Motion Detection by Microcontroller for Panning Cameras

Jesús Benito-Picazo[1(✉)], Ezequiel López-Rubio[1], Juan Miguel Ortiz-de-Lazcano-Lobato[1], Enrique Domínguez[1], and Esteban J. Palomo[1,2]

[1] Department of Computer Languages and Computer Science, University of Málaga, Bulevar Louis Pasteur, 35, 29071 Málaga, Spain
{jpicazo,ezeqlr,jmortiz,enriqued,ejpalomo}@lcc.uma.es
[2] School of Mathematical Sciences and Information Technology, University of Yachay Tech, Hacienda San José s/n, San Miguel de Urcuquí, Ecuador
epalomo@yachaytech.edu.ec

Abstract. Motion detection is the first essential process in the extraction of information regarding moving objects. The approaches based on background difference are the most used with fixed cameras to perform motion detection, because of the high quality of the achieved segmentation. However, real time requirements and high costs prevent most of the algorithms proposed in literature to exploit the background difference with panning cameras in real world applications. This paper presents a new algorithm to detect moving objects within a scene acquired by panning cameras. The algorithm for motion detection is implemented on a Raspberry Pi microcontroller, which enables the design and implementation of a low-cost monitoring system.

Keywords: Foreground detection · Background modeling · Probabilistic self-organizing maps · Background features

1 Introduction

Moving object detection is very important for video surveillance. This task is known to be a significant and difficult research problem in many real environments. Motion detection consists of detecting a change in the position of an object relative to its surroundings or a change in the surroundings relative to an object.

Video surveillance systems have become an extremely active research area due to increasing levels of social conflict and public awareness about security issues. This has led to motivation for the development of robust and precise automatic video surveillance systems, which are essential tools for safety and security in both public and private sectors.

J.M. Ferrández Vicente et al. (Eds.): IWINAC 2017, Part II, LNCS 10338, pp. 279–288, 2017.
DOI: 10.1007/978-3-319-59773-7_29

One of the most common algorithms is to compare the current frame with the previous one. If the pixel difference is bigger than a predefined alarm level or threshold, a motion event alarm is generated. The estimated background is just the previous frame. It clearly works under easy conditions of foreground objects, motion speed and frame rate but it is very sensitive to the threshold so that for a noisy image, motion will be detected in many places even if there is no motion at all. If the object is moving smoothly, a small change, which is less than the predefined threshold, is obtained. Therefore, the moving object would not be detected.

Microcontroller boards are economic, small and flexible hardware devices. They are frequently employed in motion detection systems due to their low energy consumption and reduced cost. Kinetically challenged people can benefit from microcontroller based input devices specifically designed for them, which measure motion on a plane in real time [11]. A flexible Printed Board Circuit (PCB) prototype which integrates a microcontroller has been proposed to estimate motion and proximity [5]. In this prototype, eight photodiodes are used as light sensors. The efficiency of solar energy plants can be improved by low power systems which estimate cloud motion [6]. The approximation of the cloud motion vectors is carried out by an embedded microcontroller, so that the arrangement of the solar panels can be optimized for maximum electricity output. Energy-saving street lighting for smart cities can be accomplished by low power motion detection systems equipped with low consumption microcontrollers and wireless communication devices [1]. This way, the street lamps are switched on when people are present in their surroundings. Finally, a motion detection algorithm based on Self-Organizing Maps (SOMs) was developed in an Arduino DUE board [10]. The implementation of the SOM algorithm was employed as a motion detector for static cameras in a video surveillance system.

Research on computer vision systems based on pan-tilt-zoom (PTZ) cameras has been intense for many years [2,4,8,13]. Nevertheless, there is a lack of a comprehensive theory which sets the foundations for the development of practical systems. Fragmentary approaches that are limited to some parts of the problem are available, but it is still not clear how to combine them to yield complete and reliable systems that can be deployed in many situations. In the present work we focus on the panning movement of a PTZ camera, which is able to cover the entire environment of the camera.

