REAL-TIME GENERATION OF THREE-DIMENSIONAL MOTION FIELDS

Hiroaki Niitsuma and Tsutomu Maruyama

Systems and Information Engineering, University of Tsukuba
1-1-1 Ten-ou-dai Tsukuba Ibaraki 305-8573 JAPAN
niitsuma@darwin.esys.tsukuba.ac.jp

ABSTRACT

In this paper, we describe a compact system for real-time generation of three-dimensional motion fields. Our system consists of one FPGA, two cameras and one host processor. With our system, we can generate dense three-dimensional motion fields (640×480 vectors in a standard size image) at video-rate from dense optical flow and dense depth map obtained by area-based matching. The performance can be improved up to 840 frames per second in small size (320×240) images by configuring another circuit, though it requires more amount of hardware resources. By changing search spaces for optical flow and depth map by reconfiguration, we can control the maximum motion speed which can be detected, and the minimum distance to moving objects in the image, under limited hardware resources.

1. INTRODUCTION

In this paper, we describe a compact system for real-time generation of three-dimensional motion fields using optical flow and depth map. Optical flow is a two-dimensional vector field in the image plane, which is obtained by comparing two images at t-Δt and t taken by one camera. Optical flow can be used as the projection of the three-dimensional motion of the world in general. Depth map is a distance map to objects in the image, which is obtained by comparing two images (left and right) at t taken by two cameras (*stereo vision*).

In our system, in order to suppress noises using redundancy, dense three-dimensional motion fields are generated from dense optical flow and dense depth map obtained by area-based matching. In the area-based matching, the most similar parts to small windows ($w \times w$ pixels) in one image are looked up in the target area in another image using the SAD (Sum of Absolute Difference) algorithm. We implemented two kinds of circuits for the area-based matching. In the first implementation, intermediate results in the computation along x and y axes are stored on the FPGA and reused w times (but part of them are recalculated in order to minimize the amount of data which have to be stored on the FPGA) in order to achieve highest performance. This implementation achieved 840 frames per second (fps) on small size images (320×240 vectors in a image), which drastically outperforms previous works. In the second implementation, intermediate results along x axis are stored, but operations along y axis are re-executed w times in order to minimize the circuit size while maintaining video-rate on standard size images (640×480 vectors in a image).

The size of circuits for optical flow and depth map is almost proportional to the size of target area, though the area is two dimensional in optical flow and one dimensional in stereo vision (under epipolar constraint). With larger area, faster movement can be detected in optical flow, and closer objects can be found in stereo vision. However, under limited hardware resources, it is difficult to provide enough area for both of optical flow and stereo vision. By changing the area sizes by reconfiguration, we can control the maximum motion speed which can be detected, and the minimum distance to moving objects, depending on situations.

2. PREVIOUS WORKS

Computing the three dimensional motion is a fundamental task in computer vision, and many approaches using optical flow and depth map such as [2, 3, 4] have been proposed. Our approach is not new, but a brute force approach using dense vector field and dense vector map to suppress noises using redundancy, which becomes possible because of recent progress of FPGAs.

Many approaches to reduce the computational complexity of the optical flow have been proposed[5][6], but in those algorithms, computations of areas which seem to be unnecessary for detecting moving objects are not executed, and users need to think of trade-offs between accuracy and efficiency. In order to accelerate its performance by hardware, many systems have been proposed to date[7][8][9][10]. In those systems, in order to achieve real-time processing, sizes of images are limited or only sparse vector fields are generated. Their performances are, however, still slower than video-rate in the standard size images.

On the other hand, recent progress has already made it possible to generate dense depth map based on area-based matching[11, 12].

