Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 6 (2014) 5–19

Development and Embedded Implementations of a Hardware-Efficient Optical Flow Detection Method

László BAKÓ,[1] Fearghal MORGAN,[2] Szabolcs HAJDÚ,[3]
Sándor-Tihamér BRASSAI,[3] Róbert MONI,[3] Călin ENĂCHESCU[4]

[1]Research Department, Petru Maior University of Tîrgu Mureş, Romania,
e-mail: lbako@ms.sapientia.ro
[2]Bio-Inspired Electronics and Reconfigurable Computing (BIRC) Research Group,
National University Ireland Galway, Ireland,
e-mail: fearghal.morgan@nuigalway.ie
[3]Electrical Engineering Department, Faculty of Technical and Human Sciences,
Sapientia Hungarian University of Transylvania, Romania,
e-mail: tiha@ms.sapientia.ro
[4] Department of Informatics, Faculty of Sciences and Letters,
Petru Maior University of Tîrgu Mureş, Romania,
e-mail: ecalin@upm.ro

Manuscript received September 29, 2015; revised November 25, 2015.

Abstract: The main goal of the proposed project is to enhance the capabilities of a wheeled or flying mobile robot with features like egomotion estimation and/or obstacle avoidance. This implies the implementation of vision-based navigation of robots using artificial vision, computed with on-board embedded hardware. The current paper aims to contribute on the implementation of a real-time motion extraction from a video feed using embedded FPGA circuits. An alternative implementation using a Raspberry Pi is also presented. A performance analysis is given with references to other works.

Keywords: motion extraction, optical flow, embedded implementation, real-time, FPGA circuit, VHDL, low-resource-cost, Raspberry Pi.

1. Scientific background

The optical flow calculation involves extracting a dense velocity field of an image sequence assuming that the intensity is preserved during the motion. This result may then be used for other applications, such as three-dimensional (3-D) reconstruction, time interpolation of image sequences, video compression, motion segmentation, tracking, robot navigation, and time to collision estimation. There are several ways to recover 3-D information from two-

The results of this study were partially presented at the 5th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics 2015.

dimensional images (2-D) using various signals. In this article we will describe implementation of a motion flow system in real time, with low resource cost properties. Optical flow algorithms are widely covered in the specific literature. Some authors have undertaken a comparative study of the accuracy of different approaches with synthetic sequences [1]. We have focused on a model of classical gradient based method of Lucas & Kanade (L & K) [2]. Several authors have emphasized satisfactory balance between precision and efficiency in this model, which is an important factor in deciding which model is best suited for use as a real-time processing system.

One of the most important choices at the design level of a vision system is the selection of the image acquisition hardware. For instance, there are alternatives to the cameras similar to vertebrate-like single-lens eyes, such as insect-like compound eyes [3, 4] that are developed by prestigious research groups. These offer a dynamically adaptable structure with panoramic field of view, low distortion and aberration, and good temporal resolution while yielding high spatial resolution alongside a reduced size. These properties are highly useful for visually-controlled navigation, specifically for tasks like take-off, landing, collision avoidance and other optically driven responses, which do not require a high resolution image acquisition. The local sensory adaptation capabilities of insect compound eyes can compensate for significant changes in light intensity at the photoreceptor level and distribute information in a neuronal circuitry, resulting in fast and low-power integrated signal processing.

The processing hardware support [5, 6, 7, 8, 9, 10] selection for implementing these artificial vision systems can be critical for a high value outcome [10, 11, 12, 13]. Being at the center of group's research activity, the new generations of SRAM-based FPGA devices are a proper choice for the implementation of reconfigurable computing platforms that need accelerated processing in real-time systems. On the other hand, the hardware-software co-design problem is more complex in system development because the components need to be more advanced. The requirements for runtime partial reconfiguration capability in embedded applications can be sustained by storing multiple bit-stream generation choices, including direct bit-stream manipulation for logic blocks and hybrid one-dimensional and two-dimensional physical area relocation control modules.

The main goal of the proposed project is to enhance a mobile robot with evolutionary optimization capabilities for tasks like ego motion estimation and/or obstacle avoidance. The robot will learn to navigate different environments and will adapt to changing conditions. Using the run-time reconfiguration properties of modern digital reconfigurable hardware-based (FPGA) platforms [12, 13, 14, 15], an otherwise days-long evolutionary cycle of a physical robot can be slashed to a matter of milliseconds. By implementing this technique, the most common issues that emerge when using evolutionary

simulation – modeling the real world environment [15, 16, 17] as accurately as possible and modeling only those characteristics of the robot that are relevant for achieving the desired behavior – are avoided.

