Embedded Real-time Implementation of a Computational Efficient Optical Flow Extraction Method for Intelligent Robot Control Applications

Róbert Moni[1], László Bakó[2], Szabolcs Hajdú[2], Fearghal Morgan[3] and Sándor-Tihamér Brassai[2]

[1] Sapientia Hungarian University of Transylvania, Faculty of Technical and Human Sciences, Tîrgu Mureş, 547367 Corunca 1C, Romania, robertmoni@ymail.com
[2] Sapientia Hungarian University of Transylvania, Electrical Engineering Department, Faculty of Technical and Human Sciences, Tîrgu Mureş, 547367 Corunca 1C, Romania {lbako, hajduszabolcs, tiha}@ms.sapientia.ro
[3] Bio-Inspired Electronics and Reconfigurable Computing (BIRC) Research Group, National University Ireland Galway, University road, Galway, Ireland, fearghal.morgan@nuigalway.ie

Abstract. The main role of an autonomous car is tracking a path on a determinate distance, while being able to notice road signs and to avoid collisions. Essential parts of these functions are the sensors, which identify the elements in the vehicle's environment. Path following can be done by different ways, from which we will underline the use of a new method, based on video processing and Optical Flow extraction. The aim is to build a real-time system suitable for implementation on resource-restricted platforms. Experimental results with the embedded, real-time implemented – on a FPGA supported Raspberry Pi platform – method are given in the paper, put to use in a line-following mobile robot application with intelligent control. We also prove the applicability of the new method in the take-off and landing stabilization of autonomous UAVs.

Keywords: Optical Flow, Real-time, Embedded, Raspberry Pi, FPGA, edge detection, image processing, robot control, mobile robot, UAV, quadcopter, autonomous vehicles.

1 Introduction

Many advanced artificial vision systems make use of the information provided by the extraction of Optical Flow (OF) from video signals. However, implementing the state-of-the-art OF extraction methods with severe power consumption and computing resource availability constraints can become quite a challenge. This problem can become even more difficult to solve if there is also a need for real-time computation that arises when the target application area is focused on autonomous mobile robots.

On the other hand, these application always rely on complex control algorithms that might use OF as an additional input variable in this system. This leads to a resource distribution that correlates with the role of these algorithmic components.

All in all, the OF extraction needs to execute fast, with high power and resource usage efficiency. Even so, the OF determination is usually only an early phase of a complex control algorithm, that poses the need for optimal resource distribution between tasks. Since in embedded environments the available computational resources and power availability are not as plentiful as in general purpose computing platforms, the challenge to implement real-time image processing and artificial vision algorithms becomes even greater and more appealing.

Several studies in the scientific literature present novel methods for OF computation, discussing accuracy, vector field density and computing complexity [1], [2]. As the classic platforms struggle to offer a viable solution for real-time OF extraction with usable framerate, alternative platform implementations are gaining momentum [3], [4]. Apart from ASICs (Application Specific Integrated Circuit), that arguably could be the best choice, a possible compromising solution is offered by SoCs (System on Chip) and FPGA (Field-Programmable Gate Array) circuit-based hybrid platforms [5], similar to the one presented in this paper.

The essential features and properties of a video signal processing algorithm for a line tracking robot are speed, precision and robustness. The high execution speed of the algorithm is required in order to process in real-time the incoming camera images in real-time transmitting the collected useful information as needed. The track recorded by the camera from different directions, may suffer from different types of noises. Such is the angle of incidence of light, shadows or appearance of bright spots. For a robot control tasks, these have to be eliminated by a robust video signal processing algorithm.

2 Embedded real-time OF extraction method for robot control

The main function of a line tracking robot is to follow a predetermined path with the help of a sensor that collects data about the line position in an autonomous mode. These robots are used in multiple industrial domains, especially in product transportation inside storage spaces, in automated hospitals for delivering medication and medical equipment. The most common and simplest way for a robot to follow a line is by the usage of infrared sensors. In these applications the infrared sensors work as a switch and transmit information about the position of the line under the robot. Another frequently used sensor is the photoresistor. The disadvantage of the methods presented above is the fact that these sensors only “see” a small portion of the line thereby controlling the robot is not adaptive. In case of an automobile these methods can’t be used. A newer method uses video signal processing which contains more information about the path. The camera attached to the robot can “see” a larger portion of the tracked line. This application can be adapted to aid the intelligent control of autonomous vehicles, too.