3. THREE-DIMENSIONAL MOTION FIELDS

Three-dimensional motion is projected onto a two-dimensional velocity field on the image plane of the camera. If vectors in optical flow are regarded to be equivalent to the velocity vectors[1], start and end points of a vector in optical flow correspond to start and end points of a three dimensional velocity vector. Suppose that

- (X, Y, Z) is the coordinate system positioned to the optical center of a camera, so that the Z axis coincides the optical axis, and (x, y) is the coordinate system on the image plane,
- correspondence between two images (t and t - Δt) has been calculated, and (x_c, y_c) at t corresponds to (x_p, y_p) at t - Δt,
- depth(x_c, y_c) at t is Z_c, and depth(x_p, y_p) at t - Δt is Z_p.

Then, $(Z_c x_c/f, Z_c y_c/f, Z_c)$ at t corresponds to $(Z_p x_p/f, Z_p y_p/f, Z_p)$ at t - Δt in 3-D space (f is the focal length). Therefore, the velocity vector (motion vector) can be expressed as

$$(V_X, V_Y, V_Z) = ((Z_p x_p - Z_c x_c)/f, (Z_p y_p - Z_c y_c)/f, Z_p - Z_c)$$

In our implementation, vectors from t to t- Δt are calculated in optical flow. By calculating these reversed vectors, we can obtain three-dimensional motion vectors on pipelined circuits as follows.

1. Circuits for optical flow and depth map are almost same. Therefore, corresponding points to a given point (x_c, y_c) are searched in parallel on the two circuits.
2. Then, depth(x_c, y_c) at t are calculated using the distance between the corresponding points.
3. At the same time, depth(x_p, y_p) at t - Δt are obtained by reading out depth map at t - Δt stored in external memory on the FPGA board.

Suppose that the camera is moving with translational motion $\overrightarrow{T} = (T_X, T_Y, T_Z)$ and rotational motion $\overrightarrow{\omega} = (\omega_X, \omega_Y, \omega_Z)$. Then, two dimensional velocity (d_x, d_y) on the image plane can be represented as follows (suppose that the focal length is normalized to 1 to simplify the equations).

$$d_x = (-T_X + xT_Z)/Z + \omega_X xy - \omega_Y(x^2+1) - \omega_Z y$$
$$d_y = (-T_Y + yT_Z)/Z + \omega_X(y^2+1) - \omega_Y xy + \omega_Z x$$

If cameras are mounted horizontally on a vehicle with non-holonomic kinematics, we can consider $T_X, T_Y, \omega_X, \omega_Z \equiv 0$. Then, the general equations above can be reduced to the following:

$$d_x = xT_Z/Z - \omega_Y(x^2+1)$$
$$d_y = yT_Z/Z - \omega_Y xy$$

These equations can be solved for each vector as follows.

$$T_Z = (d_y(x^2+1)/y - d_x x)Z$$
$$\omega_Y = -d_x + d_y x/y$$

By calculating T_Z and ω_Y using (d_x, d_y) obtained by optical flow, and Z (distance) obtained by stereo vision, we can estimate T_Z and ω_Y. In our current implementation, values of T_Z and ω_Y for all vectors are calculated on the FPGA, and the *medians* of them are calculated on the host processor to estimate the ego-motion [4]

4. THE OPTICAL FLOW

In an image taken by a camera, each pixel corresponds to the intensity value obtained by the projection of an object in 3-D space onto the image plane. When the object or the camera moves, its corresponding projection also changes position in the image plane. Optical flow is a vector field that shows the direction and magnitude of these intensity changes from one image to the other. In the optical flow, the corresponding point to a given point in an image is searched in the next image taken by the same camera. Area-based (or correlation-based) algorithms match small windows centered at a given pixel to find corresponding points between the two images. They yield dense maps, but fail within occluded areas (occlusions are caused by the movement of the camera). Feature-based algorithms match local cues (e.g., edges, lines, corners) and can provide robust, but sparse maps which requires interpolation. In hardware systems, area-based algorithms are widely used, because the operations required in those algorithms are very regular and simple.