In this paper, we propose a motion detection algorithm and its real-time implementation on an inexpensive microcontroller. The system is able to detect motion by analyzing the output of a panning PTZ camera with the help of a feed forward neural network. Section 2 presents the motion detection algorithm for panning cameras. Section 3 outlines the hardware part of the system, which is based on the Raspberry Pi 3 model B microcontroller, and the employed software architecture. Experimental results on real video footage are reported in Sect. 4. Finally, Sect. 5 contains our conclusions.

2 Methodology

As mentioned before, our goal is to detect the motion of foreground objects while a PTZ camera is moving. Let us consider the image acquired by the camera:

$$f \; : \; \mathbb{R}^3 \rightarrow \mathbb{R}^3 \tag{1}$$

$$f\left(x_1, x_2, t\right) = \left(y_1, y_2, y_3\right) \tag{2}$$

where $\mathbf{x} = (x_1, x_2) \in [-A, A] \times [-B, B]$ are the video frame coordinates in pixels, with $(0, 0)$ at the center of the image and frame size $(2A) \times (2B)$ pixels; t is the time instant; and $\mathbf{y} = (y_1, y_2, y_3)$ comprises the color tristimulus values at the frame location and instant of interest.

For a PTZ camera moving in the horizontal direction one can write:

$$\begin{gathered} \left(x_1, x_2\right), \left(x_1 + \delta, x_2\right) \in [-A, A] \times [-B, B] \Rightarrow \\ f\left(x_1, x_2, t\right) \approx f\left(x_1 + \delta, x_2, t + \epsilon\right) \end{gathered} \tag{3}$$

where δ is the observed horizontal displacement of the image as ϵ units of time have elapsed, and the equality does not hold due to optical effects such as lens aberration, and the motion of foreground objects in the scene. The precondition means that the approximation applies to those points in the scene that are visible both at time t and $t + \epsilon$; the remaining pixels in the video frame must be ignored for our purposes. Then the error in the approximation can be computed as follows:

$$\mathcal{E}\left(x_1, x_2, t\right) = f\left(x_1, x_2, t\right) - f\left(x_1 + \delta, x_2, t + \epsilon\right) \tag{4}$$

For given values of ϵ and the camera speed, the value of δ can be estimated experimentally by finding the value of δ which minimizes $\|\mathcal{E}\|$, where $\|\cdot\|$ stands for any suitable norm. Now it is important to realize that the error comes from two sources, namely optical effects and the presence of foreground objects:

$$\mathcal{E}\left(x_1, x_2, t\right) = \mathcal{E}_{optical}\left(x_1, x_2, t\right) + \mathcal{E}_{objects}\left(x_1, x_2, t\right) \tag{5}$$

where the optical effects can be assumed to be small with respect to the foreground objects effect, if those objects are present:

$$Fore\left(t\right) \Leftrightarrow \left\|\mathcal{E}_{optical}\left(x_1, x_2, t\right)\right\| \ll \left\|\mathcal{E}_{objects}\left(x_1, x_2, t\right)\right\| \tag{6}$$

where $Fore\left(t\right)$ means that there are foreground objects in motion at time t. Moreover, if there are no foreground objects, then the associated error is zero:

$$Fore\left(t\right) \Leftrightarrow \mathcal{E}_{objects}\left(x_1, x_2, t\right) \neq 0 \tag{7}$$

Therefore, the expectation of the error norm should be larger when foreground objects are present:

$$E\left[\left\|\mathcal{E}\left(x_1, x_2, t\right)\right\| \; \middle| \; Fore\left(t\right)\right] > E\left[\left\|\mathcal{E}\left(x_1, x_2, t\right)\right\| \; \middle| \; \neg Fore\left(t\right)\right] \tag{8}$$

