FPGA-based Anomalous trajectory detection using SOFM

Kofi Appiah[1] and Andrew Hunter[1] and Tino Kluge[2] and Philip Aiken[3] and Patrick Dickinson[1]

[1] Dept. of Computing & Informatics, University of Lincoln, UK
[2] Statistical Laboratory, University of Cambridge, UK
[3] SecuraCorp, Kirkland, USA

Abstract. A system for automatically classifying the trajectory of a moving object in a scene as usual or suspicious is presented. The system uses an unsupervised neural network (Self Organising Feature Map) fully implemented on a reconfigurable hardware architecture (Field Programmable Gate Array) to cluster trajectories acquired over a period, in order to detect novel ones. First order motion information, including first order moving average smoothing, is generated from the 2D image coordinates (trajectories). The classification is dynamic and achieved in real-time. The dynamic classifier is achieved using a SOFM and a probabilistic model. Experimental results show less than 15% classification error, showing the robustness of our approach over others in literature and the speed-up over the use of conventional microprocessor as compared to the use of an off-the-shelf FPGA prototyping board.

1 Introduction

An intelligent video surveillance system should be able to keep track of objects in a camera view *(identity tracking)*, determine where the objects are in the camera view *(location tracking)* and what the people, vehicles or objects are doing in the scene *(activity tracking)*[1]. Increasing safety and security concerns have resulted in the development of complex video surveillance and traffic monitoring systems in both research and industrial communities [2]. An intelligent surveillance system should have the capability to process video streams and characterize the actions taking place, to distinguish between normal and abnormal actions, and to draw the attention of a human operator when an action poses a threat.

CCTV systems have a human operator monitoring a number of cameras at the same time. Typically, the concentration level of the operator reduces after 15 minutes monitoring a non-active camera scene. Increasing demands for use of surveillance cameras in crime prevention calls for the development of automated techniques capable of detecting actions that poses a threat in a camera scene and subsequently signal an alert to the human operator for verification. Such automation will augment rather than replace the human operator.

Evolution of computer vision algorithms as well as Ambient intelligence (AmI) [20], oriented towards ubiquitous computing and smart environments that

react in an adaptive and active way to the presence and actions of objects implemented on embedded systems, has become an interesting area of research over the past decade. Such systems allow the implementation of early vision processes similar to the first neural layer in the retina, for pre-filtering conspicuous information [21]. Field Programmable Gate Array (FPGA), a technology which has recently been made available to researchers, is used as an alternative platform for speedup in computer vision and digital image processing [19]. The potential uses of FPGAs in areas like medical image processing, computational fluid dynamics, target recognition, embedded vision systems, gesture recognition and automotive infotainment have been demonstrated in [19] [20] [21] [22] [23].

We use a reconfigurable architecture, FPGA, in conjunction with an image sensor to process trajectory information of moving objects and only send alerts to a central monitoring station. The system architecture is based upon the commercially available *Algorithm Based Object Recognition and Tracking (ABORAT)* system designed for wireless IP cameras for highly sophisticated security surveillance systems[24]. Whereas ABORAT uses an Intel Bulverde (PXA270) in this paper an FPGA is used for the pre-filtering and classification of trajectories as either normal or abnormal.

2 Related Work

Abnormal activity detection has been divided into two categories: parametric and non-parametric by Zhou et. al[13]. The parametric approach models normal and abnormal activities using visual features like position, speed and appearance, while the non-parametric learns the normal and abnormal patterns from the statistical properties of the observed data. In this paper we further divide the non-parametric into two sub-groups; the on-line and the batch approach. The batch approach trains and detects normal and abnormal activities using complete trajectories. The on-line approach may or may not train the system using complete trajectories, yet it is able to detect normal/abnormal activities using incomplete trajectories; hence the ability to detect abnormalities as they happen.

Generally, the trajectory data of tracked objects are recorded as a set of (x, y) locations of the tracked object's centre of mass from frame to frame. In [10], they used flow vectors $f = \{x, y, \delta x, \delta y\}$ rather than sequence of positions to describe an object's movement. Thus if an object i appears in n frames it can be represented by a set Q_i of n flow vectors all lying within a unit hypercube in 4D phase space: $Q_i = \{f_1, f_2, \ldots, f_{n-1}, f_n\}$. Owens et. al in [8], used a hierarchical neural network as a novelty detector. Normal trajectories are used during training, and experiment conducted shows a high detection rate. Humphreys et. al [7] has extensively use cost functions based on SOFM to detect, track and classify object trajectories. The paper also demonstrates improved performance, by breaking down the SOFM into three parts.

