Expert Systems With Applications 64 (2016) 476–489

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Smart motion detection sensor based on video processing using self-organizing maps

Francisco Ortega-Zamorano[a,b], Miguel A. Molina-Cabello[a], Ezequiel López-Rubio[a,*], Esteban J. Palomo[a,b]

[a] *Department of Computer Languages and Computer Science, University of Málaga, Málaga, Spain*
[b] *School of Mathematics and Computer Science, University of Yachay Tech, San Miguel de Urcuquí, Ecuador*

A R T I C L E   I N F O

Article history:
Received 8 March 2016
Revised 13 July 2016
Accepted 2 August 2016
Available online 3 August 2016

Keywords:
Self-organizing map
Microcontroller
Arduino
Image processing
Block processing

A B S T R A C T

Most current approaches to computer vision are based on expensive, high performance hardware to meet the heavy computational requirements of the employed algorithms. These system architectures are severely limited in their practical application due to financial and technical limitations. In this work a different strategy is used, namely the development of an inexpensive and easy to deploy computer vision system for motion detection. This is achieved by three means. First of all, an affordable and flexible hardware platform is employed. Secondly, the motion detection algorithm is specifically tailored to involve a very small computational load. Thirdly, a fixed point programming paradigm is followed in implementing the system so as to further reduce the computational requirements. The proposed system is experimentally compared to the standard motion detector for a wide range of benchmark videos. The reported results indicate that our proposal attains substantially better performance, while it remains affordable and easy to install in practice.

1. Introduction

Motion detection is the process of detecting a change in the position of an object relative to its surroundings or a change in the surroundings relative to an object. Motion detection can be achieved by either mechanical or electronic methods, but it is most usually implemented by electronic sensors.

Motion sensors can be passive or active. Passive sensors do not emit any energy to the environment and they are the most common kind of electronic sensors. They are sensitive to a person's skin temperature through emitted blackbody radiation at mid-infrared wavelengths, in contrast to background objects at room temperature. On the other hand, active sensors emit some type of signal like light, microwave or sound into the environment and they detect some change in the behavior of the responses.

Currently new techniques are being introduced in motion detection systems with the proliferation of digital cameras capable of shooting video. Nowadays it is possible to use the output of such a camera to detect motion in its field of view using software. Motion detection is usually carried out by a software-based monitoring algorithm. When the algorithm detects motions it signals the surveillance camera to begin capturing the event. This is also called activity detection. An advanced motion detection surveillance system can analyze the type of motion to see if it warrants an alarm (García, García, Ponz, de la Escalera, & Armingol, 2014; Gómez, García, Martín, de la Escalera, & Armingol, 2015).

The Self-Organizing Map (SOM) is a kind of artificial neural network which is capable of unsupervised learning (Kohonen, 1982). Since its proposal, the SOM has been applied to knowledge discovery, data mining, detection of inherent structures in high-dimensional data and mapping these data into a two-dimensional representation space (Kohonen, 2013; Yin, 2008). This mapping retains the relationships among input data and preserves their topology. Hence this artificial neural network has had a wide range of application fields over the decades (Oja, Kaski, & Kohonen, 2003; Kaski, Kangas, & Kohonen, 1998). In particular, it has been applied to several areas of computer vision, such as color quantization (Dekker, 1994; Palomo & Domínguez, 2014; Papamarkos, 1999; Xiao, Leung, Lam, & Ho, 2012), and image segmentation (Bhandarkar, Koh, & Suk, 1997; Dong & Xie, 2005; Lacerda & Mello, 2013; Maddalena & Petrosino, 2008a). The SOM is based on an incremental (online) learning process, which has better ability to escape from local minima than batch learning (Bermejo & Cabestany, 2002) and consumes less computational time in

* Corresponding author. Fax: +34 952 13 13 97.
E-mail addresses: fortega@lcc.uma.es (F. Ortega-Zamorano), miguelangel@lcc.uma.es (M.A. Molina-Cabello), ezeqlr@lcc.uma.es (E. López-Rubio), ejpalomo@lcc.uma.es (E.J. Palomo).

http://dx.doi.org/10.1016/j.eswa.2016.08.010
0957-4174/© 2016 Elsevier Ltd. All rights reserved.

color quantization problems (Chang, Pengfei, Xiao, & Srikanthan, 2005). Moreover, it has been employed previously to detect foreground objects in video sequences (López-Rubio, Luque-Baena, & Domínguez, 2011; Maddalena & Petrosino, 2008a). However, these approaches require a SOM for each pixel of the video frame. Consequently a SOM must be trained and queried for each pixel and frame in real time as the video sequence progresses. Therefore they are not suitable for implementation on microcontrollers, which do not have the computational resources to accomplish such a complex task.

All of these schemes require a large amount of computation, which is an important challenge of computer vision systems (Casanova, Franco, Lumini, & Maio, 2013). For this reason it has been necessary the use of PCs to implement these types of learning processes, so that the resultant systems are very expensive and complex to produce on a large scale. In this work we propose changing the strategy to obtain simpler and cheaper motion detectors.

Microcontroller boards are economic, small and flexible hardware devices. They are commonly employed in important technologies such as Embedded systems (Mamdoohi et al., 2012; Marwedel, 2006), Real-time systems (Kopetz, 1997; Wang, Xu, & Gong, 2010) and Wireless sensor networks (Sengupta, Das, Nasir, & Panigrahi, 2013; Yick, Mukherjee, & Ghosal, 2008). They have a reduced amount of hardware resources and limited computing speed, not allowing extensive use of these devices in complex tasks. However, recent advances in the computing power of microcontrollers and a change in their programming paradigm allows the inclusion of learning schemes in the device (“on-chip” learning), adapting their behavior dynamically according to the sensed data (Aleksendrić, Jakovljević, & Irović, 2012; Mahmoud, Lotfi, & Langensiepen, 2013; Ortega-Zamorano, Jerez, Subirats, Molina, & Franco, 2014).

Microcontrollers 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 (Papadimitriou, Dollas, & Sotiropoulos, 2006). A flexible Printed Board Circuit (PCB) prototype which integrates a microcontroller has been proposed to estimate motion and proximity (Dobrzynski, Pericet-Camara, & Floreano, 2012). 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 (Fung, Bosch, Roberts, & Kleissl, 2014). 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. Finally, 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 (Adnan, Yussoff, Johar, & Baki, 2015). This way, the street lamps are switched on when people are present in their surroundings.

In the present work, we have fully implemented the SOM in an Arduino DUE board, including the whole learning process to implement the automatic motion detection process for decision-making into the detector in all types of environments, avoiding offline computation and communication to other devices.

The Arduino DUE board was used (Oxer & Blemings, 2009) as it is a popular, economic and efficient open source single-board microcontroller that allows easy project development (Cela et al., 2013; Kornuta, Nipper, & Brandon Dixon, 2012; Lian, Hsiao, & Sung, 2013; Ortega-Zamorano, Jerez, Urda Munoz, Luque-Baena, & Franco, 2015). We also propose a change in the data type representation used in the programming of the Arduino from the floating point representation commonly employed in this type of system to fixed point representation, in order to obtain a faster system with less hardware resources. This enables the utilization of the SOM in this kind of device.

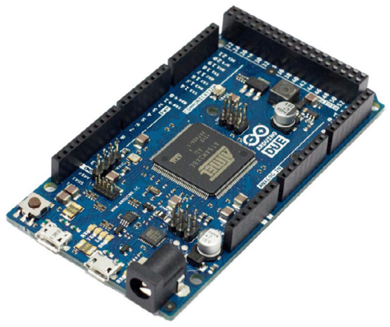
Fig. 1. Picture of an Arduino DUE board used for the implementation of the SOM-based motion detection model.

The paper is structured as follows. In Section 2 the microcontroller system is briefly described, and our fixed point programming approach is outlined. Then we introduce a new motion detection model including the SOM, which is specifically designed to meet the computation capabilities of microcontrollers (Section 3). Section 4 explains the details of the implemented application. The obtained experimental results are reported in Section 5. Finally, conclusions are extracted in Section 6.

2. Microcontroller (μC) system description

We have implemented the SOM-based motion detection model in an Arduino DUE microcontroller. The details of the implemented system are described below, with an emphasis on the comparison between using a fixed point representation or a floating point one. Section 2.1 describes the Arduino hardware, Section 2.2 gives an overall view of the motion detection software, and Section 2.3 discusses the options to implement arithmetic operations.

2.1. The Arduino board

Arduino is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible (Oxer & Blemings, 2009). The hardware consists of a simple open source board designed around an 32-bit Atmel ARM core microcontroller, and the software includes a standard programming language compiler that runs in a standard PC and a boot loader for loading the compiled code on the microcontroller. Arduino is a descendant of the open-source *Wiring* platform and is programmed using a Wiring-based language (syntax and libraries), similar to C++ with some slight simplifications and modifications, and a processing-based integrated development environment. Arduino boards can be purchased pre-assembled or do-it-yourself kits, and hardware design information is available. The maximum length and width of the Arduino UNO board are 10.2 and 5.3 cm respectively, with the USB connector and power jack extending beyond the former dimension.

The Arduino DUE is based on the SAM3X8E ARM Cortex-M 3 CPU (Atmel), and it has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, four UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, two DAC (digital to analog), and a reset and erase buttons. The SAM3X has 512 KB (two blocks of 256 KB) of flash memory for storing code, it also comes with a preburned bootloader that is stored in a dedicated ROM memory. The available SRAM amounts to 96 KB in two contiguous banks of 64 and 32 KB. A picture of the Arduino DUE board is shown in Fig. 1.

The Arduino Due has a number of facilities for communicating with a computer, another Arduino or other microcontrollers,

Table 1
Computation time required for the basic arithmetic operations depending on the variable type (integer, float and double) used in the Arduino DUE microcontroller.

