# Facilitating Visual Surveillance with Motion Detections

Man Qi

Computing, Digital Forensics and Cybersecurity
Canterbury Christ Church University
Canterbury, Kent, CT1 1QU, UK

**Abstract**

Visual surveillance is playing an ever increasing role in criminal detection due to a rapid deployment of surveillance cameras. Motion detection which refers to the process of detecting a change in the position of an object in relation to the background or the change in the background in relation to the object has become one of the enabling techniques to facilitate visual surveillance. This paper parallelizes a motion detection algorithm using a cluster of inexpensive computing devices. Custom region of interest is implemented to enhance the performance and accuracy of the motion detection algorithm. The performance of the parallelized algorithm is evaluated from both the scalability in computation and the accuracy in motion detection. Performance evaluation results show that the enhanced algorithm achieves higher accuracy in motion detection with reduced execution times in computation.

Keywords: Visual surveillance, motion detection, message passing interface.

## 1. Introduction

Surveillance devices have been widely used by government and businesses for crime detection and prevention. The intelligent electronic systems perceive and understand the behaviour of people in image sequences to provide situational awareness. The importance of computer vision systems is gaining a momentum in present day civil and non-civil sectors. Computer vision systems have been used to enhance sectors such as traffic control, private and public surveillance, industrial automation via visual sensors, medical image analysis, biometric recognition systems, motion capture, Optical Character Recognition (OCR), Unmanned Arial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) [1].

In a broad sense, a major application of computer vision is automation in image based systems. Automation can be defined as the use or introduction of automatic equipment (e.g. machines, information systems and software) in a manufacturing or other process facility. In production systems, automation is used to increase system productivity and quality. Similarly, in computer vision, automation is used to increase system efficiency and accuracy beyond what is currently achievable by human labour capabilities. Some of the means by which automation is achieved in computer vision systems include motion detection, object recognition and object tracking.

Computer vision automation techniques can greatly increase productivity in the visual surveillance sector. According to a 2011 survey undertaken by the Deputy Chief Constable of Cheshire (in the UK), Graeme Gerrard and the Geographical Information Systems Administrator at the Cheshire Constabulary, Mr Richard Thompson estimates the number of CCTV cameras in the UK at a figure of 1,737,681 (approximately 1.8 million) [2].

There is currently no substantial research on the ratio of surveillance personnel to the number of CCTV or other video surveillance cameras. However, most non-domestic surveillance software systems are configured to display a minimum of 4 to 15 cameras per personnel. The accuracy and efficiency of such a purely human monitored system is only as reliable as the alertness of the personnel. Human alertness varies over time depending on the human factors, ergonomics, and also non-ergonomic factors. Motion detection techniques can be used as assistive technology to improve

the accuracy and productivity of visual surveillance systems. There are a number of algorithms that have been developed for motion detection [3, 4, 5]. However, as the number of cameras and the video quality is increased, the computational power and consequently the electrical power consumption increase. Thus the efficiency of both the motion detection algorithm and the computing architecture employed needs to be optimized. The aim of this optimization is to attain a balance between ensuring low energy usage and maintaining the performance of the surveillance system.

This paper parallelizes motion detection in surveillance videos using a cluster of inexpensive Raspberry Pi computers. Raspberry Pi is a fully fledged mini-computer in a credit card size. It has limited computing power with only 700MhZ-ARM11-processor and 512MB RAM (Model B+) but it is cheap, ready to use and meanwhile a large community exists which supports its development. Cox et al. [6] presented a 64 Raspberry Pi cluster and argued that such systems should be considered in some additional specialist application areas where these unique attributes may prove advantageous with a relatively little energy consumption. We build a Raspberry Pi cluster utilizing a 100Mbps network and employ MPI4Py [7] for parallel execution of motion detection and communication among the Raspberry Pi nodes using Message Passing Interface (MPI). Experimental results show that the Raspberry Pi cluster speeds up the computation significantly compared with the performance of using a single Raspberry Pi computer.

The rest of the paper is organized as follows. Section 2 gives a review on motion detection in visual surveillance. Section 3 presents the design of parallel motion detection using the Raspberry Pi cluster. Section 4 evaluates the performance of the parallel motion detection from the aspects of both accuracy in detection and efficiency in computation. Section 5 concludes the paper and points out some future works.