2. The developed method for resource-efficient optical flow extraction

The first studies on optical flow computation date back to 1980 and there are many alternative methods offered. They can be based on gradient, correlation, energy and phase methods, creating well-defined groups [4]. Gradient methods are based on the evaluation of spatial-temporal derivatives. The first such methods are presented by Horn and Schunck [1], respectively Lucas and Kanade [2]. All these methods are difficult to implement in digital hardware, due to their high resource-cost.

If we represent the image with a matrix A, its values will represent the gray level of a point in the image. When representing a grayscale pixel on 8 bits, these values will vary between 0 and 255. In *Fig. 1* we can see a frame of test video sequence named GRID. The images corresponding to this and other sequences were used as inputs to the algorithm for calculating the Optical Flow (OF), developed using Visual C++.

The gradient in an image is a vector indicating the direction of variation of image intensity (grayscale variation direction). This can be determined by calculating the value difference of adjacent image points. Consider a new matrix B which contains the gradient values of the matrix A. Using the values adjacent to the pixel *p* in the image in the calculation of the gradient, will result in a properly aligned gradient. Detection of outliers in this gradient will then lead to the detection of edges in images. This method, however, is sensitive to noise and luminance variations. The effect of noise can be reduced by calculating the average values of the gradient in the orthogonal direction, too. A horizontal gradient used so far is made by calculating the difference between values of two columns.

$$B(j, k)=A(j, k+1) - A(j, k-1) \tag{1}$$

This can be represented as a filter matrix of the form

–1	0	1

where the values multiplied with the pixel gray-level values will determine the locations of sharp tone differences in the image.

In order to reduce noise sensitivity of the method we have studied the possibilities of determining the average value of the gradient calculated from the video images.

Vertical edges are obtained by averaging the three rows in this matrix:

–1	0	1
–1	0	1
–1	0	1

Similarly, averaged horizontal edge values may be obtained using a vertical mask of the form:

1	1	1
0	0	0
–1	–1	–1

The result of these operations will be placed at the location indexed by the central element of these matrices. In fact, these 3×3 mask matrices are a basic form in these types of applications, but can have many variations by changing the weighting of the cells. We have experimented two of the well known mask matrices in the literature, with which we run experiments. The first option is the one developed by Roberts and Sobel's is the second. These methods are effective, as demonstrated by the abundance of their applications in the literature. It is also important to mention, that these methods require fewer resources for implementation in digital hardware than other methods such as Canny, LoG (Laplacian of Gaussian), Prewitt, Frei-Chen.

A. Optical flow computation experiments with different video sequences as input data

In order to test and validate the algorithm developed and implemented at first in software, we chose three different video sequences. Two of them are real and the third video is an animation.

In the top left corner of *Fig. 1* we can see a frame from one of the video sequences used as test data, called Grid (31 frames with a resolution of 320×240, with 8 bits/pixel). These images were used as inputs to the algorithm for calculating the optical flow (Optical Flow - OF).

Examples of calculating the horizontal (top right) and vertical (bottom left) gradient of the Grid sequence of video frames can be seen in *Fig. 1*. The detection of horizontal and vertical edges is the next step performed, as the bottom right section of *Fig. 1* shows.

In *Fig. 2* we can see one frame of the test video sequence called *Anim* (51 frames with a resolution of 200×200, with 8 bits/pixel).

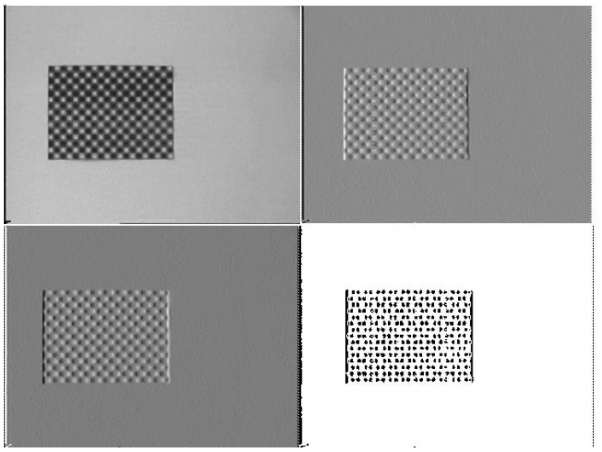

Figure 1: The GRID video sequence's frame, with computed horizontal, vertical gradients and detected combined edges.

We also have implemented a feature of the program to calculate the gradient direction obtained with the following trigonometric relationship:

$$\phi = \arctan\left(\frac{B_v(j,k)}{B_h(j,k)}\right) \tag{2}$$

After reading frames of the video sequence files, the first operation performed by the method's testing program is a Gaussian filtering with a filter matrix of 5×5 pixels. This first step is followed by the calculation of vertical, horizontal and combined gradients, with results stored separately. The algorithm continues with the positive and negative edge detection based on the frame intercorrelations, than it comes to determining the optical flow.