Variable	Basic operations			
Type	+	−	∗	/
Integer	59.8	59.8	71.7	99.9
Float	3965.4	4146.9	3751.8	5269.8
Double	5113.2	5139.4	4763.3	13635.2

and different devices like phones, tablets, cameras and so on. The SAM3X provides one hardware UART and three hardware USARTs for TTL (3.3V) serial communication.

2.2. Initialization and execution phases of the algorithm

The implementation of the proposed SOM-based motion detection model comprises two phases: the initialization phase which generates the initial state of the model, and the execution phase in which the microcontroller updates the model and makes decisions according to the input data. The input video frame is divided into several non overlapping pixel blocks, so that a SOM model is associated to each block.

The initialization phase generates the initial state of the SOM model associated to each pixel block. In order to do this, the prototypes of all the neurons of the SOM are initialized to the average color of the pixels which belong to the pixel block in the first incoming video frame.

The execution phase has been divided into two different processes: the learning process and the decision process. For each pixel block, the learning process summarizes the color information from all the pixels of the block into an input vector which is supplied as a training sample to the SOM associated to the block. Then the decision process estimates whether each individual block contains moving objects with the help of the SOM model associated to the block. More details about the learning and decision processes are given in Section 3.

2.3. Data type representation

Microcontrollers are devices with limited computing power so in order to speed up the learning process, it was decided to employ a fixed-point data type representation. Please note that floating-point is the most commonly employed data type representation in this kind of device, but this representation is not always the most efficient one.

The change of the paradigm in the data type representation involves a change in the type of variables used in the software implementation of the SOM model. The floating-point representation is stored in a “float” or “double” variable with a size of 4 or 8 bytes respectively, depending on the precision that is required. On the other hand, the fixed-point representation is stored in an “integer” variable with a size of 4 bytes.

This paradigm shift involves profound changes in the way the SOM model is programmed but in return it offers a faster learning process and a smaller size representation of variables. Table 1 shows the computational time (in μs) required for the calculation of each basic arithmetic operation { + , - , ∗ , / } with the mentioned three variable types (integer, float and double) in the Arduino DUE microcontroller.

3. Motion detection model

The motion detection system proposed in this work is based on the subdivision of the input frame into several non overlapping rectangular blocks of the same size. A color model is learned for each block by means of a SOM, so that color anomalies can be measured in each region separately. Then the color anomalies are analyzed so as to determine whether they are associated to moving foreground objects. Section 3.1 explains the frame subdivision arrangement, Section 3.2 describes the self-organizing map model, and Section 3.3 details how to analyze the measured anomalies. Section 3.4 gives details about the storage of the SOMs. The employed algorithm to compute the exponential function is explained in Section 3.4. Finally, Section 3.6 is devoted to compare the fixed point and floating point implementations of the proposed model.

3.1. Frame subdivision

Most current approaches to motion detection either build a color model for each pixel (Bouwmans, 2014b). However, this is not feasible for microcontrollers due to their hardware limitations. Therefore we propose to use a subdivision of the input frame into non overlapping rectangular blocks, so that a color model is learned for each region.

Let us assume that the input video frames have size $N_{row} \times N_{col}$ pixels, and that for each pixel a RGB color vector $\mathbf{y_h} \in [0, 1]^3$ is obtained from the camera, where $\mathbf{h} \in \{1, \dots, N_{row}\} \times \{1, \dots, N_{col}\}$ are the pixel coordinates. Then the input frame is divided into $B_{row} \times B_{col}$ non overlapping blocks each of size $\frac{N_{row}}{B_{row}} \times \frac{N_{col}}{B_{col}}$ pixels, where N_{row} is an integer multiple of B_{row} and N_{col} is an integer multiple of B_{col}.

For each block it is necessary to summarize the color information provided by the pixel color data $\mathbf{y_h}$ in a fast way. Here we propose to compute the average color of each block:

$$\mathbf{x_r} = \frac{1}{N_{block}} \sum_{\mathbf{h} \in \mathcal{B}_\mathbf{r}} \mathbf{y_h} \tag{1}$$

where $\mathbf{x_r} \in [0, 1]^3$, $\mathcal{B}_\mathbf{r}$ is the set of the pixels which belong to the block with coordinates $\mathbf{r} \in \{1, \dots, B_{row}\} \times \{1, \dots, B_{col}\}$, and N_{block} is the number of pixels per block:

$$N_{block} = \frac{N_{row} N_{col}}{B_{row} B_{col}} \tag{2}$$

For each incoming video frame and block, the average block color $\mathbf{x_r}$ is provided to the self-organizing map associated to the block as an input training sample, as seen next.

3.2. Self-organizing map

Next we are going to describe Kohonen's SOFM model which is used to learn a color model of a block of the input frame. Let M be the number of neurons of the self-organizing map associated to a certain block of the input frame. The neurons are arranged in a lattice of size $a \times b$, where $M = ab$. The topological distance between the neurons i and i', located at positions $(y_1, y_2) \in \mathbb{N}^2$ and $(y'_1, y'_2) \in \mathbb{N}^2$ in the lattice space, is given by:

$$d(i, i') = \sqrt{(y_1 - y'_1)^2 + (y_2 - y'_2)^2} \tag{3}$$

Every neuron i has a prototype vector $\mathbf{w}_i$ which represents a cluster of input samples. Please note that $\mathbf{w}_i \in [0, 1]^3$, where we consider three-dimensional real valued vector inputs which codify colors in the RGB color space. At time step n, a new sample $\mathbf{x}(n)$ which represents the average color for the block is presented to the network, and a winner neuron is declared:

$$Winner(\mathbf{x}(n)) = \arg \min_{j \in \{1, \dots, M\}} \left\| \mathbf{x}(n) - \mathbf{w}_j(n) \right\| \tag{4}$$

Then the prototypes of all the units are adjusted, for $i \in \{1, \dots, M\}$:

$\mathbf{w}_i(n+1) =$

$$\mathbf{w}_i(n) + \eta(n)\Lambda(i, Winner(\mathbf{x}(n)))(\mathbf{x}(n) - \mathbf{w}_i(n)) \tag{5}$$

where $\eta(n)$ is a decaying learning rate and the neighborhood function Λ varies with the time step n and depends on a decaying *neighborhood radius* $\Delta(n)$:

$$\eta(n+1) \leq \eta(n) \tag{6}$$

$$\Lambda(i, Winner(\mathbf{x}(n))) =$$

$$\exp\left(-\left(\frac{d(i, Winner(\mathbf{x}(n)))}{\Delta(n)}\right)^2\right) \tag{7}$$

$$\Delta(n+1) \leq \Delta(n) \tag{8}$$

At initialization time $n = 0$, each prototype $\mathbf{w}_i$ is set to the observed sample $\mathbf{x}(0)$ for $i \in \{1, \ldots, M\}$:

$$\mathbf{w}_i(0) = \mathbf{x}(0) \tag{9}$$

The receptive field of neuron i, i.e. the region of the input space which is represented by i, is defined as:

$$F_i = \left\{\mathbf{x} \in \mathbb{R}^D \mid i = Winner(\mathbf{x})\right\} \tag{10}$$

Vector quantization is one of the main goals of self-organizing maps. We are interested in the quantization error q_k associated to an input $\mathbf{x}_k$:

$$q_k = \min_{j \in \{1,\ldots,M\}} \left\|\mathbf{x}_k - \mathbf{w}_j\right\| \tag{11}$$

The global performance of a map for this task is commonly measured by the mean squared error (Beaton, Valova, & MacLean, 2010; Dlugosz, Talaska, Pedrycz, & Wojtyna, 2010; Hsu & Halgamuge, 2003; Yin, 2008):

$$MSE = \frac{1}{K}\sum_{k=1}^{K} q_k^2 \tag{12}$$

3.3. Anomaly analysis

As the self-organizing map associated to each block learns the color information corresponding to that block, it is possible to estimate whether the block contains a substantial part of moving objects. This is done in a fast way by considering the quantization error $q_{n,\mathbf{r}}$ at time step n of the current average block color $\mathbf{x}_{n,\mathbf{r}}$, as represented by the self-organizing map associated to the block:

$$q_{n,\mathbf{r}} = \min_{j \in \{1,\ldots,M\}} \left\|\mathbf{x}_{n,\mathbf{r}} - \mathbf{w}_{j,n,\mathbf{r}}\right\| \tag{13}$$

where $\mathbf{w}_{j,n,\mathbf{r}} \in [0, 1]^3$ stands for the prototype at time step n of the j-th neuron of the self-organizing map associated to the block with coordinates $\mathbf{r}$. The block is declared to contain moving objects if and only if $q_{n,\mathbf{r}}$ is above a threshold T:

$$\text{Block } \mathbf{r} \text{ contains moving objects} \Leftrightarrow q_{n,\mathbf{r}} > T \tag{14}$$

where $T > 0$ is a tunable parameter of the system. The rationale behind this is that moving objects usually have a color which differs significantly from the background color.

3.4. SOM model storage

The number of bytes used for representing the SOM model in each pixel block depends on the data type representation. Employing fixed-point representation allows using 32 bits for each variable since they are stored in an “integer” variable. In this case as the SOM model values range between 0 and 1, the precision of this type of variables is $2^{-32} = 2.328 \cdot 10^{-10}$.

Taking into account that the available SRAM memory amounts 96 KB to store all variables of the algorithm and that the SOM models are the most memory consuming variables, the SRAM memory has been divided in two parts. One part stores the SOM models of all pixel blocks with 80 KB, and the other part comprises the rest of the variables involved in the execution of the algorithm with 16 KB. Therefore, the maximum number of pixel blocks that can be stored in the implemented system is given by the following equation:

$$N_{blocks} \leq \frac{80KB}{M \cdot 4Byte}, \tag{15}$$

where $N_{blocks} = B_{row}B_{col}$ is the number of pixel blocks and $M = a \times b$ is the number of neurons in the SOM of each pixel block, which has been set to $M = 3 \times 4 = 12$ because it offers a good tradeoff between the ability of the SOMs to represent complex input color distributions and the computational load required to train the SOMs.