Our proposal takes advantage of this by training a feed forward neural network classifier in order to estimate the probability that foreground objects are present, $P\left(Fore\left(t\right)\right)$. To this end, the error norm is summarized by pixel columns, so that the sum of the norms of the differences of the pixels at columns x_1 and $x_1 + \delta$ is computed. Then the error norm sums are added for contiguous pixel columns, so that a reduced set of sums of error norms are obtained. These sums are provided as inputs to the neural network, while the desired output $z\left(t\right)$ is 1 whenever $Fore\left(t\right)$ holds, and -1 otherwise. The probability is then estimated as follows:

$$P\left(Fore\left(t\right)\right) = \frac{1}{2}\left(z\left(t\right) + 1\right) \tag{9}$$

Subsequently a probability threshold θ is applied in order to declare whether foreground object motion has been detected:

$$Detection\left(t\right) \Leftrightarrow P\left(Fore\left(t\right)\right) > \theta \tag{10}$$

Next the details of the implementation of the above proposal are described.

3 System Architecture

Hardware choice is such an important issue when it comes to microcontroller-powered computer vision applications. In general, projects involving real-time motion detection should consume a minimal amount of computing power, but at the same time, they must be affordable and low-energy consuming insofar as a certain amount of them may be required to monitor a medium sized building or building complex and the spots they are going to be placed in may not have access to the general power network. All these reasons present Raspberry Pi class microcontrollers as a good choice for our project. Hence, we have chosen a Raspberry Pi 3 model B microcontroller (Fig. 1), running under Linux Raspbian distribution.

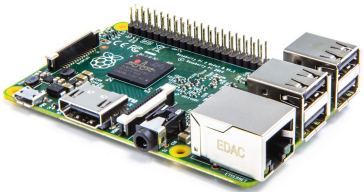

Fig. 1. Raspberry Pi 3 model B overview

This device features an ARM CortexV8 Quad Core CPU running at 1200 MHz, 1 GB RAM, and a 8 GB micro-SD data storage card. It can be powered by a 5.1 V power source and its power consumption reaches 1.2 Amps/h approximately at max operating load.

The second component of our system architecture is a PTZ camera software emulator called *Virtual PTZ* [3]. This software consists of a C++ library that simulates the functionality of an actual Sony SNC-RZ50N PTZ camera from spherical panoramic video footage. In particular, the experiments in this paper employ sequences obtained by a *Point Grey Ladybug 3 Spherical camera* (Fig. 2).

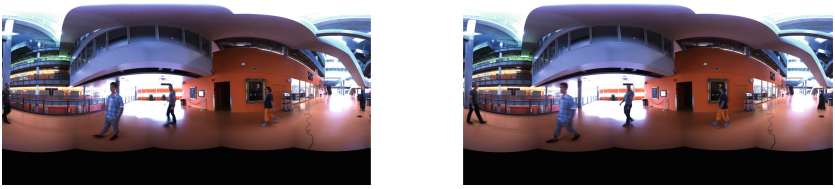

Fig. 2. 360° spherical images supplied by the *Point Grey Ladybug 3 Spherical camera*.

When it comes to PTZ cameras, *Virtual PTZ* software has been proposed as a valid framework for researching because of its capability of substituting a real PTZ camera by providing the user with the possibility of moving the virtual camera through an almost-spherical 360° video frame that can be totally controlled and that is not affected by dynamical issues or physical limitations. Since the only output our system requires from the PTZ camera is real-time video streaming from a panning-capable camera, the virtual PTZ software stands as a convenient framework for this project.

4 Experimental Results

As explained in Sect. 2, our motion detection system can be regarded as a classifier which consists of an algorithm that is in charge of obtaining training samples from a set of consecutive images and supplying them to a multilayer perceptron that will decide whether there are foreground moving objects in the video frames supplied by a PTZ camera. Because of its speed and ease of use, the multilayer perceptron implementation chosen for this project is the fast artificial neural network from Nissen [9]. In order to increase our control over the experimentation process, tests have been performed from videos supplied by the Virtual PTZ camera. For the same reason, as the video frame rate is about 16 fps, camera rotation speed has been adjusted to a constant rate of 16 degrees per second to the left. After performing several tests, the δ value has been estimated as 5 pixels/degree and Mean Squared Error (MSE) has been considered as the error norm $\|\mathcal{E}\|$ (see Sect. 2). All set up, to perform the comparison of two frames, the process described in Fig. 3 has been carried out.