2.1 Methods to extract the tracked line from the images

The basic characteristics of video signal processing algorithms are: execution speed, precision and robustness. In articles studied during this research many methods

with the scope of line extraction from video images where examined. The first studied method [3] processes the images transformed into the Hue Saturation Intensity (HIS) color model which is very close to the human perception of color. In this model, the Hue image is used which is separated based on light intensity eliminating these types of noises. The second method [4] is based on artificial neural networks. In this application, the robot equipped with a camera, learns to auto-navigate on a predetermined path. The third method [6] processes binary images in black and white and extracts the objects' skeleton using morphology. With the methods for edge detection, the tracked line can be extracted from the image precisely and in real time. Methods frequently used for edge detection are Sobel [7], Canny [8], Prewitt [9] and Marr Hildrett [10]. Two methods were tested and implemented for edge detection. In these tests the velocity, precision and robustness of the image processing methods where analyzed. Each of the methods uses filtering masks (see Figure 1 and Figure 2).

-1	0	+1
-2	0	+2
-1	0	+1

-1	-2	-1
0	0	0
+1	+2	+1

Fig. 1. Sobel masks for edge detection, Vertical *(left, Gx)* and Horizontal *(right, Gy)*

The OpenCV library was used that offers fast and optimal image processing solutions based on C/C++ language. The Sobel mask of 3x3 pixels is computed as follows, than the derivative of one pixel in the image is calculated:

$$\begin{aligned} Pixel\_GX_{[x,y]} = {} & Gx[0,0] * img[x-1,y-1] + Gx[1,0] * img[x,y-1] + Gx[2,0] \\ & * img[x+1,y-1] + Gx[0,1] * img[x-1,y] + Gx[1,1] * img[x,y] \\ & + Gx[2,1] * img[x+1,y] + Gx[0,2] * img[x-1,y+1] + Gx[1,2] \\ & * img[x,y+1] + Gx[2,2] * img[x+1,y+1] \end{aligned} \quad (1)$$

$$\begin{aligned} Pixel\_GY_{[x,y]} = {} & Gy[0,0] * img[x-1,y-1] + Gy[0,1] * img[x-1,y] + Gy[0,2] \\ & * img[x-1,y+1] + Gy[1,0] * img[x,y-1] + Gy[1,1] * img[x,y] \\ & + Gy[1,2] * img[x,y+1] + Gy[2,0] * img[x+1,y-1] + Gy[2,1] \\ & * img[x+1,y] + Gy[2,2] * img[x+1,y+1] \end{aligned} \quad (2)$$

where [x, y] = [0:5:n-1].

Afterwards, the results are combined to give the gradient so that the edges can be extracted:

$$G = \sqrt{\text{Pixel\_Gx[x,y]}^{\,2} + \text{Pixel\_Gy [x,y]}^2} \quad (3)$$

The Canny method has more steps for edge detection, these steps are the following:

1. Filtering the image using the Gaussian method,

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2 + \left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1) \quad (4)$$

2. Detecting the intensity of the gradient,
3. Filtering the weak edges,
4. Thresholding two times,
5. Thickening the dominant edges.

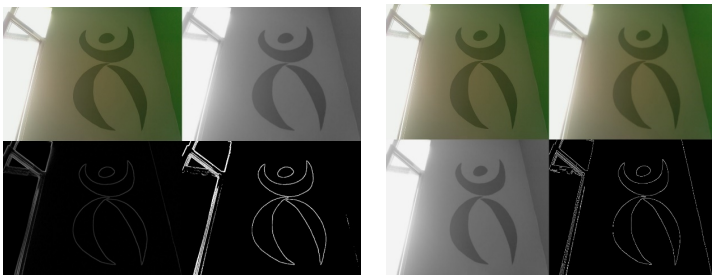

Fig. 2. Preprocessing the images using edge detection methods *Sobel (left)* and *Canny (right)*

2.2 The implemented line detection algorithm

The 2D images captured by the camera hold valuable information for the robot about the tracked line. For a better understanding of the information processing method first a research in flow direction determination will be presented which resulted the development of a real time algorithm for optical flow direction determination. There are many articles that explain the optical flow phenomena [11]. The evaluation of the gradient method is based on spatial-temporal derivatives. The approaches in the articles of Horn and Schunk [12] or Lucas and Kanade [13] work well and forms the base of the gradient method developed by the authors [5].