Therefore the maximum number of pixel blocks that can be stored is 1706 pixel blocks for variables with a size of 4 bytes.

3.5. Computation of the exponential function

The computation of the exponential function can be carried out using the specific ALU (Arithmetic and Logic Unit) by means of the specific library “math.h” in order to evaluate the exponential functions involved in the model. The computational time for this procedure is equal to 58.9 μs in the used microcontroller.

An approximation to carry out the exponential function has been implemented in order to reduce the computational time for this function. This reduction in the computation time allows updating more pixel blocks in a given time, thereby increasing the maximum number of pixel blocks that can be processed.

The approximation has been performed by a table lookup followed by a linear interpolation. This method has been extensively studied in previous works (Ortega-Zamorano, Jerez, Juarez, Perez, & Franco, 2014).

The look-up table contains the values of the exponential function for equispaced values of the independent variable. Nevertheless, as high precision values are needed for the correct execution of the algorithm, the computation of the function approximation is further complemented by a linear interpolation procedure using two adjacent tabulated values (lower and larger) with respect to the input value of the independent variable. In this case the required computation time is reduced to 1.437 μs, which means a 97.5% reduction in comparison to the specific library “math.h”.

Storing table values requires large amounts of memory depending on the accuracy of the approximation. Fig. 2 shows the necessary memory size according to the accuracy of the approximation of the optimized method based on a lookup table plus linear interpolation of adjacent values.

A maximum absolute error lower than $5 \cdot 10^{-5}$ has been selected. Therefore the memory size necessary to store the lookup table is equal to 4 KB.

Fig. 3 shows the absolute error involved in the computation of the negative exponential function in the range from 0 to 16.

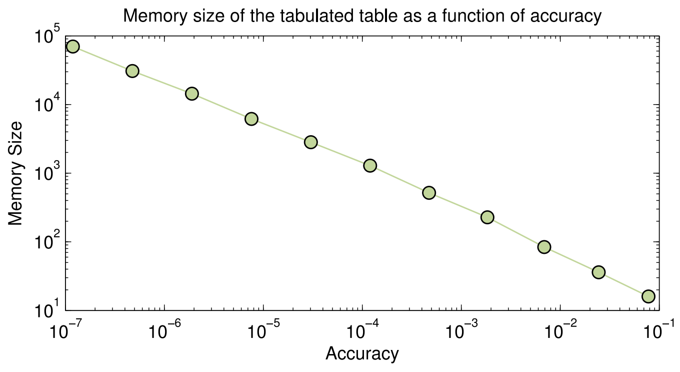

Fig. 2. Required memory size according to the accuracy of the approximation of the optimized method based on a lookup table plus linear interpolation of adjacent values.

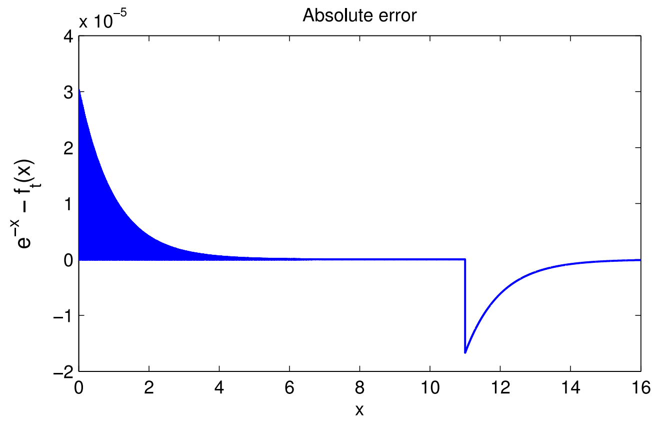

Fig. 3. Absolute error committed in the approximation of the exponential function (see text for more details).

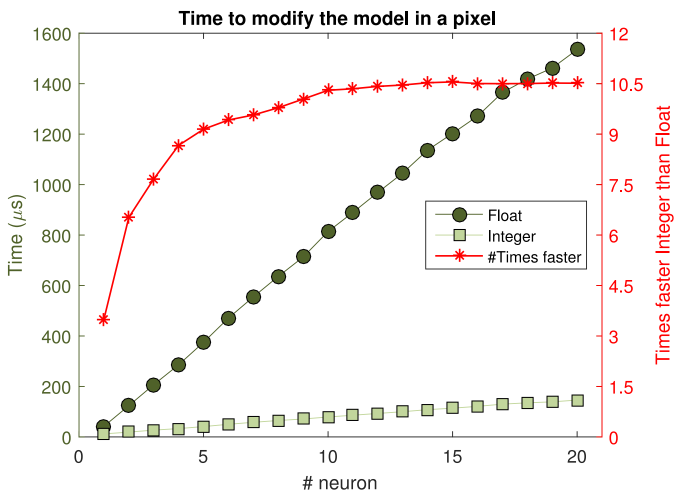

Fig. 4. Computation time (left y-axis) in μs required to update the SOM model of a pixel block with different implementations of variables (Integer and Float) and the number of times (right y-axis) that "integer" variables are faster than "float" variables as a function of the number of neurons of the model.

3.6. Fixed point vs floating point representation comparison

Fig. 4 shows the computation time (left y-axis) in μs required to update the SOM model of a single pixel block when it is implemented with variables of "integer" and "float" type and the number of times (right y-axis) that the "integer" variables are faster than "float" variables as a function of the number of neurons of the SOM model.

The computation time for updating a pixel block places an upper bound on the number of pixels that the system can update in real-time. Nowadays real-time operation for computer vision means that a single frame must be processed within 30–40 ms (Pulli, Baksheev, Kornyakov, & Eruhimov, 2012). Since the video camera used for the experiments acquires 30 frames per second, which is 33.33 ms per frame, it can be considered that for real time operation a full video frame must be processed in less than 1/30 s. This way, the maximum number of pixel blocks that can be processed in the implemented system is given by the following equation:

$$N_{blocks} \le \frac{0.03333(s)}{T_{up}}, \tag{16}$$

where N_{blocks} is the number of pixel blocks and T_{up} is the computation time for updating the SOM model of a pixel block.

As the number of neurons (M) has been set to 12, T_{up} is equal to 93 μs for "integer" variables and 969 μs for "float" variables. Therefore the maximum number of pixel blocks that can be updated is 358 blocks for "integer" variables and 34 blocks for "float" variables.

4. Application

Intrusion detection systems are widely used in all types of premises from households to public buildings, so that there are many contexts where this kind of system could be deployed. We have focused on making the system easy to replicate in order to be able to have multiple motion detectors. In particular, we have focused on low cost and low power consumption.

The proposed system is composed of a camera to obtain the image of the scene and a microcontroller to decide the existence of unusual movement in the scene. Both the microcontroller and the camera have been selected to be devices of low cost and low power consumption. The selected microcontroller is the Arduino DUE (see Section 2.1) and the video camera used in the application is the C429-RS232, a highly integrated, compact serial and encoded video camera module. The module uses an OmniVision^{TM} CMOS MT9V011 VGA color sensor, matched with a Vimicro VC0706 control chip to provide a complete low cost, low power camera system. It has an on-board RS232 serial interface for direct connection to a microcontroller. Serial transfer rate is at 115.2 Kbps for transferring color or monochrome images in VGA (640 × 480), QVGA (320 × 240), or QQVGA (160 × 120) resolution. Real-time video output is provided at 30 fps as CVBS signal, NTSC or PAL. C429-RS232 needs only 80 mA from a 5 V power supply.

Fig. 5 illustrates the tasks that are carried out as an incoming video frame is processed. The five pictures on the left side depict the processing of a video frame with no intrusion detected, while the five pictures on the right side correspond to a video frame where an intrusion is detected.

The pictures on the first row show the RGB image that is captured by the video camera. These pictures are the starting point of the intrusion detection process.

The pictures on the second row show the downsized images that are sent to the microcontroller to be processed. The maximum size of the downsized images is determined by Eq. 16. However, in this case the selected size is 12 × 16 = 192 blocks in order to demonstrate that smaller sizes also can be used for this application with no significant decrease of the efficiency. With this number of blocks, real time operation is attained since the number of blocks is below the upper limit for real time which is 358 blocks, as explained before.

The third row shows the SOM models for every pixel (see Section 3.2). For each of the 12 × 16 blocks of the downsized images a mosaic is shown with 3 × 4 small rectangles, where each