Grimson et. al [15] used the (x, y) location, speed/direction (dx, dy) and size to develop a codebook using an on-line Vector Quantization(VQ). A co-

occurrence statistic is accumulated over the codebook and a hierarchical classification performed to identify normal and abnormal activities. In [14] a Dynamic Oriented Graph (DOG) is used to structure common patterns of objects activities. The entrance, path and departure nodes are used to construct the graph. The spatial motion information, size and colour of the objects are used to classify their activity. During testing, known or normal object patterns should match an existing node, else the activity is rejected and classified as unusual or abnormal.

Jiang et. al [5] uses an information-based trajectory dissimilarity measure making use of Bayesian information criterion (BIC) to determine the number of clusters in the agglomerative hierarchical clustering algorithm. Each trajectory or feature sequence is modelled by a hidden Markov model (HMM) and the number of clusters in the BIC decreases as similar trajectories are merged. In [13] an unsupervised spectral clustering, represented by the mean and variance of their data points is used for clustering trajectories. An unusual approach is taken by Dickinson et. al [6], who use HMMs to differentiate between normal and unusual behaviour in domestic scenes. The hidden states correspond to learned inactive states, such as sitting on a particular chair, and transitions correspond to movement between them. Trajectories are therefore represented as movements between familiar static events. The learned model of normal behaviour is used in conjunction with a threshold model to classify test sequences.

A multi-sample-based similarity measure using a dynamic hierarchical clustering method has been present in [2]. Trajectory data acquired over a period of time are represented by a 5-state HMM with Gaussian emission probability and used as the training data. HMMs are learnt from clusters with large number of samples and are used for detecting abnormal trajectories. Han et. al[25] used an unsupervised fuzzy self-organising map trained with normal activities. Trajectory features $(x, y, \sqrt{\delta x^2 + \delta y^2}, \tan^{-1}(\delta y/\delta x))$ are translated into a fixed length vector of size $4N$. An object with trajectory length less than N, is padded with it last centre position (x, y), zero speed and zero direction. The $4N$ vector is used to train the FSOM, which is then used for the abnormal activity detection in real-time/on-line mode.

A Spatial Occupancy Map (SpOM) built from object trajectories has been used in [4] for detection of unusual trajectories in a camera scene. Object trajectories are modelled as motion time series in [12] to train a coefficient feature space, which is subsequently used for trajectory classification. Principal Component Analysis (PCA) trained with sub trajectories, used in conjunction with HMM has been presented in [3] for classifying motion trajectories. In [11] trajectories are collected and processed on-line as a list of vectors representing the spatial position of the object. Rather than the Euclidean distance, a new distance measure has been introduced to check if a trajectory fits a given cluster.

Owens et. al in [9] used a self-organising feature map to learn and characterize normal trajectories. The 4D flow vector $(x, y, \delta x, \delta y)$ used in [15] has been extended into 8D $(x, y, s(x), s(y), s(\delta x), s(\delta y), s(\delta^2 x), s(\delta^2 y))$ to include a second order motion information. The on-line system presented in [9] is capable of detecting abnormalities in both instantaneous and whole trajectory motion. In

[16] a feature vector $\dot{\gamma}_t$ which encapsulates the local curvature of the trajectory as well as the local velocity magnitude is used to represent the tracked object's position. The feature vectors are modelled with a HMM and the similarity between trajectories expressed with a quantization-based HMM. Piciarelli et. al in [17] used a support vector machine (SVM) for the classification and clustering of 2D trajectory data. Mixtures of Gaussians (MoGs) have been used in [18] to group 4D motion histogram data into coherent trajectories and used to identify events after training.

In this paper we present an on-line base event detection system using trajectory data and implemented on Field Programmable Gate Array (FPGA) for real-time purposes. The system is able to detect abnormal trajectory data point-by-point using SOFM in conjunction with a Gaussian distribution. The paper is structured as follows. Section 3 introduces our approach with the theoretical background to the parameter selection for SOFM neural network. This is followed by details of our FPGA implementation in section 4. Experimental details are given in section 5 and we conclude with some future pointers to this work in section 6.