# 2. Motion Detection

Motion detection refers to the process of detecting a change in the position of an object in relation to the background or the change in the background in relation to the object. In visual systems, motion detection can be achieved by a multi-stage image processing and background subtraction approach. There are mainly two stages in motion detection in computer vision systems - object detection and background subtraction.

## 2.1 Object Detection

Object detection is a technology widely used in computer vision that entails the detection of instances of objects belonging to a certain category (e.g. humans, cars, or text) in digital images and videos. According to [8] edge detection is the first task in object detection. Thus, it is crucial to have a firm understanding of what edge detection is, how it has been implemented over the years, and how the information obtained by detecting the edges can be used to identify objects in the scene. Edge detection is a term which describes a group of mathematical techniques for identifying points in a digital image at which the image intensity or colour changes abruptly. The identified points are grouped into line segments referred to as edges. Edge detection is an essential technique in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction. Edge detection simplifies an image by extracting only the relevant features of the image thereby making it easier to analyse the image further. Ideally, when edge detection is performed on an image, it should detect the edges that represent the object's outer edges (boundaries), the boundaries of markings such as text which exist on surfaces, and changes in the orientation of the object surface.

Sharify et al. [9] presented a comparative study on edge detector and explicitly classified edge detectors into 5 broad categories:

- Gradient edge detectors
- Zero Crossing
- Laplacian of Gaussian
- Gaussian Edge Detectors

- Coloured Edge Detectors

*2.1.1 Gradient Edge Detectors*

This category includes classical operators and uses the first directional derivative (i.e. the gradient) operation from which the name is derived. Generally, the points with higher gradient magnitudes are regarded as strong edges. These algorithms may use a threshold value to select which magnitudes are valid edges.

*2.1.2 Zero Crossing*

This category includes edge detectors that use the Laplacian operator and second derivative operation to differentiate and identify the likely edges. The likely edges are the points where the magnitude of the second derivative operation is zero (hence the name zero-crossing). Edge detectors based on the second-derivative expression have a low tolerance to noise. This led to the invention of the 'Laplacian of Gaussian' (LoG) edge detectors.

*2.1.3 Laplacian of Gaussian (LoG)*

This category of edge detectors was invented by Marr and Hildreth in 1980. The LoG detectors use a Gaussian filtering for noise reduction and a Laplacian operation for finding the edges. Due to the development of more noise tolerant methods, this category of algorithms has seen less frequent usage in computer vision.

*2.1.4 Gaussian Edge Detectors*

This category of edge detectors utilizes functions that are symmetric along the edge. They are synonymous for having high tolerance to noise because they reduce the noise by smoothing the image. The major algorithms in this category are the Canny algorithm [9] and the ISEF (Shen-Castan) which convolve the image using a kernel [10].

*2.1.5 Coloured Edge Detectors*

This category includes edge detectors which operate on coloured images. They are divided into three sub-categories: output fusion methods, multi-dimensional gradient methods, and vector methods.

## 2.2 Background Subtraction

Background subtraction is a common approach for identifying moving object(s) in a video by differentiating them from the background. Background subtraction detects moving objects by subtracting the current frame from a reference frame generated by a background modelling algorithm. The reference frame must be kept up-to-date to compensate for variations in the scene. Most background modelling algorithms require that the camera be static [11]. Modern background modelling algorithms have extended the concept of "background subtraction" beyond its literal meaning and employ a range of techniques from simple ones, which aim to maximise speed and limit the memory usage, to more sophisticated ones which aim to achieve the highest possible accuracy under any possible circumstances [12]. All background modelling algorithms aim for real-time performance; hence there is a lower bound on speed. These models include:

- Running Gaussian average
- Temporal median filter
- Mixture of Gaussians
- Kernel density estimation (KDE)
- Sequential KD approximation
- Co-occurrence of image variations

- Eigen backgrounds

In this work, the Running Gaussian Average algorithm is employed for background subtraction due to its advantage of speed and low memory requirement which makes it highly scalable when used in large scale surveillance systems.