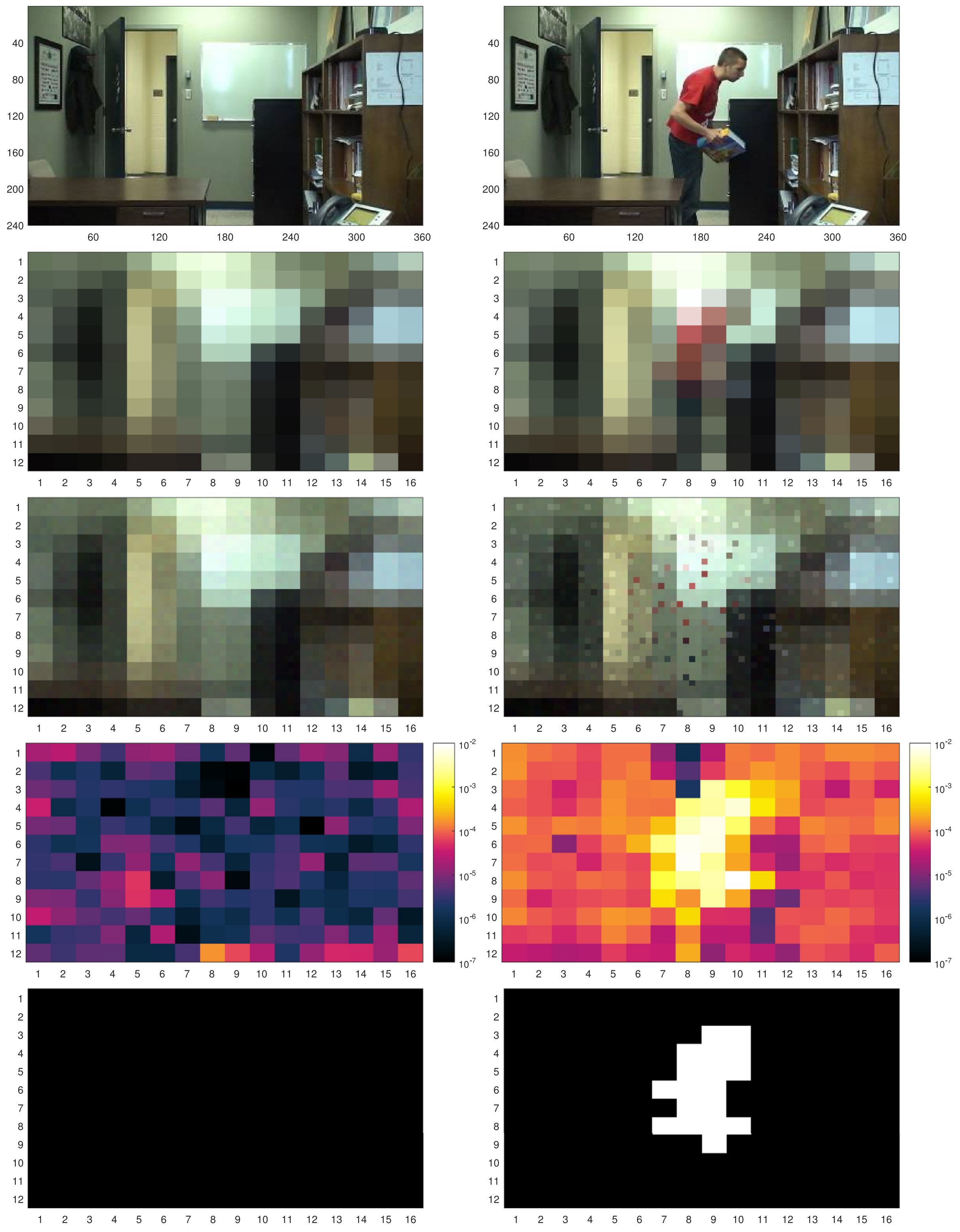

Fig. 5. The steps carried out in the imaging processing of a frame for a detection of intrusion (the five pictures in the right side) and another scene without detection(the five pictures in the left side). (see text for more details).

small rectangle represents the prototype of a neuron of the SOM associated to the block. This is because the used SOM networks have a rectangular topology with $3 \times 4 = 12$ neurons. The small rectangles show the prototype of the associated neuron as a RGB color. It can be observed that the neurons associated to the same block are quite similar on the left side when the intrusion has not been detected yet. On the other hand, the neurons of the same block are significantly different on the right side when the intrusion is being detected. This is because some neurons learn the colors of the intruding object.

The fourth row (second row from the bottom) shows the quantization error of each input block of the captured pictures given by

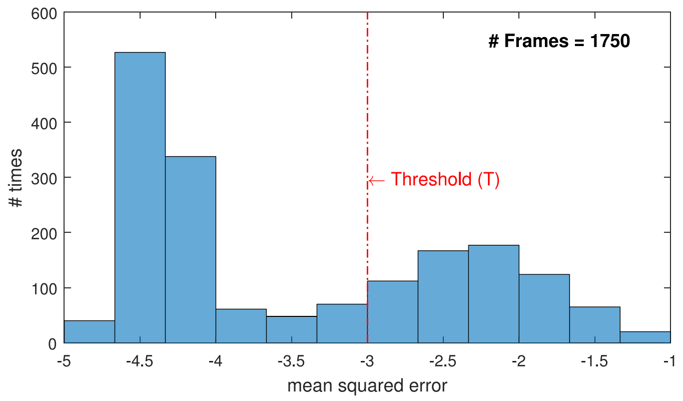

Fig. 6. The histogram of the maximum quantization error of all blocks for a video sequence which comprises 1750 frames. The threshold which separates the quantization errors for blocks with no movement (left) and blocks with movement (right) is shown as a vertical dash-dot line.

Eq. (13). It can be observed that the maximum and the minimum tones used in the colorbar are the same for both pictures, so that the shown quantization errors can be compared. As seen, the video frame without any intruding objects (left) yields smaller quantization errors than the video frame with an intruding object (right).

The fifth and last row shows the decision whether each block contains moving objects, as calculated from Eq. (14). The decision is made depending on the value of quantization error in every time instant. If the error is higher than a determined *"threshold"* (T), then the block is declared as foreground, i.e. a block with a detected intrusion. In the pictures the blocks which exhibit a quantization error higher than the threshold are painted in white, while the blocks with quantization errors lower than the threshold are painted in black. A frame is declared to contain an intrusion in a video scene whenever two or more blocks have a quantization error higher than the threshold.

In order to determine the value of the threshold we have analyzed a video sequence and we have obtained the histogram of the maximum quantization errors for all the blocks of a frame. In Fig. 6 it can be observed that the histogram exhibits two modes. The left mode corresponds to absence of intrusion, while the right mode is associated to intrusions. As seen, the majority frames without intrusion detection have a quatization error in the range from 10^{-4} to 10^{-5}, while for the frames with intrusion detection the quantization error is around 10^{-2}. For this reason we have selected the value of the threshold (T) as 10^{-3}.

5. Results

In this section, we have tested the implemented system for different well-known benchmark videos (Wang et al., 2014) in order to demonstrate the utility of the proposed scheme. Each raw RGB video comes with an associated "ground truth" black and white video sequence which establishes which regions of each frame actually correspond to moving objects. The ground truth video is only used to measure the detection performance of the competing approaches; it is not provided to the detection systems in any way. In order to obtain replicable results with known videos, we have connected the Arduino microcontroller by the USB port with a serial communication to the PC. Under this configuration the PC is programmed to simulate a camera. That is, the PC sends the video in the same way that the camera does, so that the microcontroller does not notice the difference.

The traditional detector can be implemented in different ways (Naghiyev, Gillott, & Wilson, 2014; Park et al., 2015), although the most useful has frequently been passive infrared sensors (PIR). The traditional detector looks for abrupt changes in the global illuminance of the scene $\bar{y}$:

$$\bar{y} = \frac{1}{3N_{row}N_{col}} \sum_{\mathbf{h}\in\mathcal{B}_\mathbf{r}} \left(y^1_\mathbf{h} + y^2_\mathbf{h} + y^3_\mathbf{h}\right) \tag{17}$$

where $y^j_\mathbf{h}$ is the value of the jth color channel at the pixel with coordinates $\mathbf{h}$.

In order to measure the changes in $\bar{y}$ across time steps n, the time average of $\bar{y}$ can be estimated at time n as follows:

$$\hat{y}(n+1) = \hat{y}(n) + \eta(n)\left(\bar{y}(n) - \hat{y}(n)\right) \tag{18}$$

where $\eta(n)$ stands for the learning rate already introduced in (5). At initialization time $n = 0$ the estimation is set to the observed global illuminance:

$$\hat{y}(0) = \bar{y}(0) \tag{19}$$

Motion is detected at time n whenever the current global illuminance $\bar{y}(n)$ differs from the estimated average $\hat{y}(n)$ by more than a threshold T:

$$e_n = \left|\bar{y}(n) - \hat{y}(n)\right| \tag{20}$$

$$\text{The frame contains moving objects} \Leftrightarrow e_n > T \tag{21}$$

Furthermore, we have selected some reference pixel-level foreground detection methods from previous literature which have a public and reasonably well tested implementation, in order to carry out comparisons with them. These methods have been run on a standard PC with a 3 GHz CPU and 8 GB RAM, since they are too computationally demanding to be executed on a microcontroller. The first algorithm we have considered is the method we note as WrenGA (Wren, Azarbayejani, Darrell, & Pentl, 1997), which is the oldest one and features a single Gaussian probabilistic model. Other chosen Gaussian methods are GrimsonGMM (Stauffer & Grimson, 1999), that uses two Mixture of Gaussians; and the ZivkovicGMM (Zivkovic, 2004; Zivkovic & van der Heijden, 2006) method, which has a non-fixed number of Gaussian distributions. Additionally, an artificial neural networks approach method noted MaddalenaSOBS (Maddalena & Petrosino, 2008b) is also considered. These tested methods are available on BGS library version 1.3.0, which is accessible from its website[1]. In addition, we have selected the MFBM (López-Rubio & López-Rubio, 2015) method which is based on the stochastic approximation theory and was recently published by our research group. The tuned values of each method are selected from the authors' recommendations. They are shown in Table 2.

It must be pointed out that the motion detection problem that the traditional and the proposed detectors aim to solve is a binary classification problem. The positive class is formed by those video frames where there are moving objects. On the other hand, the negative class comprises those video frames where no moving objects exist. Therefore we have considered the binary classification performance measures recommended in the CDnet 2014[2] motion detection benchmark:

$$Recall = \frac{TP}{TP+FN} \tag{22}$$

$$Specificity = \frac{TN}{TN+FP} \tag{23}$$

$$FPR = \frac{FP}{FP+TN} \tag{24}$$

[1] https://github.com/andrewssobral/bgslibrary.
[2] http://www.changedetection.net.

Table 2
Considered parameter values for the competing methods, forming the set of experimental configurations.