The gradient methods work by processing two consecutive images. As shown in Figure 3, the object - in this case a pixel - acquired at time t in the position P moves to position P', acquired at time $t+\Delta t$. The difference can be determined with derivation in the Cartesian coordinate system, therefore the optical flow can be calculated. For the optical flow calculation it was specified that the object moves in one direction from the point of view of the camera and the intensity of the object doesn't change:

$$DI(p,d,\Delta t) = I(p+d, t+\Delta t) - I(p,t) \tag{5}$$

where:

- DI – intensity difference,
- p – the point on the image,
- d – the motion.

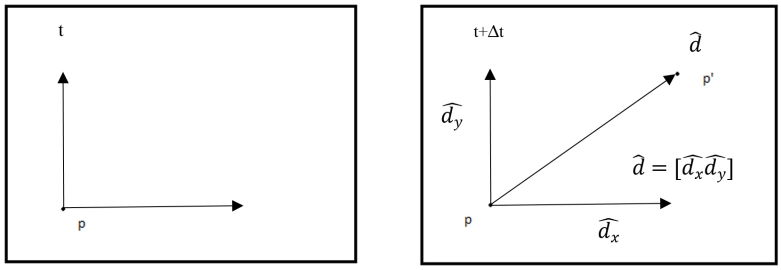

Fig. 3. Optical flow detection with the gradient method

This paper proposes a new gradient based method, with the aim of optical flow direction determination in real time. Two consecutive images are processed with the help of a 5x5 pixel mask composed from four overlapping quadrants (A, B, C, D). The optical flow direction is detected from the activated quadrants. The mask is iterated trough the two consecutive images and is evaluated at the same position. A quadrant is activated when it contains edge pixels and the number of these pixels changes (decreases or increases) between the image captured at time t and the image captured at time $t+\Delta t$. The method collects the information from the activated quadrants, namely which quadrant has gained the most pixels and the number of pixels accumulated. For example, on the left image of Figure 4 the pixels marked with the number *1* represent the position of a detected edge at time t, and the pixels marked with a * represent the position of a detected edge at time $t+\Delta t$. Thereby, in the C quadrant the number of pixels has increased, meaning the quadrant was activated and the optical flow was detected in the direction of the C quadrant. The center image of Figure 4 shows the directions that the offered method can specify. Figure 5 presents the steps taken to process the image and determine the optical flow direction. Dividing the image into frames is achieved by placing a fitting number of mask (kernels) instances on the captured images. At this point, each and every frame is examined in both images at the same position. During examination the algorithm searches for dominant quadrants and eliminates unneeded calculations, such as frames with no edges.

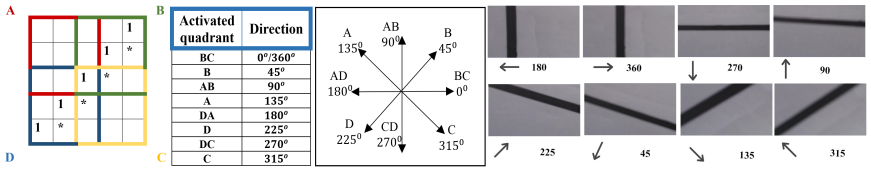

Fig. 4. Mask for optical flow detection (*left*), Directions in the Cartesian coordinate system (*center*), Example frames of the test video sequences (*right*)