The effort invested in writing this software without the use of existing function libraries for image processing, has paid off in the next phase of the project – presented in this paper – the FPGA hardware implementation of the method using hardware description language (VHDL) and Xilinx ISE development environment (Design Suite 14.7).

B. Description of the system designed and built for parallelized implementation on FPGA

In *Fig. 3* polygons with green background symbolize BRAM modules (Block RAM) of the FPGA circuit. These were configured using IP Core Generator tool from Xilinx ISE development system Design Suite to store a selected video frame sequence (image grayscale, 8-bit resolution of 200×200 pixels), using 10 of 38 Kbits of BRAM memory.

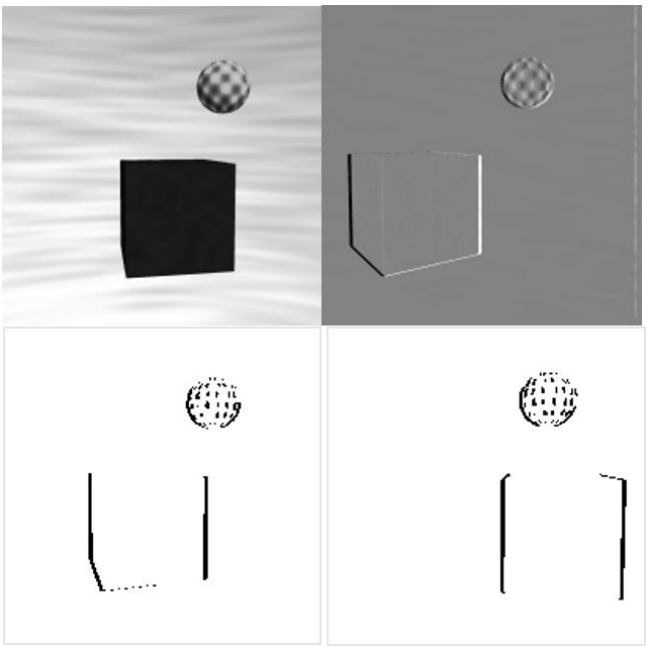

Figure 2: The ANIM video sequence's frame, with computed horizontal, vertical gradients and detected combined edges.

We developed a double pipe-line structure to parallelize execution of operations. The calculation steps determined in the C++ program were implemented here in separate modules that are synchronized by a finite state machine (FSM). Observe the two parallel pipe-lines, processing data from two consecutive frames of video.

Each of these performs the following steps:

- Scanning the image to determine the minimum, maximum and average values, data needed for subsequent calculations, scaling, etc.
- It runs matrix Gaussian filtering algorithm.
- Reading consecutive pixel values, that are inserted into the pipe-line which runs several phases:
  - ✓ vertical and horizontal gradient computation,
  - ✓ positive and negative edge detection,
  - ✓ determining gradient direction.
- After completing these calculations, the results are saved in separate BRAM modules.
- Based on these partial results, which can be computed from two consecutive images in a synchronized manner, the method calculates their intercorrelation.