The most common pixel-based matching algorithm is squared intensity differences (SSD) and absolute intensity differences (SAD). We used the SAD (Sum of Absolute Difference) algorithm because it is the simplest, and its result is almost same as other algorithms in the stereo vision. In the SAD algorithm for the optical flow, ξ and η which minimize the following equation are searched.

$$SAD(x, y, \xi, \eta) =$$
$$\sum_{i=-w/2}^{w/2} \sum_{j=-w/2}^{w/2} |I_0(x+i, y+j) - I_1(x+i+\xi, y+j+\eta)|$$

In this equation, I_0 and I_1 are images in $time = t$ and $time = t + \Delta t$ respectively, and $w \times w$ is the size of the window centered at a given pixel (its position is (x, y)). The range of ξ and η decides the size of area where the corresponding point to (x, y) is searched.

In Figure 1, a small window centered at (x, y) (Figure 1(a)) is compared with all windows in its target area centered at (x, y) (Figure 1(b)(c)(d)). When the size of the target area is $(k+w-1) \times (k+w-1)$, there are $k \times k$ windows in the target area, and $k \times k$ SADs (Sum of Absolute Differences) are calculated. Then, the window which gives the minimum SAD is chosen, and its center point (x', y') is considered as the corresponding point to (x, y). In this comparison, every pixel in the window in $time = t$ is compared with $k \times k$ pixels in the target area (the range of ξ and η is $-k/2$ to $k/2$).

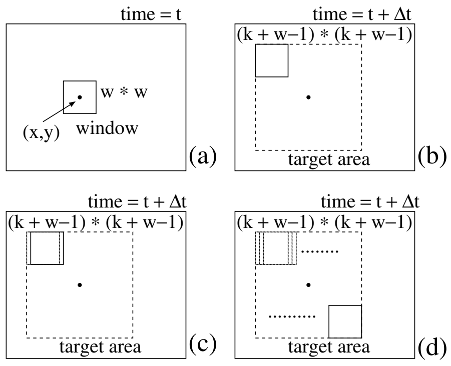

Fig. 1. Area-Based Matching in the Optical Flow

By this two-dimensional search, we can obtain one vector from (x, y) to (x', y').

4.1. A Technique to Realize the Maximum Performance

In order to achieve maximum performance on hardware, all operations have to be processed in parallel and in pipeline. Therefore, suppose that all operations described in this subsection are executed in parallel and in pipeline.

In Figure 2(a), suppose that we have calculated $k \times k$ SADs ($k \times k \times w \times w$ ADs (Absolute Differences) have been calculated) and chosen the minimum of them to obtain one vector (computations of only two SADs are shown to simplify the figure). During these computations, no operations on same data are executed.

Then, the window is shifted to the right by one pixel to obtain next vector (Figure 2(b)). At this point of time, pixels in a rectangle with slanting lines in the shifted window ($time = t$) are already compared with pixels in rectangles with slanting lines in its target area ($time = t + \Delta t$) during the computation of the previous vector. Therefore, by storing $k \times k \times (w - 1) \times w$ ADs (Absolute Differences) calculated in Figure 2(a), the number of new ADs to obtain the new vector can be reduced to $k \times k \times w$. When the window is shifted down by one pixel as shown in Figure 2(c), pixels in a rectangle with slanting lines in the shifted window are already compared with pixels in rectangles with slanting lines in its target area. In this case, we can also reduce the number of AD operations to $k \times k \times w$ by storing and reusing the $k \times k \times w \times (w - 1)$ ADs.

In Figure 2(d), suppose that the image size is $X \times Y$ and the window is shifted to the right (along x axis) first, and when the window reaches to the right-end of the image, the window is moved to the left-end again and shifted down by one pixel. In this case, when the window is shifted to the right by one pixel, ADs for $w \times w - 1$ pixels in the shifted window (all pixels in the window except for one pixel shown by a black dot) are already calculated ($k \times k \times (w \times w - 1)$ ADs are already calculated) during the computation of previous vectors. Therefore, by storing $k \times k \times (w - 1) \times X$