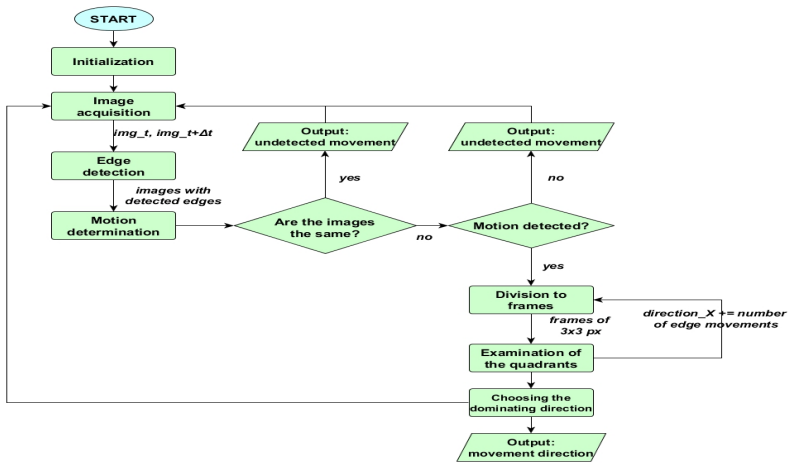

Fig. 5. The optical flow direction extraction algorithm

3 Intelligent visual robot control based on extracted OF data

3.1 Components and structure of the designed robot platform

The robot platform has been designed and implemented as a real-time embedded testing environment for the OF extraction method as well as for the intelligent, fuzzy-like control algorithm, that will be presented in the following section. The robot has three wheels, with two actively driven and one free wheel.

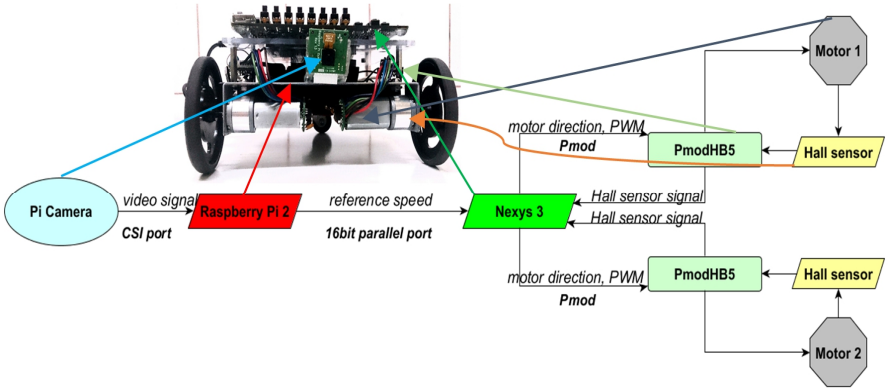

Fig. 6. The main components of the line tracking robot platform

The main computation unit of the robot is the Raspberry Pi 2 Model B (RPi) embedded computer, consisting of a Broadcom BCM2835 SoC with a 900 MHz Quad-core ARM Cortex-A7 CPU and a Broadcom VideoCore IV 250MHz GPU, using 1 GB SDRAM memory. A Pi camera is connected to the dedicated port as the main input signal acquisition module. As Figure 6 presents, the second computing unit of the robot is a Nexys 3 FPGA development board with a Xilinx Spartan-6 LX16 FPGA circuit. This is responsible for interfacing the two 12V DC motors with mounted Hall sensors via two PmodHB5 dual H-bridges that are driving the active wheels of the robot.

The devised robot control system is split into two tasks that are executed in parallel. The first part - that is responsible for the video signal acquisition, OF direction extraction and computing the command signals for the motors (the reference speed) - is executed on the RPi microcomputer.

Part two is implemented in VHDL, therefore corresponding circuits are generated on the FPGA of the Nexys 3 development board, that function in parallel and serve as the actuator signal generators (PWM signals), applying the speed corrections based on the reference values received from RPi. The two platforms are connected via a custom-built 16-bit parallel communication port, using the GPIO pins of the RPi (acting as master) and peripheral module (PMOD) ports of the FPGA board (slave).

The main steps of the control loop are represented in Figure 7.