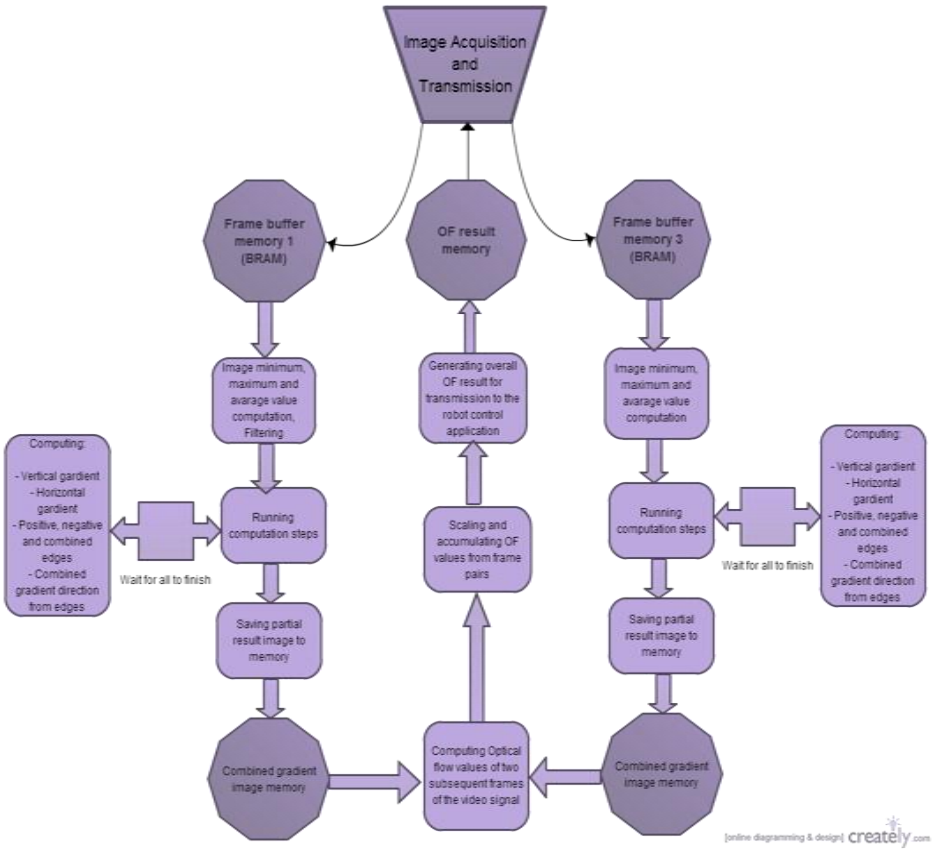

Figure 3: Block diagram and sequence of operations implemented on the FPGA.

- This yields the corresponding OF values.
- The OF values will be scaled and accumulated from several pairs of images in the sequence.
- The end result is saved in the dedicated OF BRAM memory, from where it can be passed on to an application that will use it.

The state-diagram of the finite-state-machine (FSM) controlling one thread of the pipe-line structure is shown in *Fig. 4*. Note the loop formed by the states 1, 2, 3 and 5 corresponding to the data input phase from the BRAM memory (Frame Buffer in *Fig. 4*) and the image parameters computation. It then passes to the second loop (states 4, 5, 6, 7 and 8) where it performs the calculations of the gradient, edge detection, OF, etc. The last state saves the results. The novelty consists in a method able to detect the image parameters while running the filter algorithm, thus saving an entire image scanning cycle.

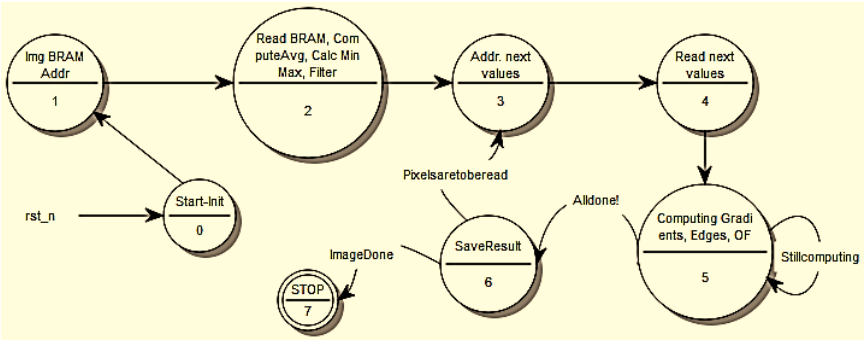

Figure 4: State-diagram of the FSM controlling one thread of the pipe-line structure.

The new, resource-efficient motion estimation method developed by our team, uses an OF extraction algorithm consisting of the following steps:

a) Based on the detections results of the previous stages, from each frame of the video sequence we have generated a flag matrix signaling the edge positions in the image. A flag value (logical 1 bit value) is placed on the *x, y* coordinates of the generated matrix in the vicinity of the locations where an edge is detected. While scanning the input images with the Sobel filter matrixes, for each output value a single bit of the flag matrix is generated, therefore reducing the size of the data to be processed in the next step.

b) The next step consists in scanning the flag matrix pairs generated from two consecutive frames with a 5×5 pixel window to determine the local direction of travel (motion) of the existing edges.

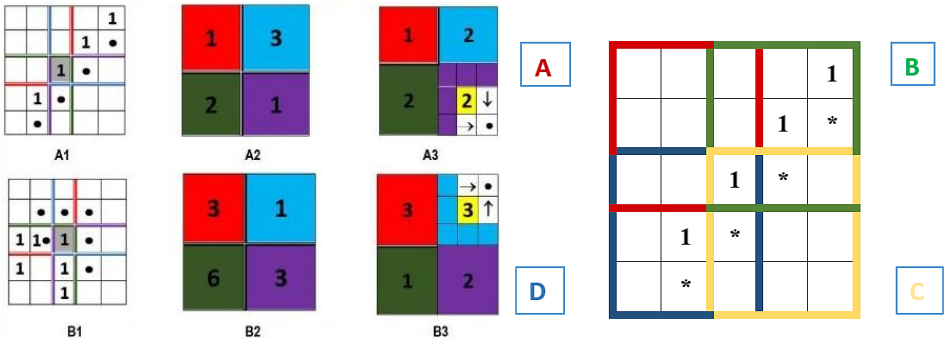

Figure 5: The introduced method for determining the local displacement of edges.