First, frame $n+1$ is shifted δ pixels to the left with respect to frame n to compensate camera rotation. Next, both frames are divided into 63-pixel wide stripes (all but the last stripe, which will be 68-pixel wide) and the mean squared error is calculated for each stripe of the two frames. Finally, a 30 component

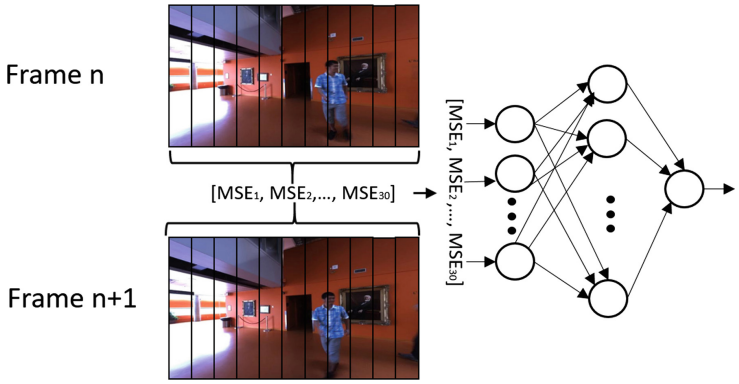

Fig. 3. Example of how samples are obtained and supplied to the perceptron.

vector (10 for each RGB color channel) plus one number, which will be 1 if the sample is positive and −1 if the sample is negative, will be saved as a sample for perceptron training, validation and testing.

In order to evaluate system performance and accuracy when it comes to detecting movement in video streams supplied by the Virtual PTZ, several tests have been performed. These tests involve multilayer perceptron general performance values measured for various different network topologies.

Multilayer perceptrons can be calibrated by modifying several parameters with the objective of achieving better performance rates. However, since the number of parameter combinations would eventually grow exponentially, testing the system by varying every parameter would not be practical. Therefore, for this work, perceptron training and performance comparisons are done just by modifying the number of neurons in its hidden layer, while keeping fixed the rest of them. Thus, the neural network used for this system will have the characteristics listed in Table 1.

Table 1. Test parameters for multilayer perceptron

Neural network class	Multilayer perceptron
Number of inputs	30
Number of neurons in hidden layer	50–600
Number of outputs	1
Learning algorithm	Backpropagation
Max training epochs	10000
Learning rate	0.7

In order to guarantee a correct neural network performance evaluation as much as possible, a 10-fold cross-validation procedure has been established for

our system, so separate sample sets have been used for the training, validation and test phases. For this purpose, it is well known that several measures are available. Because of its simplicity, one of the most popular ones is the classification accuracy, which computes the number of correct predictions divided by the total amount of test samples [12]. The networks have been trained with a class-balanced set of 576 samples. Figure 4 shows the accuracy values for the training, validation and test sets versus the number of neurons in the hidden layer of the perceptron.

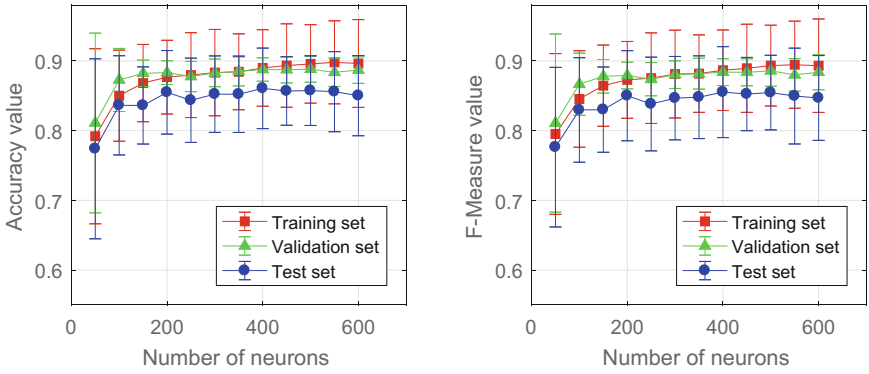

Fig. 4. Accuracy and F-measure error bar chart for training, validation and test sets (higher is better).