3 Our Approach

To efficiently implement a trajectory discriminator in hardware using Self Organising Feature Map (SOFM) and Gaussian distribution, we have conducted two basic analyses. First, we analyse the minimal dimension that can be used to represent the point-to-point trajectory data (x_t, y_t) without losing any behavioural information. Intuitively, the minimum dimension is 2D, yet in [16] the (x_t, y_t) coordinate information has been reduced to a single value $\dot{\gamma}_t$ encoding the local curvature and velocity information. The penalty for the model is the high dimensional vector used in the HMM. Secondly, we analyse the most efficient way to represent the trajectory data in the SOFM. By reducing the dimension of the trajectory data we are able to implement the SOFM on FPGA using the internal/embedded Block RAM.

3.1 Curse of Dimensionality

In general, the more data put into the state vector the better one would expect to be able to distinguish between usual and unusual behaviour. Obvious variables are the (x_t, y_t) coordinates, their time derivatives representing speed as well as their second time derivatives representing acceleration, which would leave us with a 6D vector $(x, y, \delta x, \delta y, \delta^2 x, \delta^2 y)$. Other papers [16, 18, 11, 9] have used even higher dimensions. However, as the dimensionality of the input increase the number of nodes required in the SOFM grows very rapidly. The number of nodes required to populate the input state space to a given density increase exponentially with the dimension, and although the data may lie on a lower-dimensional manifold, this does mean that performance can drop if too many input features are used. Therefore, dimensionality reduction should be performed

whenever the setup allows, e.g. if the speed of an object is generally independent of its position then the state variables (x, y) and $(\delta x, \delta y)$ can be represented in different networks. However, this is not the case in general, for example if the scene contains a highly obstructed footpath and a clear road, then object speed will be higher on the road than the footpath. In conclusion, we will use a 4D vector with the objects position (centroid) (x, y) and speed $(\delta x, \delta y)$ to model nodes in the SOFM.

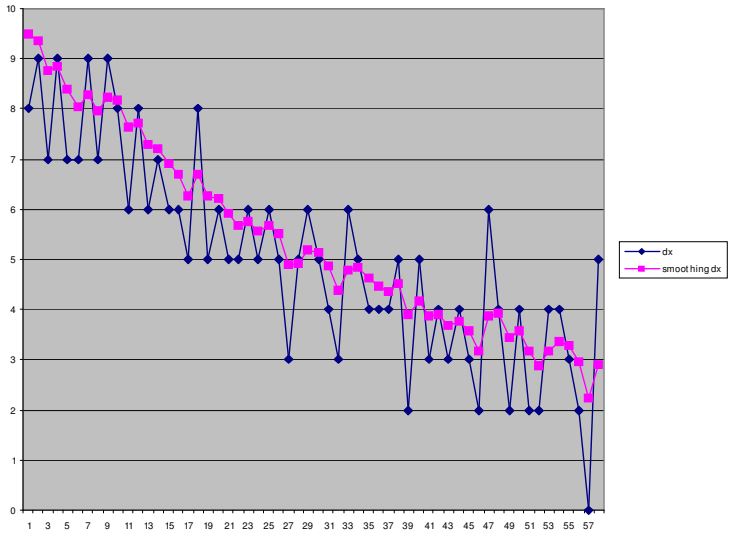

Fig. 1. A graph showing the effect of smoothing the speed component (δx).

3.2 Importance of smoothing the speed data

The trajectory coordinates obtained represent the centroid of the object tracked over the input image frames. However, due to camera jitter and slight movement, the centroid includes some noise. Let the observed trajectory be given by $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ then we say $x_i = x_i^* + \epsilon_i$ where x_i^* are the true x coordinates and ϵ_i independent identically distributed random variables representing the error. The same applies to the y coordinates.