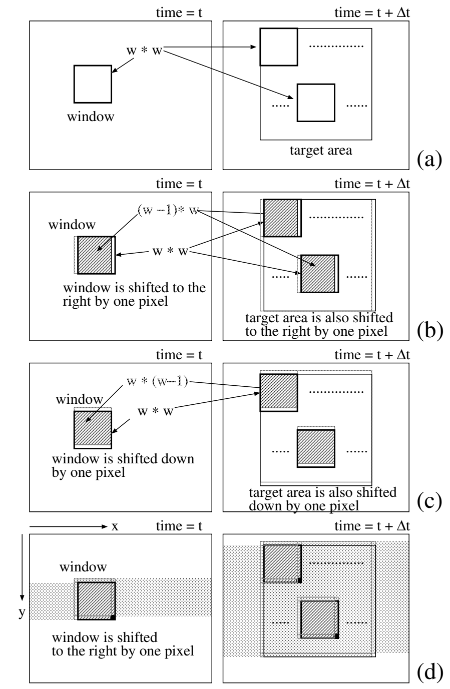
time = t
time = t + Δt
w * w
window
target area
(a)
(w –1)* w
window is shifted to the right by one pixel
target area is also shifted to the right by one pixel
(b)
w * (w–1)
window is shifted down by one pixel
target area is also shifted down by one pixel
(c)
x
y
window is shifted to the right by one pixel
(d)

Fig. 2. Reuse of the Intermediate Results

ADs (which correspond to the gray area in Figure 2(d)), we can calculate $k \times k$ SADs which are necessary to obtain a new vector by only calculating $k \times k$ new ADs (ADs between the pixel shown by the black dot and $k \times k$ pixels in the target area). In this computation, we need to access $k \times k \times (w \times w - 1)$ ADs (which are already calculated and stored) in parallel in order to achieve maximum performance.

Figure 3 shows an implementation technique to make the parallel access possible (the upper half of the figure shows the two images which are compared, and the lower half of the figure shows an array of SAD units, and the inside of a SAD unit). In Figure 3, suppose that the vector for a pixel (light gray square in the window) was just obtained, and the window is shifted to the right to find the vector for the next pixel (dark gray square). Then, the window is compared with $k \times k$ windows in its target area ($k \times k$ SADs are calculated), and the minimum SAD is searched. In order to achieve maximum performance, $k \times k$ SAD units are prepared and the $k \times k$ SADs are calculated in parallel (In Figure 3, only two units are shown to simplify the figure). In Figure 3, $A_{i,j}$ are ADs (Absolute Differences) which are already calculated during the computation of previous vectors. In Figure 3, a new SAD is calculated using $A_{i,j}$ as follows.

1. w-1 ADs ($A_{i,6} (i = 2, 5)$ (squares with sparse slanting lines)) are read out from memory M_A.
2. A new AD for the black square (which becomes $A_{6,6}$) is calculated (pixel data of the black square (I_4) is broadcasted to all SAD Units on the array).