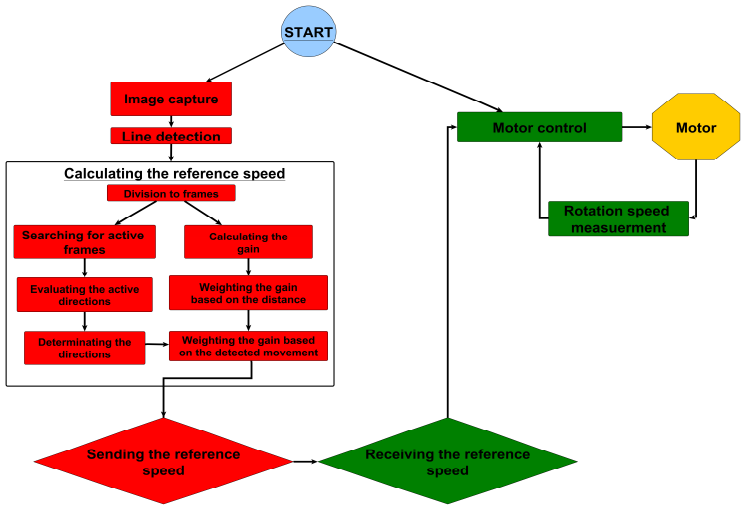

Fig. 7. Execution diagram of the robot control algorithm

3.2 Fuzzy logic-like method for motor control - reference signal computation from video-signal based OF values

In order to avoid controller parameter tuning difficulties, a hybrid proportional controller has been devised and implemented that uses a two layer fuzzy-logic-like rule-set system to compute the command signals. In the first layer, the image processing algorithm assigns each kernel of the kernel grid covering the image frame of the video signal a certain weight, both horizontally and vertically, according to the rule-set presented in the left side of Figure 8. These form gain vectors that are further used to gauge the actuator (DC motor) signal ratios according to the position of the perceived line to be followed in the consecutive frames of the video signal.

These vectors are described by a sigmoid function represented on the right hand side chart of Figure 8. The sum of these weights (vertical and the horizontal) is equal to one, and will modulate the speed of the motors depending on where (on which kernel row and column) has the line been detected on. These weights formed by the gain vectors are not influenced by the extracted OF direction values, only by the location of the line to be followed in images acquired by the Pi camera.

Actually, the images used for this calculation are filtered and all unnecessary information is removed, leaving only the edge locations as useful data. For optimal execution time, the two tasks – the line position detection and the OF direction extraction – are computed by two separate threads of the C++ program run on the Raspberry Pi.

The OF values are used as a second layer weighing factors, determining the preset value of the proportional controller, in other terms, the preset speed of the DC motors driving the active wheels of the robot.

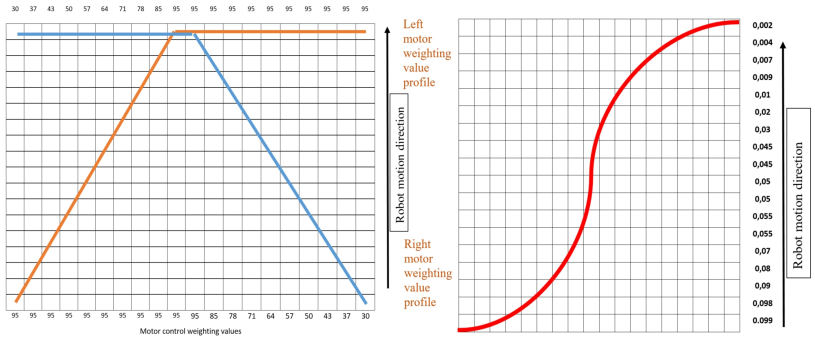

Fig. 8. Fuzzy logic-like method for motor control reference signal computation

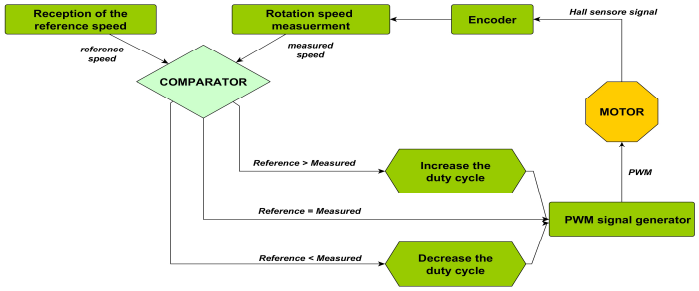

Fig. 9. Block diagram of the robot control structure

The rules in this second weighing layer are the following (the reference is the robot's direction of travel, considered to be 0°, in trigonometric direction):

- If the extracted OF direction is either 90° or 270° then the weight computed in layer one is unchanged, being applied as 100% of the actuator speed.
- If the determined OF direction is either 180° or 0°, the weight computed in layer one is applied to the motor at 85%, presuming the robot wobbles to the sides.
- When the computed OF direction is either 45°, 135°, 225° or 315°, the weight computed in layer one is applied to the motor at 75%, avoiding the line to be followed to exit the field of view of the robot's camera.