We also assume the spatial error ϵ_i is small compared to the size of the moving object. When calculating changes in position between two frames, however, relative errors can be very large because $\delta x_i = x_i - x_{i-1} = \delta x_i^* + \epsilon_i - \epsilon_{i-1}$ and $\delta x_i^* = x_i^* - x_{i-1}^*$ is likely to be small, and the standard deviation of the error is $\sqrt{2}$ times that of the spatial error. Figure 1 shows a plot of δx for a sample trajectory with 58 points and the corresponding smoothed versions $s\delta x$. We use an

exponential moving average as it is faster and simpler to implement on FPGA:

$$\begin{aligned} s\delta x_1 &= \delta x_1, \\ s\delta x_{i+1} &= \alpha\delta x_{i+1} + (1-\alpha)s\delta x_i, \end{aligned}$$

or equivalently

$$s\delta x_k = \alpha \sum_{i=0}^{k-1} (1-\alpha)^i \delta x_{k-i}. \tag{1}$$

Note, [16] uses a similar smoothing applied to the coordinates itself:

$$sx_k = c^{-1} \sum_{i=0}^{k-1} \mathrm{e}^{-\left(\frac{i}{h}\right)^2} x_{k-i}, \tag{2}$$

where h is chosen appropriately and c is a normalising constant. This is similar to the exponential moving average as we can rewrite it as

$$s\delta x_k = \alpha \sum_{i=0}^{k-1} \mathrm{e}^{\ln(1-\alpha)i} \delta x_{k-i}. \tag{3}$$

Small values of α indicate a long history of coordinates is taken into account whereas values of α close to 1 are used to prioritise the current coordinates. There is a trade off between reducing errors and taking averages over too long a period (say more than one second) which would give us outdated information. The error in the moving average due to the noise term can be described as follows. Assuming an infinite series for simplicity we have

$$\begin{aligned} \mathrm{Var}[s\delta x_k] &= \mathrm{Var}\left[\alpha \sum_{i=0}^{\infty} (1-\alpha)^i \epsilon_{k-i}\right], \\ &= \alpha^2 \sum_{i=0}^{\infty} (1-\alpha)^{2i} \mathrm{Var}[\epsilon_{k-i}], \\ &= \frac{\alpha^2}{1-(1-\alpha)^2} \mathrm{Var}[\epsilon], \end{aligned}$$

and hence the standard deviation of the error of the moving average is $\sqrt{\frac{\alpha}{2-\alpha}}$ times that of the error in δx. For values $\alpha = 1/4$ down to $\alpha = 1/10$ we get a reasonable reduction of noise to the order of about 0.38 and 0.23, respectively, and assuming we have 25 frames per second the contribution of frames older than a second to the moving average is negligible.

3.3 Trajectory modelling with SOFM

Similar to [9], our systems monitors trajectories as they are generated as opposed to other systems [2, 13] which need the entire trajectory to make a decision. Hence the trajectory encoding used here converts both full and sub trajectories into a fixed length feature vector $F = (x, y, s\delta x, s\delta y)$, where $s\delta x$ and $s\delta y$ are the moving averages for the change in x and y respectively. As the feature vector generated for each individual point is of fixed length, a SOFM-based has been used for classification.

The self-organising feature map (SOFM) is a neural network model based on Kohonen's discovery that important topological information can be obtained through an unsupervised learning process[26]. It has an input layer, with one weight each for all elements in the feature vector F. It has two phases: the training and the test phases. During the training phase, data is presented to the network and the winning node (typically the Euclidean distance measure) is updated to reflect the input data. Similarly, during the test phase an Euclidean distance measure is used to identify the winning node *(winner)* and a decision made on how close the input is to the network node.

We have designed our SOFM with 100 network nodes, each with four weights representing the 4-input feature vector $(x, y, \delta x, \delta y)$. During the training, we maintain four extra parameters for each node in the network: the total number of training samples that get associated with each node T_i, the maximum distance between the node and all associated inputs, M_i, the mean μ_i and variance σ_i^2 of the distances. A Gaussian distribution of all distances associated with every node is also maintained.

The training data is made up of both normal and abnormal trajectories, yet our implementation is able to distinguish between normal and abnormal trajectories after training. Trajectory data (x, y) is collected over a period of time from a stationary camera and converted into a 4D feature vector F for training the SOFM. During training, the 100 network nodes are randomly initialized, then for every input vector (feature vector), the Manhattan distance between the input vector and every network node is computed to estimate the winner. For a winner w_t and input vector x, all the weights i of the winning node are updated as follows $w_{i,t+1} = w_{i,t} + \beta(x - w_{i,t})$ to reflect the input data. If the Manhattan distance $m_{w,x}$ between w_t and x is the maximum for node $w_t, M_w = m_{w,x}$. Similarly, the total distance for the winner T_w is increased by $m_{w,x}$.