*2.2.1 Running Gaussian Average*

According to [12], the Running Gaussian Average background modelling algorithm is based on fitting a Gaussian probability density function on the last *n* pixel values. In order to avoid fitting the probability density function from scratch at each new frame time *t*, a running (or on-line cumulative) average is computed instead as:

$$\mu_t = \alpha I_t + (1 - \alpha)\, \mu_{t-1} \qquad (1)$$

In the equation (1), $I_t$ is the pixel's current value and $\mu_{t-1}$ is the previous average; $\alpha$ is an empirical weight often selected to attain a balance between model stability and the model being up-to-date. The standard deviation $\sigma_t$ of the current pixel value can be computed appropriately. In addition to high evaluation speed, the main advantage of the running average lies in its low memory requirement. Only two parameters (the average $\mu_t$, and the standard deviation $\sigma_t$) are stored instead of buffering the last *n* pixel values.

At each frame time *t*, the $I_t$ pixel's value can then be classified as a foreground pixel if the following inequality holds:

$$|I_t - \mu_t| > k\sigma_t \qquad (2)$$

Otherwise, $I_t$ pixel value will be classified as part of the background. In equation (2), *k* is the kernel and specifies how much a pixel's value may deviate from the standard deviation to be considered as part of the foreground. The name background subtraction is popularly used to indicate this set of techniques and is derived from equation (2).

Koller et al. [13] remarked that the background model in equation (1) is unduly updated also at the occurrence of such foreground values. For this reason, they proposed a modification to the way the background model is updated. In their modified model shown in equation (3), the binary value *M* is 1 in correspondence of a foreground value, and 0 otherwise. This approach is also known as selective background update.

$$\mu_t = M\mu_{t-1} + (1 - M)( (1 - \alpha)\, \mu_{t-1}) \qquad (3)$$

In their research, Wren et al. [14] proposed a model for intensity images; extensions can be made for multiple-component colour spaces such as (RGB colour format), (YUV colour format), and others. Moreover, if real-time requirements constrain the computational load, the update rate of either μ or σ can be set to less than that of the frame (sample) rate. However, it should be noted that the lower the update rate of the background model, the less a system will be able to respond quickly to the actual background dynamics. This may render the background model inaccurate when computational resources are low and the scene background is very dynamic.

Further research done to enhance background subtraction include [15] which proposed an approach based on neuronal mapping for segmentation of targets with hybrid background subtraction and adaptive mean shift jittering. With this method, scenes containing moving backgrounds and varying illumination changes can be considered effectively.

Gangodkar et al. [16] revealed that many of the proposed segmentation algorithms work well for the visible spectrum but fail to work for night vision thermal videos primarily due to the halo effect surrounding the object of interest. To solve this, they presented an approach that makes use of block matching algorithm for differentiating between background and moving objects. The full search or the exhaustive search being computationally expensive, they proposed a threshold based strategy that reduces the computational complexity to a large extent to make the solution suitable for real-

time applications. The reduced computational complexity makes it suitable for deploying on any low cost desktop computer and expanded in scale using a multi-threaded approach.

# 3. Parallel Motion Detection

In this section we present the design of the parallel implementation of motion detection using a Raspberry Pi cluster. First we give a brief introduction to MPI4Py which is employed for parallel execution of the motion detection algorithm.

## 3.1 MPI4Py

MPI for Python provides an object oriented approach to message passing which is based on the standard MPI-2 C++ bindings. The main package of MPI4Py which contains the core functionality of the MPI4Py is the MPI package. The core classes are described below.

- *Datatype*

This class matches and handles different data types.

- *Exception*

This class manages exceptions e.g. converting error codes into error messages.

- *Group*

This class manages the groups of processes e.g. comparing a group or making an union out of two groups.

- *Status*

This class represents a status of a request.

- *Op*

This class provides methods to create user-defined operations, which can be applied on data.

- *Errhandler*

This class is a simple error handler.

- *File*

This class provides methods for manipulating files.

- *Info*

This class provides methods to store an unordered set of key-value pairs.