After all computations are completed on the RPi, the control signal values for the two motors are transmitted to the FPGA board via a 16 data-bit custom parallel port.

The hardware implemented part (in the FPGA) of the project is then switched on to use the two 8 bit values received to compute the PWM motor command signals, setting the speed of motors. The feedback signal from the Hall sensors mounted on the motors is also measured by the FPGA implemented modules (Figure 9). This feedback is used to smooth the PWM changes in order to avoid sharp direction changes of the robot. These circuits where implemented in VHDL, using Xilinx PlanAhead and tested and verified in hardware using viciLab. viciLab [14] is a remote/local FPGA prototyping platform that enables the user to create and implement a digital logic component application and GUI console. It performs automated creation of the design FPGA bitstream from a VHDL model description, and performs remote or local module Xilinx Spartan-6 FPGA configuration.

4 Measurements and experimental results

4.1 Results of the introduced OF method with different edge-detection phases

The following measurements, presented in Figure 10 and Figure 11, are investigating the accuracy of the algorithm, the sensitivity of the cases when Sobel and Canny edge detection algorithms were used. After a pre-processing phase, the edge detection method is applied to the images. The measurements show, that using the Sobel edge detection method the results are noisier, but slightly faster. The use of the Canny method yields a cleaner solution as a response to the shift in movement direction, but shows a slightly slower execution time.

Measurements that used the Sobel method yielded an average running time of 40.3418 Frames per Second (FPS), while using the Canny method, the average running time was 38.3453 FPS. The same figures show that the execution time decreases when no motion is detected in a pair of frames. This is indicated as a zero degree value on the motion direction plot. The spikes on the execution time plot during the zero value of the direction curve are due to the resource sharing of the system, managed by the Raspbian operating system.

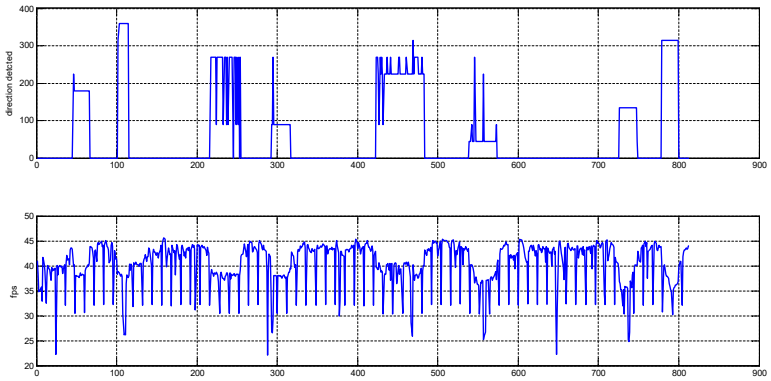

Fig. 10. Optical flow extraction with Sobel edge-detection in the pre-processing phase

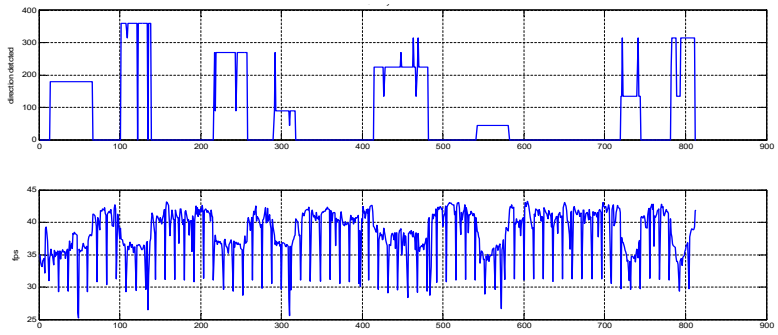

Fig. 11. Optical flow extraction with Canny edge-detection in the pre-processing phase

4.2 Measurement results on the processing speed of the introduced OF method

Table 1 presents the execution speed of the developed algorithm for determining the optical flow direction using different resolutions and programing languages. One of the goals of the research was for the algorithm to work in real time namely to achieve 25 FPS or more.

In the presented table one can observe that the algorithm is well capable of reaching the expected results. The algorithm has been tested using pre-recorded video-sequences, showing a black line moving in eight different direction on a white background (see the image on the right of Figure 4). The recording resolution of the sequences used for testing was 640x480 at 30 FPS. The recording and the frame breakdown was performed using the OpenCV library.