The training of the SOFM is repeated for a number of epochs with the same input data. The Gaussian distribution for each node is generated for a random iteration $t \leq (epoch - 1)$ during training. The network is ready for use after the training phase. During the test phase, point-to-point trajectory data (x, y) is converted into a 4D vector and used as input to the SOFM. Again, the winning node is identified as the node with the minimum Manhattan distance to the input vector. In the test phase the network isn't subjected to any further modification, but rather is used to make a decision on the input vector or trajectory.

An input trajectory data for tracked objects is identified as abnormal if any of the following conditions is true:

1. If the Manhattan distance $m_{w,x}$ between the input vector x and the winner w is greater than the maximum allowable distance for the winner M_w.
2. If T_w (the total number of input vectors associated with the winner during training) is less than a gobal threshold Th set as $0.01\% * total\ train\ points$.
3. If the Manhattan distance $m_{w,x}$ is outside 2.5 standard deviation of the Gaussian distribution for the winner.

The penalty for option 1 is the highest, followed by options 2 and 3 respectively. An input node whose Manhattan distance is greater than M_w is abnormal on the assumption that such a point is new to the SOFM. Since the system is trained with both normal and abnormal trajectories, it is possible for a node in the network to represent only abnormal trajectory points. Since unusual trajectories are rare, an assumption that no more than $Th = 0.01\%$ of the entire trajectory points are abnormal is made. Hence, any network node with less than the global threshold value Th of points, is labelled as an abnormal network node n_{ab}. Thus any input vector whose winner is n_ab is also considered abnormal.

It is also possible to associate an abnormal point to a normal network node n_{nor} during training. If this happens, we expect the Manhattan distance between the abnormal point x_{ab} and the network node n to be much greater than all other points associated with n_{nor}. The Gaussian distribution maintained for n_{nor} is then used to identify such abnormal trajectory points. Figure 2 shows two images with normal and abnormal trajectory points.

(a) Normal trajectory (b) Abnormal trajectory

Fig. 2. Images showing (a) normal and (b) abnormal trajectories. In (b), abnormal points are labelled black.

4 Hardware Implementation

The training and testing of the trajectory classifier has been implemented on an FPGA architecture, making use of the embedded RAM to store the network node values. Figure 3 is a high-level block diagram of the FPGA classifier. The

bold lines show the part of the system activated when in the test phase. During the training phase, the trajectory data is read from the external RAM and converted into 4D feature vector for the training the SOFM. In the test phase, point-to-point trajectory data is sent to the FPGA via the RS232/USB port on the development board. The entire design has been accomplished on RC340 development board packaged with Xilinx Virtex-4 FPGA chip (XC4VLX160) with approximately 152,064 logic cells with embedded Block RAM totalling 5,184 Kbits.

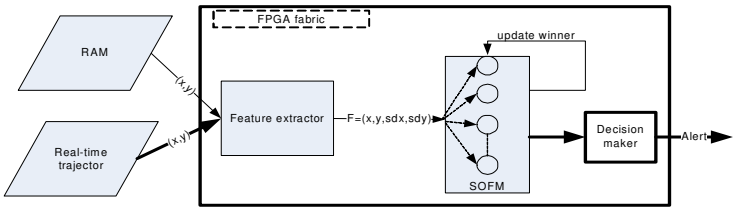

Fig. 3. A block diagram of the FPGA based trajectory classifier.

The design is made up of four hardware blocks: the initialization, feature extraction, winner identification and update (for the training phase). The initialization block is used to initialize all weights for all the network nodes in the 10×10 SOFM neural network. It takes exactly 100 cycles to initialize all the 100 nodes. Note that this is done once before training the SOFM. The feature extraction block converts the trajectory data (x, y) into 4D feature vector. It takes exactly two clock cycles in the test phase and an extra two cycles in the training phase to read the trajectory data from external ZBT RAM. The first cycle is used to convert the data into $(x, y, \delta x, \delta y)$. The second cycle is used to compute the moving averages of the first derivatives $(s\delta x, s\delta y)$.

The winner identification block is use to identify the winning node (winner). The feature vector from the feature extraction block is sent to all the 100 SOFM nodes in parallel. It takes exactly four clock cycles to compute the Manhattan distance from the input feature vector to all the nodes. Note, the feature vector is 4D, hence the four clock cycles. This can further be reduced to a single clock cycle depending on the memory structure used. The node with the minimum Manhattan distance is computed in approximately seven cycles from the 100 distances. Again, the number of cycles here is dependent on the number of network nodes.