- *Win*

This class provides methods to create a window object. Window object is a space in the memory of an intracommunicator group, which can be accessed by remote processes.

- *Request*

This class provides methods for managing requests e.g. cancel a request or wait for a request.

- *Grequest*

This class initiates a communication with a user-defined request.

- *Prequest*

This class initiates a communication with a persistent request.

- *Comm*

This class provides a communicator for communication between the processes e.g. distribute and gathering data between processes.

- *Intercomm*

This class provides an intercommunicator, which is used for communication within two or more groups of processes.

- *Intracomm*

This class provides an intercommunicator, which is used for communication within a single group of processes.

- *Cartcomm*

This class provides a cartesian topology intracommunicator.

- *Distgraphcomm*

This class provides a distributed graph topology intracommunicator.

- *Graphcomm*

This class provides a general graph topology intracommunicator.

## 3.2 Parallel Motion Detection

The Canny edge detector [9] is implemented for edge detection due to its high accuracy even in noisy conditions. To further improve efficiency in computation, an enhanced Canny with Region of Interest (ROI) is implemented which facilitates the objects which are within the custom ROI will be detected. The Running Gaussian Average algorithm is implemented for background subtraction. This is due to its advantage of speed and low memory requirement which makes it highly scalable when used in large scale surveillance systems.

The Canny edge detection detects edges at pixels where the intensity or colour changes the most. For grayscale images, the intensity is used while, for coloured images, the colour value for each channel is used. The edges are detected by determining the gradients in the image using the Sobel operator. The gradients are approximated in the horizontal and vertical directions by applying the following kernels:

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$ - applied for the horizontal gradient

$$K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$ - applied for the vertical gradient

The strength of the edge is determined by finding the magnitude of the gradient. This is achieved by applying the Pythagoras law or the Manhattan distance to reduce the computational complexity.

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$ - by Pythagoras law

$$|G| = |G_x| + |G_y|$$ - by the Manhattan distance

where $G_x$ and $G_y$are the respective gradients in the horizontal and vertical directions and $|G|$ is the edge strength.

By overlaying the computed edge on the smooth image, the edges are well detected. However, the detected edges are broad and do not give sharp indications of where the edges are. In order to sharpen the edges, the direction of the edges can be determined and used to make the edge trace more precise. The direction of the edges can be determined as the angle of the edge as obtained from the equation (4).

$$\theta = \tan^{-1}\left(\frac{|G_y|}{|G_x|}\right) \qquad (4)$$

Figure 1 shows the software architecture of the parallel implementation of the motion detector.

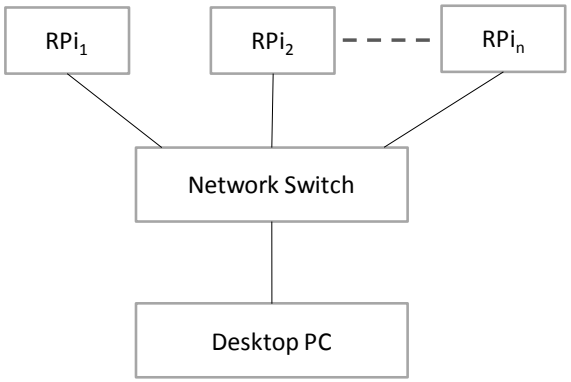


Figure 1: The software architecture of the motion detection system.

The desktop PC works as a master node using the *scatter* function in MPI4Py to distribute the video frames to Raspberry Pi (RPi) worker nodes and the *gather* function to collect the results. Figure 2 shows the workflow of the parallel motion detection algorithm.