Even though accuracy is widely accepted as an acceptable performance measuring criterion, specially in cases like the one presented here, where the number of positive and negative samples is balanced, it is interesting to extend our experimental results with the F-measure performance (Fig. 4), as F-measure is considered as another valid performance measure which eventually can be even more reliable than accuracy [12].

As both accuracy and F-measure limit their performance measurement to one single threshold value, it has been considered that a measurement that integrates all possible threshold values is necessary in order to complement the results represented in the above charts [7]. For this purpose the AUC *Area under the Curve*, has been calculated for every neural network model. So, in Fig. 5 the AUC values for our model can be seen.

All three charts illustrate how the model reaches high performance levels from 200 hidden layer neurons on, and increases its stability as the number of hidden layer neuron number grows higher. Results also show that for test samples in configurations comprehending 200 neurons or more, all three performance measures are above 80%. It is also remarkable that in order to prevent image data loss, all the results above were obtained by just processing images as they come from the Virtual PTZ. This means that no further image processing has been done to correct neither camera sensor noise nor camera lens aberration.

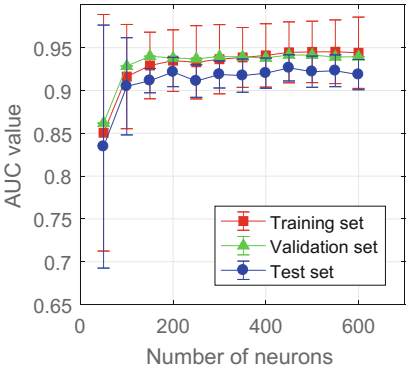

Fig. 5. AUC error bar chart for training, validation and test sets (higher is better).

Time consumption is an absolutely critical issue when it comes to real-time video processing, not to say when using microcontrollers to undertake artificial neural network processes which involve real-time presence detection from a panning camera as the one explained here does. Therefore, the algorithm not only has to be reliable but it also has to prove that the training time stays within acceptable limits and sample classifying is fast enough to provide real-time presence detection when being deployed in a Raspberry Pi. In Table 2, both training average time and single sample average classifying time versus hidden layer neuron number can be observed when executing the algorithm in a Raspberry Pi microcontroller with the features enunciated in Sect. 3. To give a clearer idea about our system performance, Table 2 also includes the average speed (measured in frames per second, fps) the system can work at, when receiving a video stream from the Virtual PTZ.

Training average time has been calculated from the values obtained by launching the training process 90 times, each one corresponding to the same number of neurons in the hidden layer. Single sample average time and processing speed in fps have been calculated from the values obtained by passing 72 different samples through the algorithm explained above, combined with each trained neural network. As can be seen in Table 2, processing speed is approximately 50 frames per second which is an excellent frame rate for real-time video processing.

Table 2. Training average time and sample processing average time versus number of neurons in the hidden layer.

# Neurons	50	100	200	300	400	500	600
Training avg. time (s)	39.49777	26.49671	36.32062	53.55619	54.05586	62.48622	93.27951
Processing avg. time (s)	0.01959	0.01963	0.01969	0.01975	0.01982	0.01988	0.01995
Fps	51.03369	50.95048	50.78488	50.62035	50.45434	50.29447	50.13059

5 Conclusions

A microcontroller-based real-time motion detection system for video surveillance panning cameras has been proposed. It features an algorithm that processes a sequence of images streamed from a PTZ camera simulation software by dividing every image in a set of stripes and comparing each one with the equivalent stripe in the next frame in order to obtain a vector of numbers that will be fed as training, validation or test samples to a multilayer perceptron that will be in charge of pointing out whether there is movement in the video stream. With the objective of increasing system power efficiency and portability, it has been deployed in a Raspberry Pi type microcontroller.