The update block takes exactly two cycles to update the winning node and its neighbouring nodes. This is only done in the training phase. After training on 100 iterations the system switches into test the phase and writes the network node values on to an external RAM block for verification. In general the implementation as it stands takes $(17 * epoch * total\ inputs) + 100$ clock cycles to

completely train the system, and 13 clock cycles to classify a trajectory point in the test phase. Using the rule of thumb, the minimum acceptable *epoch* is equal to the number of nodes in the network, 100 in this case. At 25MHz, the system is capable of completely training the SOFM with 65535 trajectory points in approximately $5sec$, excluding external factors like the access time to the external RAM. Similarly, at the same frequency the implementation is capable of classifying over a million trajectory points in a second. Table 1 shows the resource utilization of our FPGA implementation.

Resource		**Total Used**	
Name	**Total**	**Used**	**Per.(%)**
Flip Flops	135,168	3,826	2
4 input LUTs	135,168	25,821	19
bonded IOBs	768	156	20
Occupied Slices	67,584	15744	23
Block RAM	5,184kb	9,800b	-

Table 1. Implementation results for the SOFM classifier, using Virtex-4 *XC4VLX160*, package *FF1148* and speed grade *-10*.

5 Experimental Results

Three different datasets have been used in testing the implementation on a PC with a general purpose processor clocked at 2.8GHz and on an FPGA with Xilinx Virtex-4 clocked at 25MHz. All the images have been obtained using a stationary camera. The input image is sent to an object tracker and the trajectory fed to this system. Two of the image sequences have been acquired on a normal day while the last of the three has been collected on a rainy day. They have all been collected over a period of 3 hours. The datasets are made of 34713 and 21867 trajectory points for the normal day and 12636 trajectory points for the rainy day. Table 2 is a summary of the test conducted on the FPGA and PC with the same input data and epoch.

A test has also been conducted on the number of trajectory points correctly classified with the implementation. For 520 trajectory points collected on a normal day, 421 were correctly classified as normal, 76 correctly classified as abnormal and 23 were incorrectly classified as normal, representing approximately 4.4% error. A similar test conducted on the same scene, on a rainy day with a total of 151 trajectory points gave 97 correctly classified as normal, 32 correctly classified as abnormal, 19 incorrectly classified as normal with 3 classified incorrectly as abnormal. This represents a total of 14.5% error. Even though the error level is high on a rainy day its fairly acceptable on a normal day. The

Day	**Points**	**PC(min.)**	FPGA(min.)	**epoch**
Normal	34713	45	7.2	346
Normal	21867	27	4.5	218
Rainy	12636	10	2	126

Table 2. Timing results for train the SOFM on FPGA and PC

implementation on FPGA with approximately 5 fold speed improvement is a significant advantage over our PC-based implementations.

6 Conclusion

A system for classifying trajectories in real-time have been presented in this paper. The architecture is fully implemented on an FPGA making it possible to break the training time bottlenecks. This is not the first implementation of SOFM on FPGA, but the use of SOFM on FPGA as a trajectory classifier makes our implementation novel. Again, the on-line classifier based on point-to-point makes this architecture more usable for today's embedded security surveillance systems. A possible extension is to incorporate an object tracker on the FPGA architecture.

References

1. Hampapur, A., Brown, L., Connell, J., Ekin, A., Haas, N., Lu, M., Merkl, H., Pankanti, S.: Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. IEEE Signal Processing Magazine, Volume 22, Issue 2, March 2005 Page(s): 38 - 51
2. Jiang, F., Wu, Y., Katsaggelos, K.A.: Abnormal Event Detection From Surveillance Video By Dynamic Hierarchical Clustering. in Proc. IEEE Int'l Conf. on Image Processing (ICIP'07), San Antonio, TX, Sept. 2007
3. Bashir, F., Khokhar, A.,Schonfeld, D.,: Object Trajectory-Based Motion Modeling and Classification using Hidden Markov Models. IEEE Transactions on Image Processing, Vol. 16 (7), July 2007, pp. 1912-1919
4. Jung, C.,Jacques, J.,Soldera, J., Musse, S.,: Detection of Unusual Motion Using Computer Vision. XIX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'06),pp.349-356, 2006
5. Jiang, F., Wu, Y., Katsaggelos, A.K.: Abnormal event detection based on trajectory clustering by 2-depth greedy search. IEEE International Conference on Speech and Signal Processing, 2008. ICASSP 2008.
6. Dickinson P., Hunter A.: Using Inactivity to Detect Unusual Behaviour. Proc. of IEEE Workshop on Motion and Video Computing, Colorado, January 2008.
7. Humphreys, J., Hunter, A.: Multiple object tracking using a neural cost function. Image and Vision Computing, June 2008.