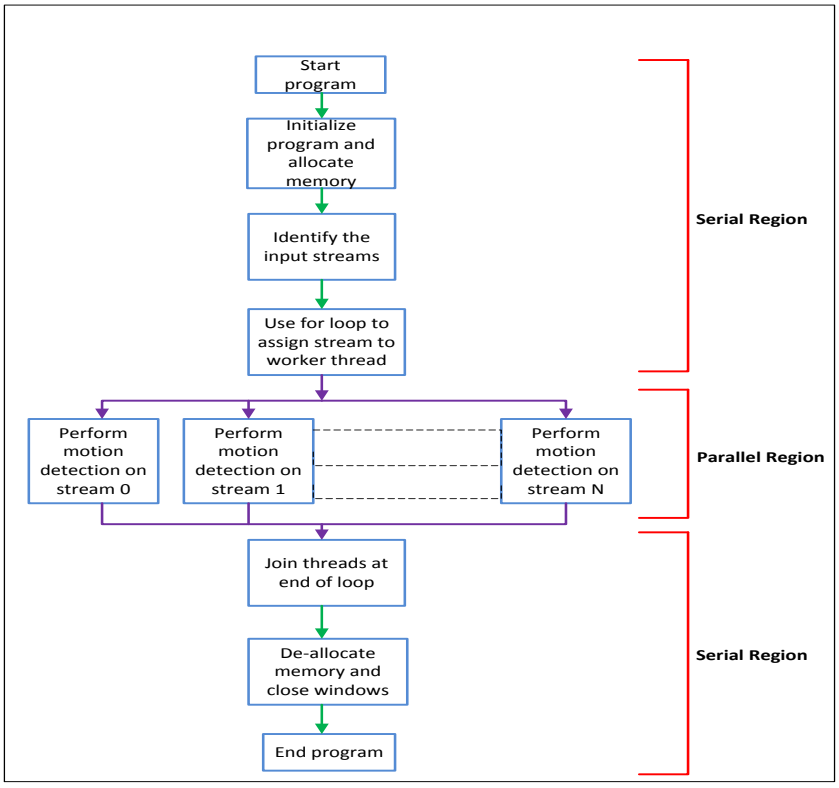


Figure 2. The workflow of motion detection.

## 4. Performance Evaluation

To evaluate the performance of the parallel motion detection algorithm, two experiments were conducted. The first experiment was conducted to measure the algorithm efficiency and compare it with that of two other algorithms. In this experiment, the accuracy of the algorithm was also compared with the accuracies of the other algorithms. The second experiment was conducted to measure the scalability of the algorithm using 10 Raspberry Pi Model B+ nodes. The desktop PC has an Intel Pentium Dual-Core CPU (T4200 @ 2 GHz) and 1.87 GB usable RAM running Ubuntu

12.04. MPI4Py was used for the desktop to communicate with the Raspberry Pi nodes using MPI. A video of 2400 frames was taken for testing purpose. Figure 3 shows a snapshot of video frame with detected objects.

Figure 3: The snapshot of a video frame with detected objects.

## 4.1 Computation Efficiency

In this test, the number frames that were fed to the motion detection algorithm was increased in steps of 200 frames till a total number of 2400 frames was reached. At each step, the total time taken by the algorithm to process the frames was recorded. The computation efficiency of the *Canny with custom ROI* algorithm was compared with that of the *Absolute Difference* algorithm, the *Canny without custom ROI* algorithm and the *Laplacian* algorithm respectively as shown in Figure 4.

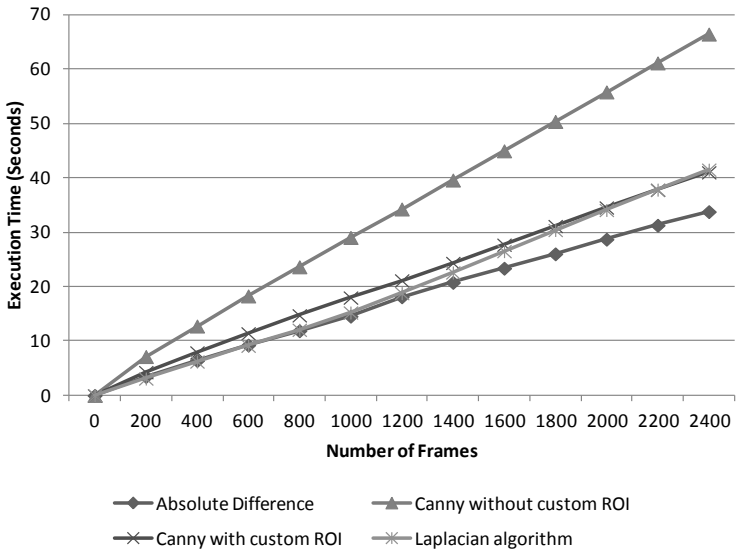


Figure 4: Computation efficiency.

