xilinx. The ML403 integrates a XC4VFX12 FPGA of the Virtex 4 family. The memory ZBT RAM was used to store the image.

Table 2 shows the consumption of hardware resources used by the *Optical Flow Calculation* module. Table 3 shows the use of resources of the *Depth Calculation* module. Table 4 refers to the resources usage of the full architecture, and finally Table 5 shows the percentage of resources used the modules and the complete architecture.

6.2. Results. An image sequence of a soda can was used to test the architecture (Figure 12). The sequence simulates the translational movement of the object on X axis by 1 pixel per frame.

Figure 13 shows the results of the optical flow obtained by the hardware module using the images sequence.

In the computation of optical flow, to try to summarize the resulting quality of millions of motion vectors as only a number is a complicated task, so several metrics were

Table 6: Comparison performance of the proposed architecture with other works.

Work	FPS	Resolution	Implement platform	Processed pixels/sec
Fife and Archibald [5]	30	320 × 240	FPGA	2304000
Diaz et al. [6]	0.05	640 × 480	FPGA	15630
Zach et al. [7]	2.46	640 × 480	HW/SW	755712
Proposed architecture	22	640 × 480	FPGA	6758400

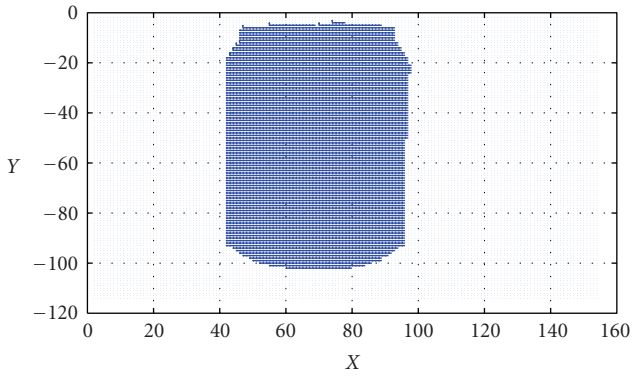

Figure 13: Optical flow calculated for the sequence.

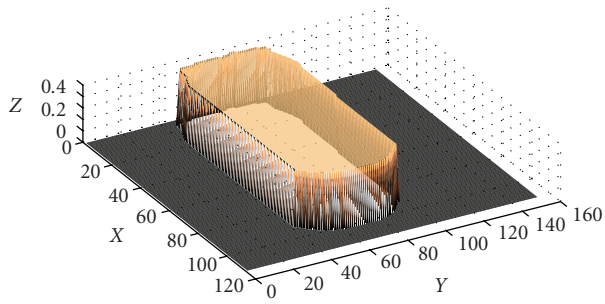

Figure 14: 3D recovery from optical flow for the can images.

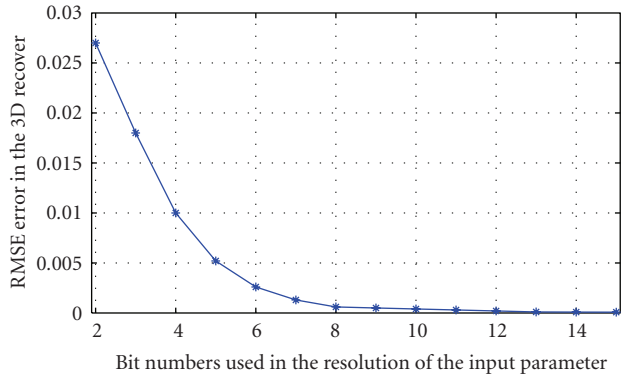

Figure 15: RMSE error in the 3D recovery, as a function of the resolution of the input variables A, B, C, U, V, y, W.