8. Owens, J., Hunter, A., Fletcher, E.: Novelty Detection in Video Surveillance Using Hierarchical Neural Networks. International Conference on Artificial Neural Networks (ICANN 2002).
9. Owens, J. and Hunter, A.: Application of the Self-Organizing Map to Trajectory Classification. IEEE Computer Society Proceedings of the Third IEEE international Workshop on Visual Surveillance (Vs'2000) July 01, 2000
10. Johnson, N., Hogg, D.: Learning the distribution of object trajectories for event recognition. Proceedings of the 6th British Conference on Machine Vision (Vol. 2) 1995, UK, 583-592.
11. Piciarelli, C., Foresti, G.L., Snidara, L.: Trajectory clustering and its applications for video surveillance. IEEE Conference on AVSS, 2005.
12. Naftel, A., Anwar, F. B.: Visual Recognition of Manual Tasks Using Object Motion Trajectories. In Proceedings of the IEEE international Conference on AVSS, 2006.
13. Yue Zhou, Y., Yan, S., Huang, T.S.: Detecting Anomaly in Videos from Trajectory Similarity Analysis. IEEE International Conference on Multimedia and Expo, 2007
14. Duque, D., Santos, Henrique Dinis dos, Cortez, P.: Prediction of abnormal behaviors for intelligent video surveillance systems. Proceedings of IEEE Symposium on computational intelligence and data mining, USA, 2007
15. Stauffer, C., Grimson, W. E.: Learning Patterns of Activity Using Real-Time Tracking. IEEE Trans. Pattern Anal. Mach. Intell. 22, 8 (Aug. 2000), 747-757.
16. A. Hervieu, P. Bouthemy, J.-P. Le Cadre.: A statistical video content recognition method using invariant features on object trajectories. IEEE Trans. on CSVT (Special Issue on "Event Analysis in Videos"), 2008
17. C. Piciarelli, C. Micheloni, G.L. Foresti: Trajectory-based anomalous event detection. IEEE Transactions on Circuits and Systems for Video Technology, 2008.
18. C. R. Jung, L. Hennemann, S. R. Musse.: Event Detection Using Trajectory Clustering and 4D Histograms. Special issue on Event Analysis in Videos in IEEE Transactions on Circuits and Systems for Video Technology, 2008.
19. Appiah, K., Hunter, A.: A single-chip FPGA implementation of real-time adaptive background model. IEEE International Conference on Field-Programmable Technology, pp. 95-102, December 2005.
20. Meng, H.,Freeman, M.,Pears, N., Bailey, C.: Real-time human action recognition on an embedded, reconfigurable video processing architecture. Special Issue of Journal on Real-Time Image Processing, 163 - 176, Volume 3, Number 3, September, 2008.
21. Chalimbaud, P., Berry, F.: Embedded Active Vision System Based on an FPGA Architecture. EURASIP Journal on Embedded Systems Volume 2007 (2007).
22. Yamaoka, K., Morimoto, T., Adachi, H., Koide, T., Mattausch, H. J.: Image segmentation and pattern matching based FPGA/ASIC implementation architecture of real-time object tracking. In Proceedings of the 2006 Conference on Asia South Pacific Design Automation (Yokohama, Japan, January 24 - 27, 2006).
23. Matteo Tomasi, Javier Daz, Eduardo Ros: Real Time Architectures for Moving-Objects Tracking. Lecture Notes in Computer Science:Reconfigurable Computing: Architectures, Tools and Applications, Volume 4419/2007.
24. ABORAT Project: Eastern Kentucky University -College of Justice and Safety. www.jsc.eku.edu/projAborat.asp, 2006.
25. Han, C., Lin, C., Ho, G., Fan, K.: Abnormal Event Detection Using Trajectory Features. International Computer Symposium, Taiwan December 2006.
26. Kumar, S.: Neural Networks a classroom approach. McGraw-Hill, 2004.