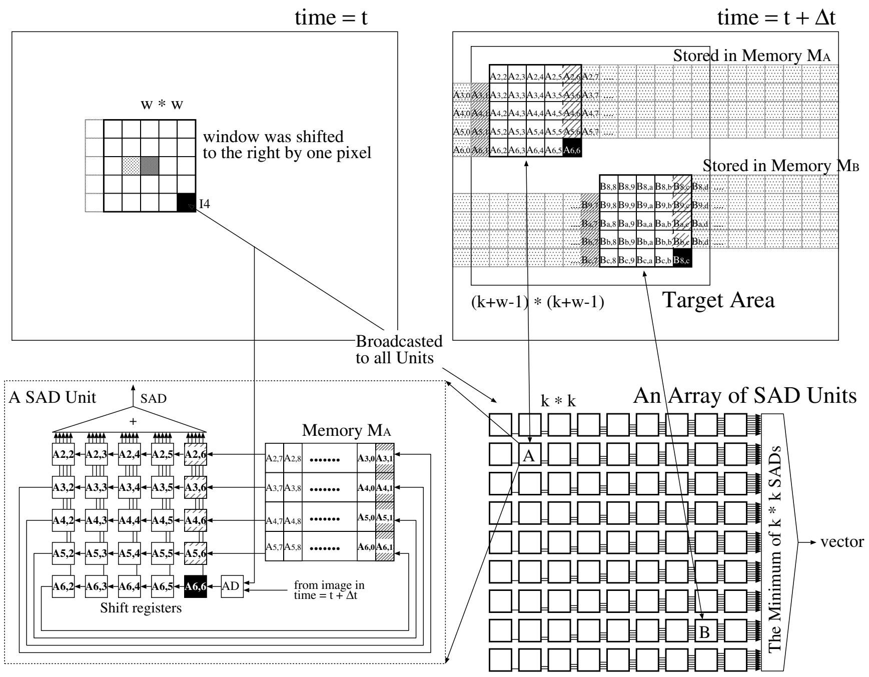

Fig. 3. An Implementation Technique to Achieve Maximum Performance

3. These w ADs are held on w shift registers in the SAD unit. Each shift register can hold w ADs (w is 5 in Figure 3). Thus, $w \times w$ ADs are on the shift registers in total. The ADs on the shift registers are shifted when a new SAD (consequently a new vector) is obtained.

4. These $w \times w$ ADs on the shift registers are summed up to calculate a new SAD.

5. Among w ADs which are shifted out from the shift registers, w-1 ADs are written back to M_A ($A_{i,1}(i = 3, 6)$ (squares with dense slanting lines)). Thus, each AD is summed up w times while it is on the shift register, and is stored and read out from the memory w-1 times, which means each AD is used for calculating $w \times w$ SADs.

By repeating the procedure above with $k \times k$ SAD units which run in parallel and in pipeline, we can continue to obtain a new vector in every clock cycle.

In this implementation, the width of memory M_A must be w-1 words. Therefore, the total number of memory banks required in this implementation becomes $k \times k \times (w-1)$, and these memory banks must be accessed in parallel. This means that these memory banks have to be located on the FPGA (because the input/output performance of LSIs (including FPGAs) is very limited). However, the number and width of internal memory banks of the latest LSIs are not enough under the practical w and k.

In order to reduce the number of memory banks, the procedure described above is modified as follows.

1. Only the sum of the w-1 ADs is stored in the memory (suppose that $\sum_{i=2}^{5} A_{i,6}$ is in the memory).

2. The sum is read out, and added with $A_{6,6}$ ($\sum_{i=2}^{6} A_{i,6}$ can be obtained).

3. w sums on a shift register are summed up to calculate new SAD ($\sum_{i=2}^{6} \sum_{j=2}^{6} A_{i,j}$).

4. At the same time, $A_{2,6}$ is calculated again, and subtracted from $\sum_{i=2}^{6} A_{i,6}$.

5. The result ($\sum_{i=3}^{6} A_{i,6}$) is stored in the memory to obtain vectors on the next row.

With this technique, we can reduce the number of memory banks to $k \times k$ from $k \times k \times (w-1)$, though we need double SAD units. The total hardware resources required by this technique are $k \times k \times 2$ SAD units, $k \times k$ memory banks and a unit to choose the minimum among $k \times k$ SADs in pipeline.

4.2. Video-Rate Processing

We can reduce the circuit size by recalculating some of ADs instead of storing all of them. This approach makes it possible to enlarge the size of the target area.