Method	Parameters
MFBM	Features, $F = \{1, 2, 3\}$
	Step size, $\alpha = 0.01$
GrimsonGMM	Threshold, $T = 12$
	Learning rate, $\alpha = 0.0025$
	Number of Gaussians in the mixture model, $K = 3$
MaddalenaSOBS	Sensitivity, $s_1 = 75$
	Training sensitivity, $s_0 = 245$
	Learning rate, $\alpha_1 = 75$
	Training step, $N = 100$
WrenGA	Threshold, $T = 12$
	Learning rate, $\alpha = 0.005$
ZivkovicGMM	Learning rate, $\alpha = 0.001$
	Number of Gaussians components, $K = 3$
	Threshold, $T = 30$

$$FNR = \frac{FN}{TP + FN} \tag{25}$$

$$PBC = 100\frac{FN + FP}{TP + FP + FN + TN} \tag{26}$$

$$Precision = \frac{TP}{TP + FP} \tag{27}$$

$$F - Measure = 2\frac{Recall \cdot Precision}{Recall + Precision} \tag{28}$$

It must be noted that, since the aim of our work is global motion detection, the above performance measures have been computed at the frame level and not at the pixel level. To this end, a frame is declared to contain movement when the fraction of pixels which belong to foreground objects is higher than $\frac{2}{192}$, which is the same criterion considered in Section 4.

The quantitative performance results are reported as follows. Tables 3 and 4 present the frames per second and the number of executed instructions per frame of each tested method over the evaluated videos, respectively. It can be seen that the pixel-level methods (last five columns) do not attain real time operation, even if they are executed on a standard PC. On the other hand, the Arduino based approaches can run in real time, including our proposal. Each Arduino based approach has the same computational requirements for all the videos, because these methods execute the same instructions for a given frame size, independently of the content of the video. Moreover, under Arduino there is no virtual memory or any other source of variability in the running time.

The recall, the specificity, the false positive rate (FPR) and the false negative rate (FNR) of the competing methods over the tested sequences are reported in Tables 5–8, respectively. Finally, the probability of bad classification (PBC), the precision and the F-measure of the tested methods over the performed sequences are detailed in Tables 9–11, respectively. From all these selected measures, the F-measure can be regarded as a reliable overall evaluation of a method, as it characterizes the performance of a classifier in the precision-detection rate space (Bouwmans, 2014a). As it can be observed our proposal obtains the best average result in terms of F-measure. As seen in Tables 7 and 8, Ilu attains a low rate of false positives, but it has a large rate of false negatives, and this imbalance hampers its performance (Table 11). On the other hand, FrameDiff has a very low rate of false negatives, but it has a very high rate of false positives, and again this produces a rather bad overall performance.

Another aspect to be pointed out is that the selected pixel based methods from the literature (the last five columns of Tables 3–11) obtain similar results in most cases. This happens because these methods are designed to detect foreground objects at the pixel level, which is different from motion detection at the frame level. Motion detection at the frame level is a simpler problem, so pixel level accuracy is not necessary to attain a good performance. We must also mention that the inherent characteristics of each tested video have a high impact in the outcomes.

In order to have a more accurate assessment of the quantitative performance of the approaches, Fig. 7 shows the receiver operating curves (ROC) for the proposed system (a), the traditional and frame difference detectors running on Arduino (b-c) and the competing methods running on a standard PC (d–g) for all videos. The plots represent the dependence between the true positive rate (TPR, higher is better), also known as recall or sensitivity, and the false positive rate (FPR, lower is better), also known as fall-out, at various threshold settings. Please note that a perfect classification would correspond to the upper left corner of the plots. Also, we have calculated the Area Under Curve (AUC, higher is better) as a single measure of the quality of a binary classifier, since it is the probability that a randomly selected positive case will receive a higher score than a randomly selected negative case. Our SOM based approach consistently attains much better results than the other Arduino based detectors, while its performance is similar to those of the PC based detectors.

Finally, from a qualitative point of view, Figs. 8 and 9 depict the detection decisions for each competing method in several frames of a sequence for the “Pedestrians” and “Sofa” videos, respectively. The first row shows five frames of the real sequence captured by the video camera, and the second row shows the detection decision for each block in these five frames for the proposed system according to Eq. (14). The third and row show the decision for a traditional sensor running on the Arduino board, taking into account that a whole frame in black means no motion detected (negative class) and a whole frame in white means motion detected (positive class). The fourth row corresponds to the frame difference algorithm running on the Arduino board. The remaining rows rep-

Table 3
Maximum frames per second of the competing methods over the tested sequences (higher is better). Please note that SOM, Ilu and FrameDiff run on the Arduino DUE in real time, while the rest of the methods run on a standard PC. Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	56.0036	**347.2222**	62.7510	8.7140	18.0037	12.6464	19.0158	32.3135
PETS2006	56.0036	**347.2222**	62.7510	2.0630	4.1255	2.7120	5.9493	5.9392
Highway	56.0036	**347.2222**	62.7510	9.4857	18.7410	14.2416	20.0945	20.0905
Pedestrians	56.0036	**347.2222**	62.7510	9.8522	20.4964	12.6251	28.9922	30.7618
Sofa	56.0036	**347.2222**	62.7510	10.5132	20.2561	15.3184	25.0976	25.4189
Canoe	56.0036	**347.2222**	62.7510	9.3113	20.1149	14.3254	30.5428	31.4138
Fountain02	56.0036	**347.2222**	62.7510	6.3019	12.9921	8.8917	14.5588	22.4193
Fall	56.0036	**347.2222**	62.7510	2.5021	4.4226	3.7189	7.2360	6.7668
Average	*56.0036*	***347.2222***	*62.7510*	*7.3429*	*14.8940*	*10.5599*	*18.9359*	*21.8905*

Table 4
Number of executed instructions per frame of the competing methods over the tested sequences (in millions, lower is better). Please note that SOM, Ilu and FrameDiff run on the Arduino DUE in real time, while the rest of the methods run on a standard PC. Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	1.0483	**0.0873**	0.8235	294.1926	192.4942	299.4168	125.5993	111.4860
PETS2006	1.0483	**0.0873**	0.8235	1,341.5196	1,023.0307	1,478.7173	600.2945	540.3325
Highway	1.0483	**0.0873**	0.8235	263.7813	199.5114	266.5234	112.5730	103.7825
Pedestrians	1.0483	**0.0873**	0.8235	296.0330	187.0545	307.9948	125.8543	112.0348
Sofa	1.0483	**0.0873**	0.8235	263.0099	185.8824	269.5781	111.4275	100.1956
Canoe	1.0483	**0.0873**	0.8235	265.6015	190.8002	260.7247	113.1358	106.8618
Fountain02	1.0483	**0.0873**	0.8235	419.9596	305.8514	439.8973	179.6190	159.1311
Fall	1.0483	**0.0873**	0.8235	1,158.3201	975.5466	1,102.7957	492.2767	468.2882
Average	*1.0483*	***0.0873***	*0.8235*	*537.8022*	*407.5214*	*553.2060*	*232.5975*	*212.7641*

Table 5
Recall of the competing methods over the tested sequences (higher is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	0.8544	0.2088	0.9905	0.9905	0.9845	0.9928	**0.9940**	0.9606
PETS2006	0.4715	0.0976	**0.9009**	0.7293	0.7955	0.7789	0.4060	0.4632
Highway	0.9304	0.4485	0.9982	0.8670	0.9789	**1.0000**	0.9700	0.9665
Pedestrians	0.9540	0.1206	**0.9889**	0.9222	0.9587	0.9683	0.9206	0.9413
Sofa	0.9150	0.4511	**0.9985**	0.9872	0.9737	0.9962	0.9714	0.9714
Canoe	0.9637	0.3387	**1.0000**	0.9032	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Fountain02	0.7192	0.0000	0.9557	0.5616	0.8177	**1.0000**	0.6305	0.5123
Fall	0.6371	0.5747	0.9966	0.7323	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Average	*0.8057*	*0.2800*	***0.9787***	*0.8367*	*0.9386*	*0.9670*	*0.8616*	*0.8519*

Table 6
Specificity of the competing methods over the tested sequences (higher is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	0.5810	**0.9866**	0.0126	0.0140	0.1326	0.0140	0.0140	0.1732
PETS2006	0.9944	0.9983	0.3952	0.9915	0.9786	0.9957	**1.0000**	**1.0000**
Highway	0.9753	**1.0000**	0.0229	**1.0000**	0.9681	0.3617	0.9787	0.9787
Pedestrians	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Sofa	**0.8309**	0.8138	0.0024	0.4707	0.6772	0.1511	0.6413	0.6402
Canoe	**1.0000**	0.9266	0.0000	**1.0000**	0.0000	0.0000	0.0000	0.0000
Fountain02	0.9500	0.9951	0.0206	**1.0000**	0.9384	0.0000	**1.0000**	**1.0000**
Fall	0.5834	0.4162	0.0022	**0.8870**	0.0000	0.0000	0.0000	0.0000
Average	*0.8644*	***0.8921***	*0.1827*	*0.7954*	*0.5869*	*0.3153*	*0.5793*	*0.5990*

Table 7
False positive rate (FPR) of the competing methods over the tested sequences (lower is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	0.4190	**0.0134**	0.9874	0.9860	0.8674	0.9860	0.9860	0.8268
PETS2006	0.0056	0.0016	0.6048	0.0085	0.0214	0.0043	**0.0000**	**0.0000**
Highway	0.0247	**0.0000**	0.9770	**0.0000**	0.0319	0.6383	0.0213	0.0213
Pedestrians	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Sofa	**0.1691**	0.1862	0.9976	0.5293	0.3228	0.8489	0.3587	0.3598
Canoe	**0.0000**	0.0734	1.0000	**0.0000**	1.0000	1.0000	1.0000	1.0000
Fountain02	0.0500	0.0049	0.9794	**0.0000**	0.0616	1.0000	**0.0000**	**0.0000**
Fall	0.4166	0.5838	0.9978	**0.1130**	1.0000	1.0000	1.0000	1.0000
Average	*0.1356*	***0.1079***	*0.8180*	*0.2046*	*0.4131*	*0.6847*	*0.4207*	*0.4010*