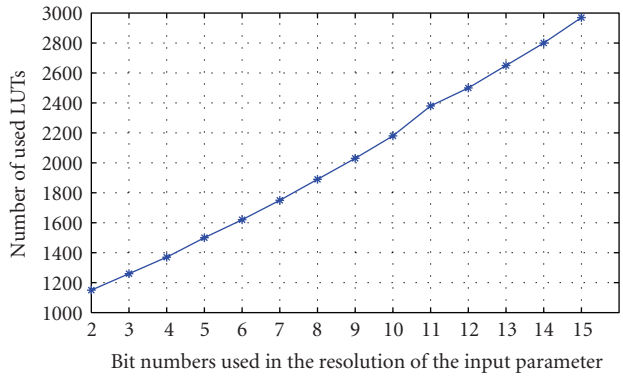

Figure 16: Number of used LUTs against the resolution of the input variables A, B, C, U, V, y, W.

evaluated. The first one is the error between the angle of the vectors obtained by software simulation and by the architecture [12]. The 100% of the vectors obtained by the architecture are correct with respect to those of the software implementation.

Figure 14 shows the results obtained by the processor for the 3D recovery.

Figure 15 shows the error obtained in the calculation of the depth against the resolution of the input variables. As in the graphic of Figure 8, the curve decreases quickly for the first values and then it stabilizes. At this point, the data representation precision increment has a little effect in the reduction of the error when calculating the 3D recovery.

Finally, the graphic in Figure 16 depicts the variation in the number of used LUTs of the device against the resolution of the input variables. Contrary to the last two graphics, this one presents an almost linear behavior. The consumption of resources grows as the number of bits used in the variables is incremented.

From the graphics, the number of bits for the input variables can be selected. In Figures 8 and 15, it can be seen that a resolution of 8 bits gives good results. Moreover, the amount of resources used with this resolution is moderated. It can also be seen that, when incrementing the number of bits in more than 8, the reduction in the error of the calculation of the 3D recovery is minimum. As a result of these points, the selected resolution for the 3D recovery based on the optical flow was 8 bits.

To measure the quality of the depth recovery, the RMSE metric was used. The average error given was of 0.00105 for several performed recoveries.

6.3. Discussion. The performance of the architecture is given as a function of the number of processed images, the number of operations performed in one second, and the number of

computed depth values. The following show a quantitative analysis of the architecture.

The processing time of the images in the architecture is conditioned by the maximum operating frequency, which is established as a function of the delays of the combinational logic, the way in which the clock signal is distributed, and the internal routing of the device. In the specific case of the implemented architecture, the maximum operating frequency is 66 MHz, which allows the processing of 22 frames per second, operating with 640×480 images. The architecture has the capacity of processing 4, 915, 200 depth values per second, with an average error of 0.00105.

Once the number of images per second that the design can process is known, the number of operations per second (OPS) performed by the architecture can be calculated. The OPS is obtained by multiplying the number of fps, the number of operations of the search window, and the amount of search windows in the image

$$\begin{aligned} \mathrm{OPS}_{\mathrm{SAD}} &= \mathrm{fps} * \left(\#\mathrm{operations}_{V_b}\right) \\ &\quad * \left(\mathrm{Img}_{\mathrm{hor}} - m + 1\right) * \left(\mathrm{Img}_v er - m + 1\right), \\ \mathrm{OPS}_{\mathrm{SAD}} &= 16 * (3 * 16 * 25) * (640 - 8 + 1) \\ &\quad * (480 - 8 + 1) = 5.748 \times 10^9. \end{aligned} \tag{11}$$

For the 3D recovery from the optical flow, the number of operations is obtained by multiplying the number of frames per second, the number of operations necessary to calculate a single depth value, and the number of motion vectors calculated for the image. Equation (12) allows the calculation of the number of performed operations that have to be completed for the 3D recovery through optical flow

$$\begin{aligned} \mathrm{OPS}_Z &= \mathrm{fps} * (\#\mathrm{operations}_Z) \\ &\quad * \left(\mathrm{Img}_{\mathrm{hor}} - m + 1\right)\left(\mathrm{Img}_v er - m + 1\right), \\ \mathrm{OPS}_Z &= 16(32)(640 - 8 + 1) \\ &\quad * (480 - 8 + 1) = 153.297 \times 10^6. \end{aligned} \tag{12}$$

The architecture performs 7.904 GOPS (Giga Operations per Second) in an integer representation for the optical flow, and 210 millions OPS in fixed-point representation for the 3D recovery. Thus, the architecture performs a total of 8.115 GOPS during the full 3D recovery process.