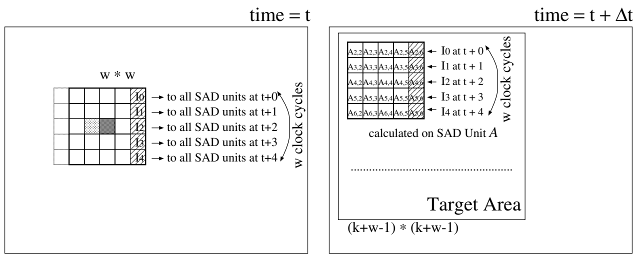

Fig. 4. An Implementation Method by Recalculation

By recalculating sums of ADs which were given by the memory banks ($\sum_{i=2}^{5} A_{i,6}$ in Figure 3), we can calculate SADs without memory banks. In Figure 4, I_0 is broadcasted to $k \times k$ SAD units first, and $k \times k$ ADs for I_0 ($|A_{2,6} - I_0|$ in Figure 3) are calculated in the $k \times k$ SAD units in parallel. In the same way, ADs for $I_j (j = 1, 4)$ are calculated sequentially. These calculations take w clock cycles in total (only one clock cycle when restored from memory banks). These ADs are, then, summed up, and held on the shift registers. The sums held on the shift registers are used w times to calculate w SADs and discarded after shifted w times.

Though this implementation requires w clock cycles to generate one vector, we can calculate vectors with $k \times k$ SAD units, no memory banks and a unit to choose the minimum among $k \times k$ SADs in pipeline. Furthermore, the size of the unit to choose the minimum SAD can be reduced to almost $1/w$, because many parts of the unit can be shared by w SAD units (each SAD unit generates one SAD in every w clock cycles).

The requirement for the video-rate processing is to obtain one vector in 108 nano seconds. Therefore, if we can build a circuit which runs faster than $108/w$ nano seconds, we can realize video-rate processing by this implementation method. The typical w used for the area-based matching is 7. Therefore, our goal is to build a circuit which runs faster than 65 MHz.

5. STEREO VISION

The area-based matching in the stereo vision is almost same with the matching in the optical flow. The only difference is that the size of the target area is reduced to $(d + w - 1) \times w$ from $(k + w - 1) \times (k + w - 1)$, though larger d becomes necessary (d is called *disparity*).

6. RESULTS

We implemented three kinds of circuits on Xilinx XC2V6000; two circuits for optical flow (*circuit-1* for the maximum performance and *circuit-2* for video-rate) and one circuit for generating motion fields at video-rate (*circuit-3*). All circuits run at 66 MHz. Table 1 shows the hardware usage

Table 1. Performance of the Circuits

		hardware usage		frame per second	
	(k, d)	BRAMs	slices	640×480	320×240
circuit-1	(11,0)	128	71%	210	840
circuit-2	(21,0)	16	84%	30	120
circuit-3	(17,91)	17	80%	30	120

and the performance of the three circuits. The computation time of the three circuits is promotional to the image size. The value of k in the circuit-1 (a circuit for optical flow at the maximum performance) is much smaller than the circuit-2 (the video-rate circuit for optical flow), because $k \times k \times 2$ SAD units are required in the circuit-1, and any part of the unit for choosing the minimum SAD can not be shared. Therefore, we could not implement a circuit to generate motion fields at maximum performance on XC2V6000 (we could not reduce k, which is already very small, to implement stereo vision together).

The maximum performance of our camera is 30 frames per second. Therefore, we could not demonstrate higher frame rates than that on our system, but we confirmed that our circuit can generate motion fields at the speeds which are shown in Table 1.

Figure 5 shows an example of the output by the circuit-3 (only a part of vectors are shown in the figure, because the image size is 640×480 and dense vector map is generated by the circuit). In Figure 5, a stuffed toy (tiger) was moved, and its movement along x and y axes is shown by arrows, and z axis by gradation. Some noises are found on areas with only small changes in the contexts. We need to add some circuits to suppress these noises using redundancy.

In order to detect faster motion, we need to enlarge the target area in the optical flow. As shown in Table 1, the maximum size which can be implemented on XC2V6000 is 21×21 pixels in the video-rate circuit. Therefore, the maximum displacement in the optical flow field is $(10, 10)$. This means that we can observe objects which traverse the image plane in 64 frames (about 2.1 seconds) when the image plane width is 640 pixels, and 32 frames (about 0.27 seconds if the cameras support 120 fps, otherwise 1.07 seconds) when the image plane width is 320 pixels. When both of optical flow and stereo vision are implemented, the maximum displacement is reduced to $(8, 8)$ (the minimum traverse time becomes 2.5 seconds and 0.33 (1.33) seconds). Therefore, a practical approach at video-rate is as follows.