Table 8
False negative rate (FNR) of the competing methods over the tested sequences (lower is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	0.1456	0.7912	0.0095	0.0095	0.0155	0.0072	**0.0060**	0.0394
PETS2006	0.5285	0.9024	**0.0991**	0.2707	0.2045	0.2211	0.5940	0.5368
Highway	0.0696	0.5515	0.0018	0.1330	0.0211	**0.0000**	0.0300	0.0335
Pedestrians	0.0460	0.8794	**0.0111**	0.0778	0.0413	0.0317	0.0794	0.0587
Sofa	0.0849	0.5489	**0.0015**	0.0128	0.0263	0.0038	0.0286	0.0286
Canoe	0.0362	0.6613	**0.0000**	0.0968	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Fountain02	0.2808	1.0000	0.0443	0.4384	0.1823	**0.0000**	0.3695	0.4877
Fall	0.3629	0.4253	0.0033	0.2677	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Average	*0.1944*	*0.7198*	***0.0213***	*0.1633*	*0.0614*	*0.0330*	*0.1384*	*0.1481*

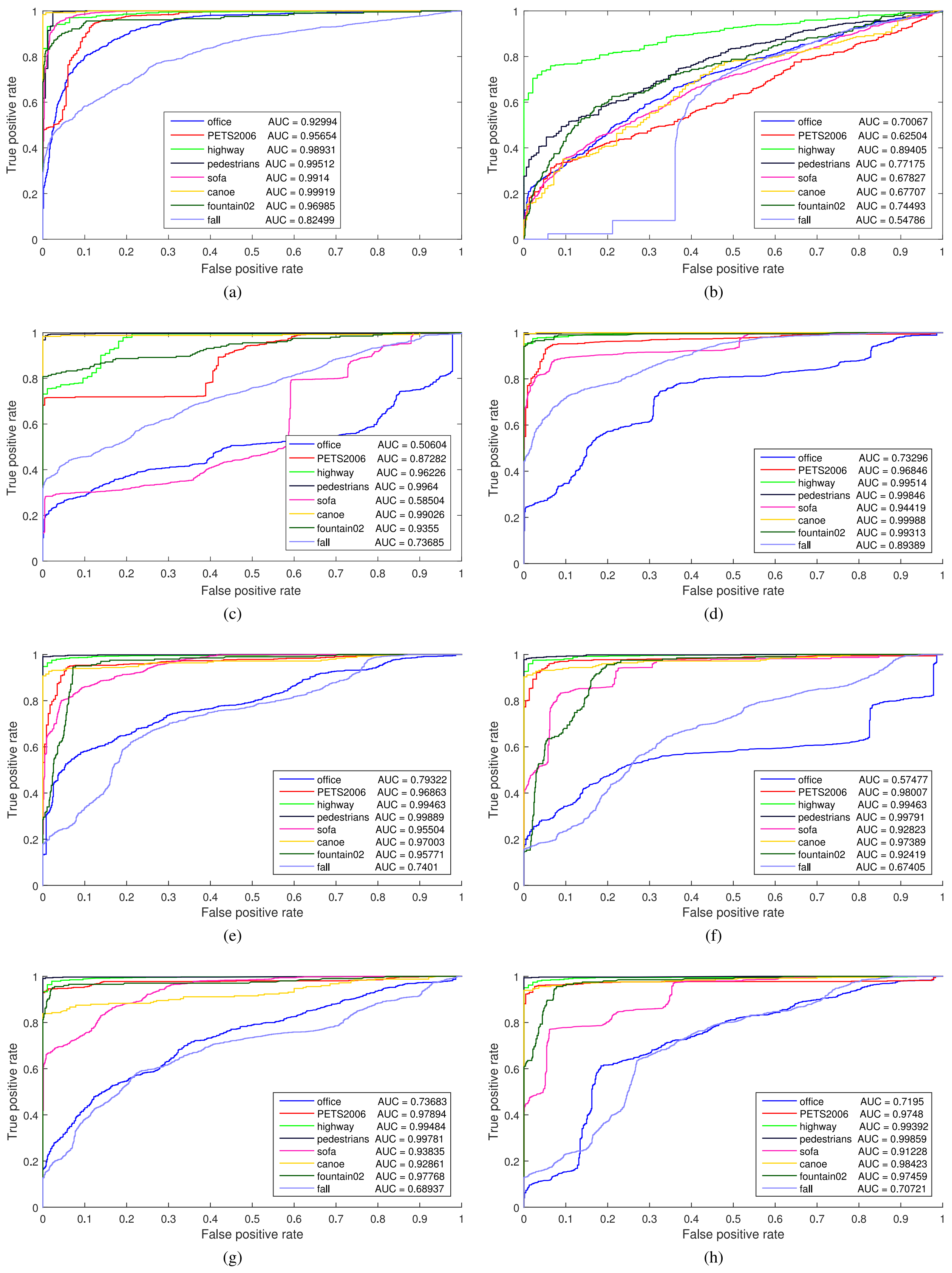

Fig. 7. ROC curves corresponding to the eight analyzed benchmark videos for the different tested methods. First row show the proposed system (a) and the traditional detector (b). FrameDiff (c) and MFBM (d) are in the second row. Third row exhibits the ROC curves for the GrimsonGMM (e) and MaddalenaSOBS (f) methods. Finally the fourth and last row presents the WrenGA (g) and ZivkovicGMM (h). Their corresponding Areas Under Curve (AUC, higher is better) are shown in the legends inside the plots.

Table 9
Probability of bad classification (PBC) of the competing methods over the tested sequences (lower is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	**20.04**	44.5	43.11	43.2725	38.4719	43.1373	43.0696	38.0663
PETS2006	34.71	47.48	20.05	20.2447	**15.6841**	16.4627	43.9377	39.7108
Highway	6.66	35.55	7.12	12.2864	**2.1969**	4.8820	2.9292	3.2547
Pedestrians	4.401	46.79	**1.099**	6.1404	3.2581	2.5063	6.2657	4.6366
Sofa	**9.28**	40.28	38.05	22.4100	14.7621	34.9489	16.3628	16.4073
Canoe	**3.502**	41.65	31.49	6.1856	36.0825	36.0825	36.0825	36.0825
Fountain02	22.81	50.12	68.23	8.9178	8.6172	79.6593	**7.5150**	9.9198
Fall	38.35	50.54	60.54	**17.4116**	60.5070	60.5070	60.5070	60.5070
Average	*17.8062*	*44.6138*	*33.7111*	***17.1086***	*22.4475*	*34.7732*	*27.0837*	*26.0731*

Table 10
Precision of the competing methods over the tested sequences (higher is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	0.8905	**0.9511**	0.5697	0.5677	0.5974	0.5683	0.5686	0.6030
PETS2006	0.9937	0.9848	0.8559	0.9959	0.9906	0.9981	**1.0000**	**1.0000**
Highway	0.9981	**1.0000**	0.930	**1.0000**	0.9973	0.9498	0.9982	0.9982
Pedestrians	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Sofa	**0.9815**	0.7823	0.6197	0.7293	0.8133	0.6290	0.7964	0.7959
Canoe	**1.0000**	0.866	0.6851	**1.0000**	0.6392	0.6392	0.6392	0.6392
Fountain02	0.9799	0.0000	0.3124	**1.0000**	0.7721	0.2034	**1.0000**	**1.0000**
Fall	0.7109	0.4906	0.3943	**0.8088**	0.3949	0.3949	0.3949	0.3949
Average	***0.9443***	*0.7593*	*0.6709*	*0.8877*	*0.7756*	*0.6728*	*0.7997*	*0.8039*

Table 11
F-measure of the competing methods over the tested sequences (higher is better). Best results of each sequence are highlighted in **bold**.

Video	SOM	Ilu	FrameDiff	MFBM	GrimsonGMM	MaddalenaSOBS	WrenGA	ZivkovicGMM
Office	**0.8721**	0.3425	0.7233	0.7217	0.7436	0.7228	0.7234	0.7409
PETS2006	0.6395	0.1776	0.8778	0.8420	**0.8824**	0.8750	0.5775	0.6331
Highway	0.9631	0.6192	0.9630	0.9287	**0.9880**	0.9742	0.9839	0.9821
Pedestrians	0.9764	0.2153	**0.9944**	0.9595	0.9789	0.9839	0.9587	0.9697
Sofa	**0.9471**	0.5722	0.7648	0.8389	0.8863	0.7711	0.8753	0.8750
Canoe	**0.9815**	0.487	0.8131	0.9492	0.7799	0.7799	0.7799	0.7799
Fountain02	**0.8295**	0.0000	0.4709	0.7192	0.7943	0.3381	0.7734	0.6775
Fall	0.672	0.5293	0.5651	**0.7686**	0.5662	0.5662	0.5662	0.5662
Average	***0.8601***	*0.3679*	*0.7716*	*0.8410*	*0.8274*	*0.7514*	*0.7798*	*0.7781*

resent the output of the pixel level algorithms running on a standard PC. As seen, our method outperforms the methods running on the Arduino board, while it is still competitive with respect to the PC based ones. This confirms the previously reported quantitative results.

6. Conclusions

The SOM algorithm has been successfully implemented in a microcontroller DUE board. The SOM has been adapted to overcome the limitations imposed by the limited resources of memory and computing speed of the hardware device. The correct implementation of the algorithm has been verified, and it has been found that as the precision is increased to avoid rounding effects, the microcontroller needs more memory size. Furthermore, a detailed study of the differences of using floating point or fixed precision representations has been carried out, concluding that better results can be obtained with an 32-bit precision fixed point representation leading to computation times up 10 times faster in the largest neural architecture (more neurons) than using the standard floating point representation. Also, the change in the data type representation paradigm allows using SOM architectures with more neurons and processing images with more resolution, obtaining a more precise block based motion detection in the proposed system.