As can be seen in the examples in *Fig. 5*, the evaluation windows are divided into four quadrants, and the resulting value of the direction of movement will be saved to coordinates that are at the intersection of these quadrants of 3×3 pixels. *Fig. 5* - A1 shows a local displacement of the edges formed by 4 points. The number of points in each quadrant is then calculated (*Fig. 1* - A2) for two

consecutive evaluated frames (*i* and *i+1*). An increase in the number of bits in a quadrant shows the direction of movement of the edge. The chosen quadrants will have an increased number of flag bits (*Fig. 1* - A3).

3. Test results of the developed embedded OF extraction system

Translation (synthesizing) programs of functional hardware description languages like VHDL to Verilog do not result in a series of instructions executed sequentially but in a draft of a digital logic circuit required to perform the algorithm described.

In this respect, the test - debug - of these programs is achievable through circuit simulation techniques. However, in order to simulate a digital circuit, implemented using a hardware description language, we need a testbench module (also developed in VHDL or Verilog) that generates input signals for the unit under test (UUT). These will yield time-varying output signals of the UUT, that will reflect the behavior of the designed circuit, thus aiding the debug process.

These VHDL simulation codes can check the outputs of the module under test (UUT - Unit Under Test) for the generated inputs, and returns status messages or error signals if detected. The Xilinx environment provides the ISIM simulation compiler that generates the graphical representation of the input signals, internal signals of the UUT and outputs in the form of timing diagrams.

In this section of the paper we present a few of these diagrams, for the implemented OF extraction project.

In *Fig. 6* one can follow the partial simulation of one thread of the pipe-line structure in *Fig. 3*. Note the double addressing of the dual-port BRAM memory to get two values simultaneously in the same clock cycle. The finite state automaton executes the first loop (states s1, s2, s3 and s5) to control the sequential reading of BRAM and calculation of the image parameters. After reaching the highest memory address (0 ... 39 999, for an image of 200×200 pixels) the FSM transitions to state s4, where the final values of the calculated parameters are available. It is important to note the time required for these operations, which is 800μs in accordance with the same timing diagram.

One of the steps difficult to implement in hardware was the calculation of the image mean values, because it requires at least one division operation, which is only possible in a digital circuit to values which are equal to 2^n.

To solve this problem and minimize the error introduced with divisions by 2^n values closest to the current divider values, we used the following method: division was achieved by using shift registers, and the error was reduced by averaging two consecutive displacement values.

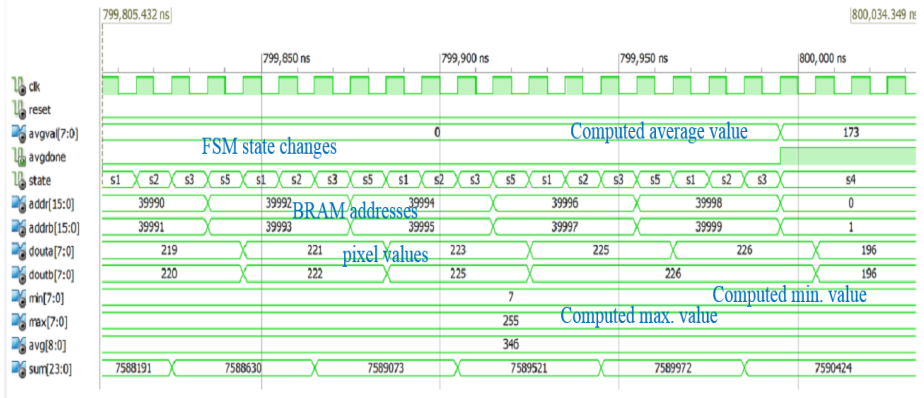

Figure 6: Simulation of reading the input values from BRAM memory (frame buffer).

The image in Fig. 7 shows a complete execution cycle of the developed algorithm by the FSM pipe-line control structure. One can observe in *Fig. 7* the evolution of calculating the image minimum and maximum values, followed by the second loop, with the gradient computations and scaling. It should be noted in this case, that the total execution time is approximately 1.6 ms. As it results from the analysis of the time diagram in *Fig. 7*, each partial result obtained in state s8 is saved and sent to the next component, namely at the end is placed in a BRAM memory called *Combined gradient memory image* in *Fig. 3*.