1. Calculate optical flow and depth map on small size images.

2. When motions of particular interest are detected, stop moving.

3. By configuring other circuit, calculate only optical flow (or optical flow and depth map, depending on situa-

Fig. 5. The Output by the Circuit

tions) on large size images in order to trace the motions.

4. Then, configure the circuit to calculate optical flow and depth map for small size images again.

We have supported only the circuits shown in Table 1, and the image size is reduced by the host processor in the current implementation, though it should be done on the FPGA. We also need to prepare circuits which support rectangular target areas such as 41×11, because movement of moving objects is not upward, nor downward in most cases.

7. CONCLUSIONS

In this paper, we describe a compact system for real-time generation of three-dimensional motion fields. The system was implemented on an off-the-shelf PCI board with one FPGA. With our system, we can generate dense three-dimensional motion fields (640×480 vectors in a standard size image) at video-rate from dense optical flow and dense depth map obtained by area-based matching. The performance can be improved up to 840 frames per second in small size (320×240) images by configuring another circuit, though it requires more amount of hardware resources. Furthermore, by changing search spaces for optical flow and depth map by reconfiguration, we can control the maximum motion speed which can be detected, and the minimum distance to moving objects in the image, under limited hardware resources.

We are now improving the system to suppress noises using the redundancy in dense fields, and to work with edge detectors to clearly distinguish borders of the moving objects.

8. REFERENCES

[1] Adiv, G., "Determing Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects", IEEE Trans. on Pattern Analysis and Machine Intelligence, 7(4), 1985. pp. 319-336.

[2] Ballard, D.H., and Kimball, O.A., "Rigid Body Motion from Depth and Optic Flow", Computer Vision Graphics and Image Processing 22, No. 1, April 1983, pp. 95-115.

[3] Inoue, H., Tachikawa, T, and Inaba, M, "Robot Vision System with a Correlation Chip for Real-time Tracking, Optical Flow and Depth Map Generation", IEEE Int. Conf. on Robotics and Automation 1992, pp. 1621-1626.

[4] Stoffler, N.O., Schnepf, Z., "An MPEG-processor-based robot vision system for real-time detection of moving objects by a moving observer", International Conference on Pattern Recognition, 1998. pp. 477-481.

[5] Camus, T.A., "Real-Time Quantized Optical Flow", Workshop on Computer Architectures for Machine Perception 1995

[6] Zelek, J.S., "Bayesian Real-Time Optical Flow", Vision Interface 2002,

[7] Liu, H., Hong, T.H., Herman, M., Camus, T.A., Chellappa, R., "Accuracy vs. Efficiency Trade-Offs in Optical Flow Algorithms", Computer Vision and Image Understanding, 72(3), 1998, pp. 271-286

[8] P.C. Arribas, F.M.H. Macia, "FPGA Implementation of Camus Correlation Optical Flow Algorithm", Vision Interface 2001

[9] M. Fleury, A.F. Clark and A.C.Downton, "Evaluating optical-flow algorithms on a parallel machine", Image and Vision Computing, 19(3), 2001, pp. 131-143.

[10] Correia, M.V., Campilho, A.C., "Real-time implementation of an optical flow algorithm", International Conference on Pattern Recognition 2002, pp. 247-250.

[11] M.Arias-Estrada, J.M.Xicotencatl, "Multiple Stereo Matching Using an Extended Architecture", FPL 2001, pp. 203-212.

[12] Y. MIYAJIMA, and T. MARUYAMA, "A Real-Time Stereo Vision System with FPGA", FPL 2003, pp. 448-457.

[13] H. Niitsuma and T. Maruyama, "Real-time Detection of Moving Objects", FPL 2004, pp. 1155-1157.