The implemented SOM algorithm has been employed as a motion detector obtaining a cheap and versatile system with which it is possible to carry out efficient video surveillance. The whole learning process has been implemented in the chip, whereby the decision-making procedure of the detector is adapted in real time to the observed changes in the scene. This way decision errors produced by the evolution in the captured environment are significantly reduced.

The efficiency of the proposed system is significantly higher than that of the traditional motion detection method. It has a higher success rate with less false positives. False positives can be reduced almost to 0 by tuning the threshold, due to the favorable dependency between the true and false positive rates reported by the ROC curves.

As an overall conclusion, we have shown the suitability of SOM algorithm for its application in a motion detection using an Arduino DUE microcontroller. Therefore the present study demonstrates the potential of the proposed methodology for its application to inexpensive systems in real scenarios. The detection performance can be further enhanced by employing more powerful

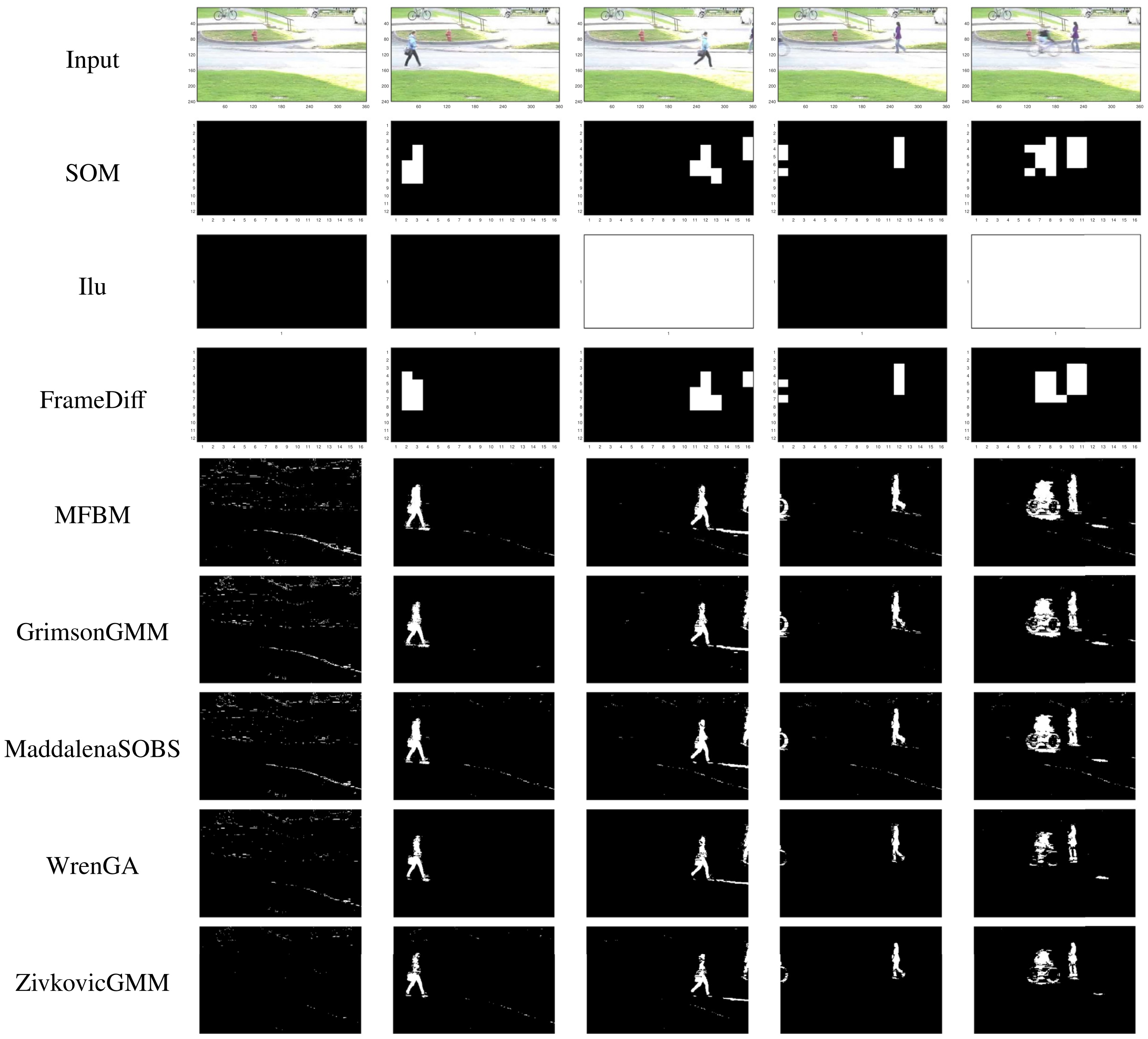

Fig. 8. Motion detection examples for the Pedestrians video. First row: raw RGB video captured by the video camera. Remaining rows: detection decision by the proposed system (SOM), the traditional detector (Ilu), FrameDiff, MFBM, GrimsonGMM, MaddalenaSOBS, WrenGA and ZivkovicGMM, respectively.

computing resources and ad hoc devices than the microcontroller board considered here.

Acknowledgments

This work is partially supported by the Ministry of Economy and Competitiveness of Spain under grants TIN2010-16556 and 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 P10-TIC-5770, P08-TIC-04026, 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. 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.

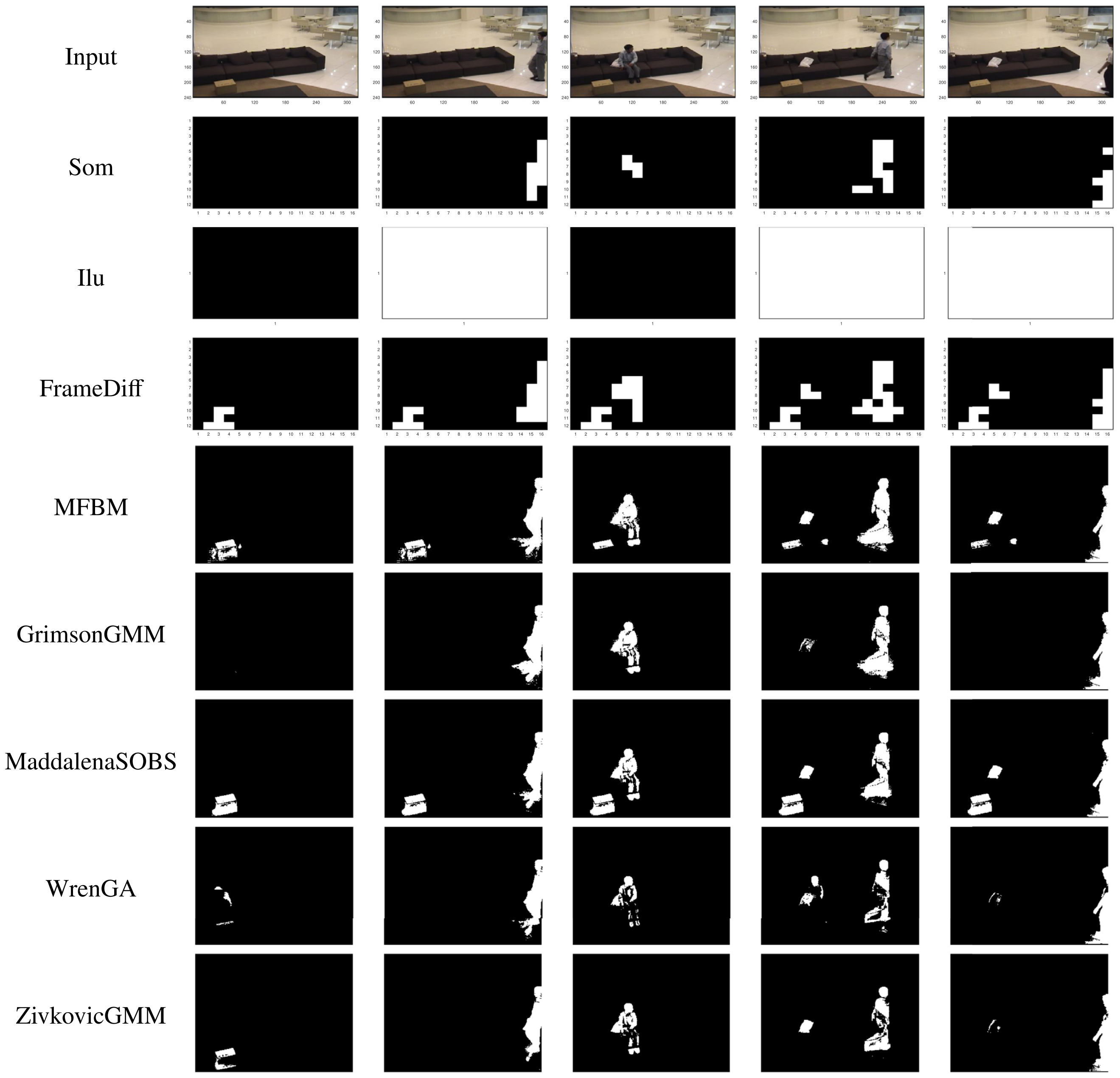

Fig. 9. Motion detection examples for the Sofa video. First row: raw RGB video captured by the video camera. Remaining rows: detection decision by the proposed system (SOM), the traditional detector (Ilu), FrameDiff, MFBM, GrimsonGMM, MaddalenaSOBS, WrenGA and ZivkovicGMM, respectively.

References

Adnan, L., Yussoff, Y., Johar, H., & Baki, S. (2015). Energy-saving street lighting system based on the waspmote mote. *Jurnal Teknologi, 76*(4), 55–58.

Aleksendrić, D., Jakovljević, I., & Irović, V. (2012). Intelligent control of braking process. *Expert Systems with Applications, 39*(14).