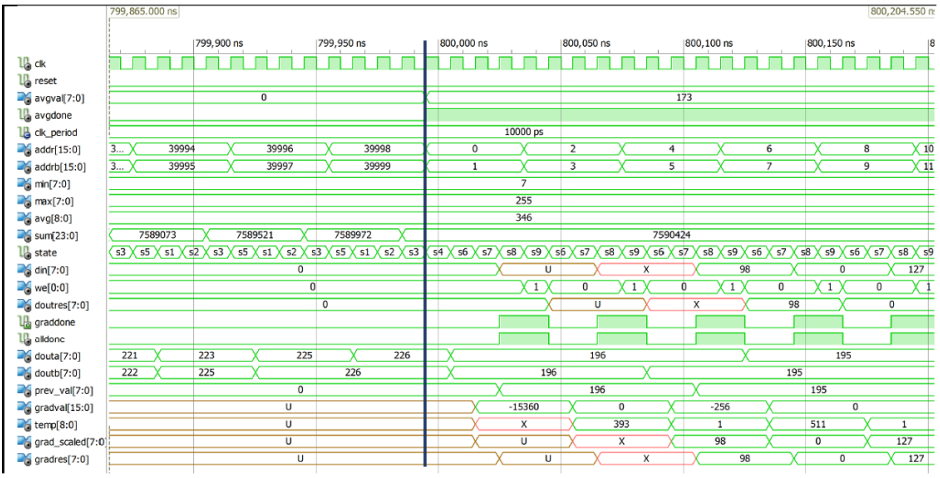

Figure 7: Simulation result showing the passage (vertical marking line) from the parameters calculation loop to calculating image gradients.

Since the hardware resources required to implement this computation flow occupies only about 1-2% of the capacity of the FPGA circuit used (a Xilinx

Virtex 5 FX30T) (Table 1), this structure should be instantiated more than twice, thus leading to a more efficient parallelized structure with the possibility of processing multiple frames of the video sequence simultaneously. This extension, however, is restricted by the number of available FPGA BRAM memories (68). In its current form, with only two parallel pipe-line structures (two frames processed simultaneously) the project uses at least 40 BRAM modules.

Table 1: *Device utilization summary for one thread of the implemented OF extraction pipe-line structure*

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	108	20480	0%
Number of Slice LUTs	206	20480	1%
Number of fully used LUT-FF pairs	79	235	33%
Number of bonded IOBs	20	360	5%
Number of Block RAM/FIFO	20	68	29%
Number of BUFG/BUFGCTRLs	3	32	9%

4. Validating the FPGA implementation of the new method using viciLAB

viciLab [8], [9] is a remote/local FPGA prototyping platform, with GUI console toolsuite support. It enables the user to create and implement a digital logic component application and GUI console. The viciLab tools perform automated creation of the remote/local design FPGA bitstream from a VHDL model description, and perform remote or local Digilent Nexys3 module Xilinx Spartan-6 FPGA configuration. The system also permits user-specific real-time FPGA application development with interactive control/visualization console. The viciLogic wrapper integrates the user design with the FPGA hardware core, and generates design metadata to aid automation and faster and easier GUI prototype development. The HDL parsing process also produces a machine readable description of the HDL design structure, which is used during course building and client GUI application creation to automate the creation of interactive animations. The wrapper auto-detects and connects the SDRAM interface, and clock and reset signals, and provides a user menu for defining signal connections to FPGA module display devices (LEDS and 7-segment displays). The DSPModule is the area where the application's main processing elements are placed. The GUI written in Python retrieves the video feed from a PC's webcam and saves the image frames into the cellular RAM memory of the Nexys 3 development board. The wrapper extracts the image data from the external RAM and drives the signals necessary for the DSPModule (dspBlock)

to perform the designed computation steps. The implemented computation processes of the dspBlock performs the following steps, also shown in *Fig. 8*:

- The video feed from the webcam is converted by the Python GUI into the format with a resolution of 100×100 and 32 bits per pixel.
- The video data is stored in the FPGA board's SRAM in a grayscale format, but still in 32 bits/pixel. In the processing phase, though, only 8 bit / pixel are used as input values.
- Each frame of the webcam video signal is scanned with a 5×5 window in order to perform a Sobel edge detection, using a filter matrix.
- The edges are stored in a 20×20 bit matrix, according to the local edge values found by the previous step.
- This matrix is then processed using the previously presented method with 5×5 local OF detection windows. The 16 resulting windows are processed in parallel by the dspBlock using as many separate VHDL processes.

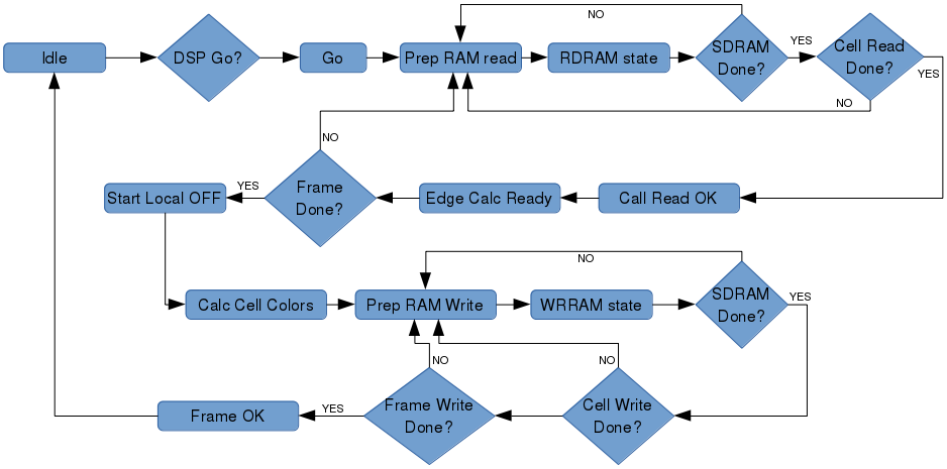

Figure 8: State diagram of the finite state machine controlling the image processing circuit.