Tests have been performed by varying the number of neurons in the hidden layer of the perceptron. They indicate that it is possible to achieve good results according to several well known classification performance measures. Time tests indicate as well that the movement detection system proposed here shows acceptable training times and when it comes to video processing, reaches processing speeds higher than 50 fps, confirming it as a valid alternative for real-time movement detection when combined with panning cameras.

Acknowledgments. This work is partially supported by the Ministry of Economy and Competitiveness of Spain under grant TIN2014-53465-R, project name Video surveillance by active search of anomalous events. It is also partially supported by the Autonomous Government of Andalusia (Spain) under projects TIC-6213, project name Development of Self-Organizing Neural Networks for Information Technologies; and TIC-657, project name Self-organizing systems and robust estimators for video surveillance. Finally, it is partially supported by the Autonomous Government of Extremadura (Spain) under the project IB13113. All of them include funds from the European Regional Development Fund (ERDF). The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the SCBI (Supercomputing and Bioinformatics) center of the University of Málaga. They also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

References

1. Adnan, L., Yussoff, Y., Johar, H., Baki, S.: Energy-saving street lighting system based on the waspmote mote. Jurnal Teknologi **76**(4), 55–58 (2015)
2. Boult, T., Gao, X., Micheals, R., Eckmann, M.: Omni-directional visual surveillance. Image Vis. Comput. **22**(7), 515–534 (2004)
3. Chen, G., St-Charles, P., Bouachir, W., Bilodeau, G., Bergevin, R.: Reproducible evaluation of pan-tilt-zoom tracking. In: Proceedings - International Conference on Image Processing (ICIP), pp. 2055–2059, December 2015
4. Ding, C., Song, B., Morye, A., Farrell, J., Roy-Chowdhury, A.: Collaborative sensing in a distributed PTZ camera network. IEEE Trans. Image Process. **21**(7), 3282–3295 (2012)
5. Dobrzynski, M.K., Pericet-Camara, R., Floreano, D.: Vision tape-a flexible compound vision sensor for motion detection and proximity estimation. IEEE Sens. J. **12**(5), 1131–1139 (2012)

6. Fung, V., Bosch, J.L., Roberts, S.W., Kleissl, J.: Cloud shadow speed sensor. Atmos. Measur. Tech. **7**(6), 1693–1700 (2014)
7. Ling, C.X., Huang, J., Zhang, H.: AUC: a statistically consistent and more discriminating measure than accuracy. In: IJCAI International Joint Conference on Artificial Intelligence, pp. 519–524 (2003)
8. Micheloni, C., Rinner, B., Foresti, G.: Video analysis in pan-tilt-zoom camera networks. IEEE Signal Process. Mag. **27**(5), 78–90 (2010)
9. Nissen, S.: Fast Artificial Neural Network (2016). http://leenissen.dk/fann/wp/. Accessed 10 Jan 2017
10. Ortega-Zamorano, F., Molina-Cabello, M.A., López-Rubio, E., Palomo, E.J.: Smart motion detection sensor based on video processing using self-organizing maps. Expert Syst. Appl. **64**, 476–489 (2016)
11. Papadimitriou, K., Dollas, A., Sotiropoulos, S.N.: Low-cost real-time 2-D motion detection based on reconfigurable computing. IEEE Trans. Instrum. Meas. **55**(6), 2234–2243 (2006)
12. Parker, C.: An analysis of performance measures for binary classifiers. In: Proceedings - IEEE International Conference on Data Mining, ICDM, pp. 517–526 (2011)
13. Song, K.T., Tai, J.C.: Dynamic calibration of pan-tilt-zoom cameras for traffic monitoring. IEEE Trans. Syst. Man Cybern. B Cybern. **36**(5), 1091–1103 (2006)