Our results compares favorably (see Table 6) with other implementations.

7. Conclusions and Future Work

The present work has discussed a parallel processor for the 3D recovery through Optical Flow inside a video sequence with real-time restrictions. The designs exhibit a balance between area utilization and processing speed. The implementation is capable of obtaining the optical flow from image sequences with VGA resolution in a predictable time, as it can process 22 fps.

It is possible to scale the proposed design so it can operate over the 30 fps or work with higher resolutions. This is performed by adding the necessary Optical Flow modules to process more search windows in a parallel fashion. In this way it could be possible to exploit the overlap of the search windows.

The computational load to perform the 3D recovery is of about 8 GOPS, which is difficult to perform in a short period (in the order of the milliseconds) with current sequential processors.

The architecture presents a small size, it is possible to implement it in systems where the space restrictions and the power consumption are the main concern, as in the case of mobile vision systems and in robotic vision.

Some points regarding future work are the following.

(i) Test different reference and search window sizes and analyze the results.

(ii) Analyze other algorithms for optical flow and their adaptation to the proposed architecture. The reuse of the architecture modules would imply minimum changes and a small or even null increment in the complexity of the proposed architecture.

(iii) Incorporate predictive algorithms and their hardware implementation to achieve a better 3D recovery.

Acknowledgment

The first author thanks the National Council for Science and Technology from Mexico (CONACyT) for financial support through the M.Sc. scholarship no. 206814.

References

[1] R. Koch and L. J. Van Gool, *3D Structure from Multiple Images of Large-Scale Environments*, Springer, London, UK, 1998.

[2] G. P. Martisanz and J. M. de la Cruz Garcia, *Vision por Computador: Imagenes Digitales y Aplicaciones*, Ra-Ma, Madrid, Spain, 2001.

[3] V. H. Rosales, *Recuperacion 3D de la estructura de una escena a partir del Flujo Optico*, M.S. thesis, INAOE, Puebla, Mexico, 2003.

[4] H. G. Nguyen, "Obtaining range from visual motion using local image derivatives," Technical Document 2918, RDT&E Division, Naval Command, Control and Ocean Surveillance Center, San Diego, Calif, USA, 1996.

[5] W. S. Fife and J. K. Archibald, "Reconfigurable on-board vision processing for small autonomous vehicles," *Eurasip Journal of Embedded Systems*, vol. 2007, Article ID 80141, 14 pages, 2007.

[6] C. Diaz, L. Lopez, M. Arias, C. Feregrino, and R. Cumplido, *Taller de Computo Reconfigurable y FPGAs, Implementacion FPGA del Calculo de Profundidades en la Recuperacion 3D Usando luz Estructurada*, Encuentro Nacional de Computacion, Apizaco, Mexico, 2003.

[7] C. Zach, A. Klaus, B. Reitinger, and K. Karner, "Optimized stereo reconstruction using 3D graphics hardware," in *Proceedings of the Vision, Modeling, and Visualization Conference (VMV '03)*, pp. 119–126, München, Germany, November 2003.

[8] A. Burton and J. Radford, *Thinking in Perspective: Critical Essays in the Study of Thought Processes*, Routledge, London,UK, 1978.

[9] D. H. Warren and E. Strelow, *Electronic Spatial Sensing for the Blind: Contributions from Perception*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1985.

[10] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[11] A. Hernandez, *Recuperacion en Tiempo Real del Flujo Optico de una Secuencia de Video usando una arquitectura FPGA*, M.S. thesis, INAOE, Puebla, Mexico, 2007.

[12] B. McCane, K. Novins, D. Crannitch, and B. Galvin, "On benchmarking optical flow," *Computer Vision and Image Understanding*, vol. 84, no. 1, pp. 126–143, 2001.

[13] K. P. Horn and B. G. Rhunck, "Determining optical flow," Tech. Rep. A.I. Nemo 572, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass, USA, 1980.

[14] P. Kunh, *Algorithms, Complexity Analysis and VLSI Architectura for MPEG-4 Mostion Estimation*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[15] A. R. Bruss and B. K. P. Horn, "Passive navigation," Tech. Rep. A.I. Nemo 662, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass, USA, 1981.