As *Fig. 8* shows, the finite state machine (FSM) controlling the OF computation circuits contains a module that is responsible for the display of the OF result values. In order to ease the evaluation of the outputs, a color code has been assigned to each of the eight possible (45, 90, 135, 180, 225, 270, 315 or 360 degrees) optical flow direction values yielded by the circuit.

These colors will fill a square, as *Fig. 9* presents, (overlaying the displayed edges) corresponding to the 5×5 bit windows of the edge bit matrix used as inputs to the displacement computation phase. These will show the OF direction

in the respective areas of the image, yielding the local OF values. Averaging these gives the overall OF value.

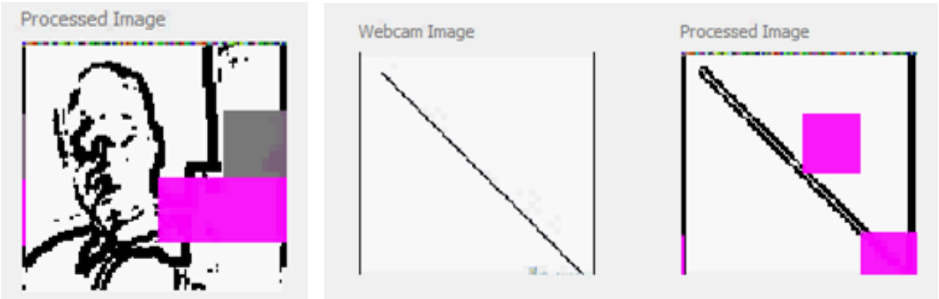

Figure 9: State diagram of the finite state machine controlling the image processing circuit.

Measurements have shown that one complete cycle for local OF and overall OF determination computes in under 25 microseconds, well within real-time requirements.

5. Raspberry Pi implementation of the developed OF method

In order to test the performance of the method on a different embedded platform we have implemented it on a Raspberry Pi compact computer.

To achieve real-time video processing of captured frames from a 30 fps 640×480 PX resolution camera, the platform was chosen to be a Raspberry PI 2 model B. The credit card sized computer contains a Broadcom SoC consisting of a 900 MHz Quad-core ARM CortexA7 CPU, a 250MHz Broadcom Video Core IV GPU, with 1 GB memory. It is also capable of sending data with a speed of 2 Gbps through the CSI-2 connector from a dedicated 5 megapixel camera directly to the GPU.

RGB frames from the camera module can be received with chosen resolution and speed defined in software. The maximum video recording features are 1920×1080 pixel on 30 fps, 1280×720 pixel on 60 fps and 640×480 pixel on 60/90 fps. For the application, 640×480 pixel sized frames where received on 30 fps.

Using OpenCv API to easily process images and to show the results, the procedure begins with transforming the three channel RGB frames received from the camera to one channel grayscale frames. The transformed pixel values were stored in the memory with 8 bit unsigned char values, varying the intensity of the grey value from 0 to 255. Edge detection in the frames is done by the Sobel operator, with 5×5 pixel kernels.

6. Results and conclusions

The designed OF extraction system implemented on a FPGA circuit is functional, operating in real-time. The precision of the OF computation is influenced by the sensitivity of the edge detection phase to the lighting conditions of the environment. One way to overcome this issue is to increase the framerate of the input video signal by using a dedicated camera directly attached to the FPGA development board. We have experimented with the use of an Omniview OF7670 sensor to replace the PC webcam, and found that the framerate would be increased tenfold. This is currently at about 3-5 fps due to the latency of the data communication via the wrapper core between the dspBlk and the webcam. By using the dedicated camera we can reach up to 30 fps with the same dspBlk structure. The limitation in this case proved to be the resources of the Spartan 6 FPGA on the Digilent Nexys 3 board supported by viciLab.

There is room for expansion in this type of project, but with certain limitations. The alternative is, however, the use of the dedicated processor module (PowerPC440 core) of the FPGA used for the execution of those tasks that require sequential steps. On the other hand, by introducing this component into the system, other problems can be solved, such as accessing the external DDR-2 RAM modules of the used OPUS FPGA development platform, as well as real-time image acquisition as input, using peripheral interfaces attached to it.