Note that the video player function used in the OpenCV library solution is significantly slower than reading directly from the dedicated camera of the Raspberry Pi, so it influences the running time of the algorithm, expressed in FPS.

Table 1. Execution speed of the OF extraction algorithm with different scenarios

Programing Language	*Captured image Resolution*	*Processed image resolution*	*Multi-threading*	*Processing speed [fps]*
C++	640x480	640x480		**8.5**
C++	640x480	100x100		**22**
C++	320x240	320x240		**21**
C++	320x240	100x100		**89**
C++	160x120	160x120		**105**
C++	320x320	320x320	Yes	**25**
Python	640x480	640x480		**2**
Python	640x480	100x100		**10**
Python	320x240	320x240		**7**
Python	320x240	100x100		**12**
Python	160x120	160x120		**15**
Python	320x320	320x320	Yes	**28**

4.3 Experimental results on the OF method's applicability in UAV motion detection

Another potential application group that might make use of the OF extraction method developed by us is the take-off and landing stabilization of autonomous UAVs. In order to prove this, we have built a custom experimental test-bench. In the previously presented application, the Pi camera was mounted externally, on a wheeled mobile robot facing the direction of movement.

For this experiment, we have mounted the camera inside the RPi's case, in its dedicated slot. In order to simulate the rotating displacement during take-off and landing of a UAV (for instance a quadcopter) we have placed the cased microcomputer on the rotor of a servomotor, also commanded by the RPi. Patterns, similar to those presented in the right side of Figure 4, but containing several lines placed in different angles, have been situated in front of the camera. By commanding (with PWM signals generated by the RPi via GPIO pins) the servomotor to turn in both directions, to different angles and with various speed values, we have set up an environment that resembles the motion of a flying robot. The plots in Figure 12 present the results of these experiments.

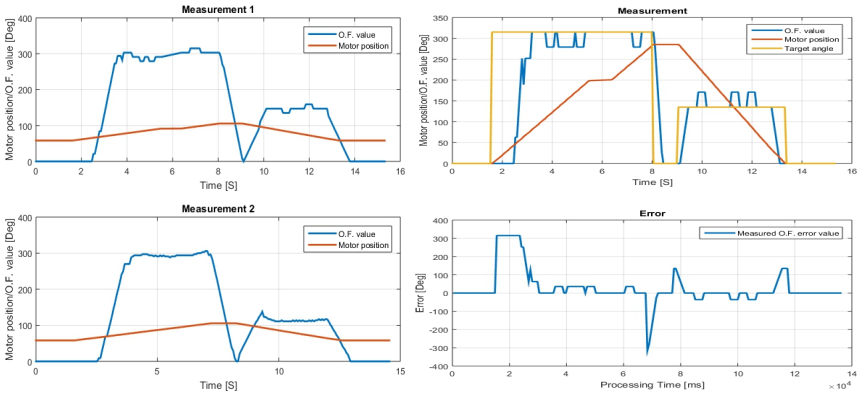

Fig. 12. Measurement results of the UAV motion detection experiments

One can observe two separate measurements in the left side of this figure, both represented by a graph showing the motor position (angle) and another graph presenting the OF direction value extracted using the same method described earlier. The *O.F. value* plots show the direction value (as in the center of Figure 4) of the perceived motion in the video signal determined in real-time by our algorithm running on the RPi. The *Motor position* plots in these figures show the servo-motor's rotor angle value as it was turning the RPi. One can notice that the motor was commanded to make turns in both directions (plot is zero where the motor stopped), with a preset speed (degrees/sec). This resulted in a motion of the pattern of lines placed in front of the Pi camera. The placement of these lines and the motor's rotation direction determined the preset OF value (*Target angle* in the upper right side of Figure 12) to be measured by the implemented system. This plot shows, that the measured *O.F. value* is computed with a delay (until the lines turned to the FOV of the camera) at the beginning of the test and when the motor changed direction. The error plot (bottom right of Figure 12) shows high values at the same phases of the measurement. The usual performance measures and the evaluation methodology of Optical Flow implementations were introduced by Barron et al. [15]. We focused our attention on one error metric, the flow endpoint error (or pixel distance) defined in [16]. Table 2 summarizes our results compared to similar implementations given in [17].