Atmel, DataSheet Atmel SAM3X8E ARM Cortex-M3 CPU. http://www.atmel.com/Images/doc11057.pdf.

Beaton, D., Valova, I., & MacLean, D. (2010). CQoCO: A measure for comparative quality of coverage and organization for self-organizing maps. *Neurocomputing, 73*(10–12), 2147–2159.

Bermejo, S., & Cabestany, J. (2002). The effect of finite sample size on on-line k-means. *Neurocomputing, 48*(1), 511–539.

Bhandarkar, S., Koh, J., & Suk, M. (1997). Multiscale image segmentation using a hierarchical self-organizing map. *Neurocomputing, 14*(3), 241–272.

Bouwmans, T. (2014a). *Background modeling and foreground detection for video surveillance* (pp. 1–54). Chapman and Hall/CRC.

Bouwmans, T. (2014b). Traditional and recent approaches in background modeling for foreground detection: An overview. *Computer Science Review, 11–12*, 31–66.

Casanova, C., Franco, A., Lumini, A., & Maio, D. (2013). SmartVisionApp: A framework for computer vision applications on mobile devices. *Expert Systems with Applications, 40*(15), 5884–5894.

Cela, A., Yebes, J. J., Arroyo, R., Bergasa, L. M., Barea, R., & López, E. (2013). Complete low-cost implementation of a teleoperated control system for a humanoid robot. *Sensors, 13*(2), 1385–1401.

Chang, C.-H., Pengfei, X., Xiao, R., & Srikanthan, T. (2005). New adaptive color quantization method based on self-organizing maps. *IEEE Transactions on Neural Networks, 16*(1), 237–249.

Dekker, A. (1994). Kohonen neural networks for optimal color quantization. *Network, 5*, 351–367.

Dlugosz, R., Talaska, T., Pedrycz, W., & Wojtyna, R. (2010). Realization of the conscience mechanism in CMOS implementation of winner-takes-all self-organizing neural networks. *IEEE Transactions on Neural Networks, 21*(6), 961–971.

Dobrzynski, M., Pericet-Camara, R., & Floreano, D. (2012). Vision tape-a flexible compound vision sensor for motion detection and proximity estimation. *IEEE Sensors Journal, 12*(5), 1131–1139.

Dong, G., & Xie, M. (2005). Color clustering and learning for image segmentation based on neural networks. *IEEE Transactions on Neural Networks, 16*(4), 925–936.

Fung, V., Bosch, J., Roberts, S., & Kleissl, J. (2014). Cloud shadow speed sensor. *Atmospheric Measurement Techniques, 7*(6), 1693–1700.
García, F., García, J., Ponz, A., de la Escalera, A., & Armingol, J. M. (2014). Context aided pedestrian detection for danger estimation based on laser scanner and computer vision. *Expert Systems with Applications, 41*(15), 6646–6661.
Gómez, M. J., García, F., Martín, D., de la Escalera, A., & Armingol, J. M. (2015). Intelligent surveillance of indoor environments based on computer vision and 3D point cloud fusion. *Expert Systems with Applications, 42*(21), 8156–8171.
Hsu, A., & Halgamuge, S. (2003). Enhancement of topology preservation and hierarchical dynamic self-organising maps for data visualisation. *International Journal of Approximate Reasoning, 32*(2–3), 259–279.
Kaski, S., Kangas, J., & Kohonen, T. (1998). Bibliography of self-organizing map (SOM) papers: 1981–1997. *Neural Computing Surveys, 1*, 102–350.
Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*(1), 59–69.
Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks, 37*, 52–65.
Kopetz, H. (1997). *Real-time systems: Design principles for distributed embedded applications* (1st). Norwell, MA, USA: Kluwer Academic Publishers.
Kornuta, J. A., Nipper, M. E., & Brandon Dixon, J. (2012). Low-cost microcontroller platform for studying lymphatic biomechanics in vitro. *Journal of Biomechanics, 46*(1), 183–186.
Lacerda, E. B., & Mello, C. A. (2013). Segmentation of connected handwritten digits using self-organizing maps. *Expert Systems with Applications, 40*(15), 5867–5877.
Lian, K.-Y., Hsiao, S.-J., & Sung, W.-T. (2013). Intelligent multi-sensor control system based on innovative technology integration via ZigBee and Wi-Fi networks. *Journal of Network and Computer Applications, 36*(2), 756–767.
López-Rubio, E., Luque-Baena, R. M., & Domínguez, E. (2011). Foreground detection in video sequences with probabilistic self-organizing maps. *International Journal of Neural Systems, 21*(3), 225–246.
López-Rubio, F. J., & López-Rubio, E. (2015). Features for stochastic approximation based foreground detection. *Computer Vision and Image Understanding, 133*, 30–50.
Maddalena, L., & Petrosino, A. (2008a). A self-organizing approach to background subtraction for visual surveillance applications. *IEEE Transactions on Image Processing, 17*(7), 1168–1177.
Maddalena, L., & Petrosino, A. (2008b). A self-organizing approach to background subtraction for visual surveillance applications. *IEEE Transactions on Image Processing, 17*(7), 1168–1177.
Mahmoud, S., Lotfi, A., & Langensiepen, C. (2013). Behavioural pattern identification and prediction in intelligent environments. *Applied Soft Computing, 13*(4), 1813–1822.
Mamdoohi, G., Fauzi Abas, A., Samsudin, K., Ibrahim, N. H., Hidayat, A., & Mahdi, M. A. (2012). Implementation of genetic algorithm in an embedded microcontroller-based polarization control system. *Engineering Application of Artificial Intelligence, 25*(4), 869–873.
Marwedel, P. (2006). *Embedded system design*. Secaucus, NJ, USA: Springer-Verlag New York, Inc..
Naghiyev, E., Gillott, M., & Wilson, R. (2014). Three unobtrusive domestic occupancy measurement technologies under qualitative review. *Energy and Buildings, 69*, 507–514.
Oja, M., Kaski, S., & Kohonen, T. (2003). Bibliography of self-organizing map (SOM) papers: 1998–2001 addendum. *Neural Computing Surveys, 3*(1), 1–156.
Ortega-Zamorano, F., Jerez, J., Juarez, G., Perez, J., & Franco, L. (2014). High precision FPGA implementation of neural network activation functions. In *2014 IEEE symposium on intelligent embedded systems (IES)* (pp. 55–60).
Ortega-Zamorano, F., Jerez, J., Urda Munoz, D., Luque-Baena, R., & Franco, L. (2015). Efficient implementation of the backpropagation algorithm in FPGAs and microcontrollers. *IEEE Transactions on Neural Networks and Learning Systems, PP*(99). 1–1
Ortega-Zamorano, F., Jerez, J. M., Subirats, J. L., Molina, I., & Franco, L. (2014). Smart sensor/actuator node reprogramming in changing environments using a neural network model. *Engineering Applications of Artificial Intelligence, 30*(0), 179–188.
Oxer, J., & Blemings, H. (2009). *Practical arduino: Cool projects for open source hardware*. Berkely, CA, USA: Apress.
Palomo, E. J., & Domínguez, E. (2014). Hierarchical color quantization based on self--organization. *Journal of Mathematical Imaging and Vision, 49*(1), 1–19.
Papadimitriou, K., Dollas, A., & Sotiropoulos, S. (2006). Low-cost real-time 2-D motion detection based on reconfigurable computing. *IEEE Transactions on Instrumentation and Measurement, 55*(6), 2234–2243.
Papamarkos, N. (1999). Color reduction using local features and a SOFM neural network. *Journal of Imaging Systems and Technology, 10*(5), 404–409.
Park, H., Park, J., Kim, H., Jun, J., Son, S. H., & Park, T. (2015). ReLiSCE: Utilizing resource-limited sensors for office activity context extraction. *IEEE Transactions on Systems, Man, and Cybernetics, 45*(8), 1151–1164.
Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). Real-time computer vision with OpenCV. *Communications of the ACM, 55*(6), 61–69.
Sengupta, S., Das, S., Nasir, M., & Panigrahi, B. K. (2013). Multi-objective node deployment in WSNs: In search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity.. *Engineering Applications of Artificial Intelligence, 26*(1), 405–416.
Stauffer, C., & Grimson, W. (1999). Adaptive background mixture models for real–time tracking. In *Proceedings of theIEEE international conference on computer vision and pattern recognition* (pp. 246–252).
Wang, J., Xu, W., & Gong, Y. (2010). Real-time driving danger-level prediction. *Engineering Applications of Artificial Intelligence, 23*(8), 1247–1254.
Wang, Y., Jodoin, P.-M., Porikli, F., Konrad, J., Benezeth, Y., & Ishwar, P. (2014). CDnet 2014: An expanded change detection benchmark dataset. In *Computer vision and pattern recognition workshops (cvprw), 2014 ieee conference on* (pp. 393–400). doi:10.1109/CVPRW.2014.126.
Wren, C., Azarbayejani, A., Darrell, T., & Pentl, A. (1997). Pfinder: Real-time tracking of the human body. *IEEE Trans. on Pattern Analysis and Machine Intelligence, 19*(7), 780–785.
Xiao, Y., Leung, C.-S., Lam, P.-M., & Ho, T.-Y. (2012). Self-organizing map-based color palette for high-dynamic range texture compression. *Neural Computing and Applications, 21*(4), 639–647.
Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks, 52*(12), 2292–2330.
Yin, H. (2008). The self-organizing maps: Background, theories, extensions and applications. *Studies in Computational Intelligence, 115*, 715–762.
Zivkovic, Z. (2004). Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the pattern recognition, 17th international conference on (icpr'04) volume 2 - volume 02*. In *ICPR '04* (pp. 28–31). Washington, DC, USA: IEEE Computer Society.
Zivkovic, Z., & van der Heijden, F. (2006). Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters, 27*(7), 773–780.