All these avenues of development will be studied and, if favorable feasibility is found, will be exploited in later stages of the research project.

The viability of the implementation results will need to be validated by demonstrating the method with a mobile robot. The final demonstration will show the collision-free, (semi-)autonomous drive of a mobile robot or even of a group of collaborating robots in a highly-cluttered environment. The implemented systems will yield a new class of artificially intelligent robots that can adapt their hardware structure in order to behave better in a changing environment. Individual or collaborating groups of robots with these abilities could be used in a variety of reconnaissance or monitoring tasks. For instance the capability to assimilate and share acquired knowledge about its environment can be useful in scenarios where hazardous spaces need to be explored and mapped fast (ex. search in earthquake-damaged buildings).

Acknowledgements

The research presented in this paper was supported by the European Social Fund under the responsibility of the Managing Authority for the Sectoral Operational Programme for Human Resources Development, as part of the grant POSDRU/159/1.5/S/133652.

References

[1] Horn, B. K. P., Schunck, B. G. "Determining Optical Flow", Technical Report. Massachusetts Institute of Technology, Cambridge, MA, USA, 1980.

[2] Lucas, B., Kanade, T., "An iterative image registration technique with an application to stereo vision," In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981, pp. 674–679.

[3] Floreano, D., Pericet-Camara, R., Viollet, S., Ruffier, F., Brückner, A. et al. "Miniature curved artificial compound eyes," in *Proceedings of the National Academy of Sciences*, vol. 110, num. 23, p. 9332–9337, 2013.

[4] Duhamel, P.E. et al., "Hardware in the Loop for Optical Flow Sensing in a Robotic Bee," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, 2011, pp. 1099–1106.

[5] Ruffier, F., Franceschini, N., "Optic flow regulation: the key to aircraft automatic guidance," *Robotics and Autonomous Systems*, vol. 50, no. 4, pp. 177–194, March 2005.

[6] Serrão, M., Rodrigues, J. M. F., du Buf, J.M.H., "Navigation Framework Using Visual Landmarks and a GIS," *Procedia Computer Science*, vol. 27, pp. 28-37, 2014.

[7] Xiuqing, W., Zeng-Guang, H., Feng, L., Min, T., Yongji, W., "Mobile robots' modular navigation controller using spiking neural networks," *Neurocomputing*, vol. 134, pp. 230–238, June 2014.

[8] Tomasi, M., Barranco, F., Vanegas, M., Díaz, J., Ros, E., "Fine grain pipeline architecture for high performance phase-based optical flow computation," *Journal of Systems Architecture*, vol. 56, no. 11, pp. 577–587, November 2010.

[9] Botella, G., Ros, E., Rodriguez, M., Garcia, A., Romero, S., "Pre-processor for bioinspired optical flow models: a customizable hardware implementation," in *Electrotechnical Conference MELECON 2006*, IEEE Mediterranean, Limassol, 2006, pp. 93–96.

[10] Zhaoyi, W., Dah-Jye, L., Brent E., N., "FPGA-based Real-time Optical Flow Algorithm," *Journal of Multimedia*, vol. 2, no. 5, pp. 38-45, September 2007.

[11] Díaz, J., Ros, E., Pelayo, F., M. Ortigosa, E., Mota, S., "FPGA-Based Real-Time Optical-Flow System," IEEE *Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 2, pp. 274–279, February 2006.

[12] Schlessman, J. et al., "Hardware/Software Co-Design of an FPGA-based Embedded Tracking System," in *Conference on Computer Vision and Pattern Recognition Workshop CVPRW '06*, 2006, pp. 123–123.

[13] Chase, J., Nelson, B., Bodily, J., Zhaoyi W., Dah-Jye, L., "Real-Time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study," in *16th International Symposium on Field-Programmable Custom Computing Machines*, 2008, pp. 173–182.

[14] Barranco, F., Tomasi, M., Diaz, J., Vanegas, M., Ros E., "Parallel Architecture for Hierarchical Optical Flow Estimation Based on FPGA," *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1058–1067, June 2012.

[15] Jin, S., Kim, D., Nguyen, D. D., Jeon, J. W., "Pipelined Hardware Architecture for High-Speed Optical Flow Estimation using FPGA," in *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Washington DC, 2010, pp. 33–36.

[16] Grabe, V., Bulthoff, H. H., Robuffo G. P., "On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, 2012, pp. 491–497

[17] Vatavu, A., Danescu, R., Nedevschi, S., "Real-Time Dynamic Environment Perception in Driving Scenarios Using Difference Fronts," in *Proceedings of the 2012 IEEE Intelligent Vehicle Symposium*, June 2012, Alcala de Henares, Spain, pp. 717–722.