Table 2. Comparison of the introduced method (*BakoMon-OF*), with other methods [17]

Algorithm	***Endpoint Error***	***Memory Space (MB)***	***Number of GOPS***
fSGM [17,18]	4.54%	20.68	37.53
Lucas-Kanade [17]	15.78%	1.47	3.15
NG-fSGM [17]	4.71%	4.39	4.31
BakoMon-OF	14.2%	0.23	5.36

5 Conclusions

We have successfully introduced a new real-time OF extraction method, implemented on resource restrictive embedded platforms. The method is accurate enough to be used – possibly in conjunction with, or complemented by data acquired from IMUs (Inertial Measurement Unit) – as the input value to intelligent robot

control algorithms. In order to prove this, we have designed and built a line tracking mobile robot, with on-board OF computation. The robot can autonomously follow a line drawn on any surface, driven by a custom, fuzzy-logic-based control algorithm, also devised by our team. Using a different experimental setup, we have also shown that the developed systems can be put to use in the intelligent flight control of UAVs.

Acknowledgments. The research presented in this paper was supported by the Sapientia Foundation - Institute for Scientific Research.

References

1. Basch, M.E., Cristea, D.G., Tiponut, V., Slavici, T.: Elaborated motion detector based on Hassenstein-Reichardt correlator model. Latest Trends on Systems, 1, 192-195, 2010.
2. Xiaofeng, R.: Local grouping for optical flow, Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. IEEE, 2008.
3. Elhady, W.E., Elnemr, H.A., Selim, G.: Implementation of autonomous line follower robot, Journal of Computer Science 10 (6): 1036-1044, 2014.
4. Gonzalez, R.C., R.E. Woods and S.L. Eddins: Digital Image Processing Using MATLAB, 2nd Edition, Gatesmark Publishing, ISBN 10:0982085400, pp: 827, 2009.
5. Bakó, L., Morgan, F., Hajdú, Sz., Brassai, S.T., Moni, R., Enăchescu, C.,: Development and Embedded Implementations of a Hardware-Efficient Optical Flow Detection Method, Acta Universitatis Sapientiae Electrical and Mechanical Engineering, 6 (2014) 5-19
6. 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.
7. Duda, R., Hart, P.: Pattern Classification and Scene Analysis, John Wiley and Sons,'73, pp.271-277
8. Canny, J.: A Computational Approach to Edge Detection. IEEE Trans. Pattern Anal. Mach. Intell. 8, 6, pp.679-698, 1986.
9. Prewitt, J.M.S.: Object Enhancement and Extraction, in "Picture processing and Psychopictorics", Academic Press, 1970.
10. Marr, D., Hildreth, E.: Theory of Edge Detection, Proceedings of the Royal Society of London. Series B, Biological Sciences 207, (1167): 187–217, 1980.
11. Thota, S.D., Vemulapalli, K.S., Chintalapati, K., Gudipudi, P.S.S.: Comparison Between The Optical Flow Computational Techniques, International Journal of Engineering Trends and Technology (IJETT) Vol. 4 Issue 10, 2013.
12. Horn, B.K.P., Schunck, B.G.: Determining Optical Flow, Technical Report. Massachusetts Institute of Technology, Cambridge, MA, USA, 1980.
13. 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, pp. 674–679, 1981.
14. Morgan, F., Cawley, S., Kane, M., Coffey, A., Callaly, F.: Remote FPGA Lab Applications, Interactive Timing Diagrams and Assessment, Irish Signals & Systems Conference, 2014.
15. Barron, J., Fleet, D., & Beauchemin, S.: Performance of optical flow techniques. International Journal of Computer Vision, 12 (1), 43–77, 1994.
16. Otte, M., & Nagel, H.-H.: Optical flow estimation: advances and comparisons, in Proceedings of the European conference on computer vision (pp. 51–60), 1994.
17. Xiang, J., Li, Z., Blaauw, D., Kim, H.S. and Chakrabarti, C.: Low complexity optical flow using neighbor-guided semi-global matching, 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, pp. 4483-4487, 2016.
18. Hermann, S. and Klette, R.: Hierarchical scan line dynamic programming for optical flow using semi-global matching, Computer Vision-ACCV Workshops, pp. 556-567, 2012.