As seen from the graphs in Figure 4, the execution times of all the four algorithms increase linearly with as the number of image frames is increased. It can also be observed that the disparity between the execution times of the algorithms increases as the number of image frames increase. The *Canny with the custom ROI* algorithm is faster in computation than the *Canny without the custom ROI*. This performance gain is achieved due to the fact that only the selected rectangular section(s) of a video frame is processed.

## 4.2 Scalability in Parallelization

A number of tests were conducted to evaluate how the motion detection algorithm scales in computation. It can be seen from Figure 5 that the execution time consumed by each video frame decreases with an increasing number of Raspberry Pi (RPi) nodes in the four scenarios using 500 frames, 1000 frames, 1500 frames and 2000 frames respectively. However, the parallelization achieves the best performance in scalability in the scenario of using 500 frames, whereas it achieves the worst scalability in the scenario of using 2000 frames. This is mainly because that more video frames are used in the detection process, a higher overhead in communication among the RPi nodes will be incurred.

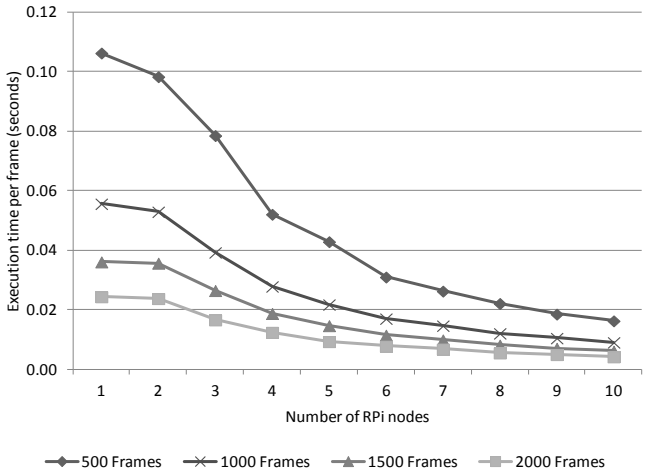


Figure 5: Scalability in computation.

## 4.3 Accuracy in Motion Detection

In motion detection, the detection accuracy is a measure of the correctness of the decision making ability of the motion detection algorithm. For every video frame processed, three decisions can be made which are true positive, false positive and false negative.

It is important that the total number of detections made by an algorithm is limited to only the relevant instances of moving objects. If a large number of irrelevant detections are made, the amount of storage available to store the detected images will be inefficiently used. Figure 6 shows the total number of detections achieved by the four algorithms .

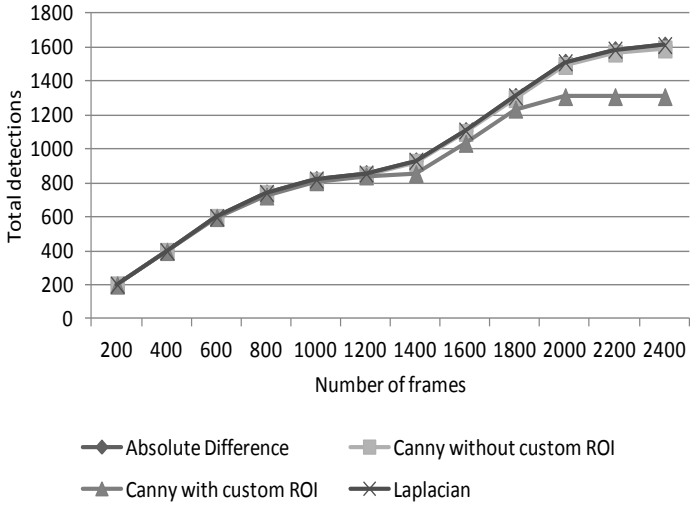


Figure 6: Total detections.

It can be observed that the total number of detections increased generally as the number of frames was increased for all the four algorithms. From 1200 frames till 2400 frames, the number of detections obtained for the *Canny algorithm with custom ROI* was less than the total detections obtained for the other algorithms. The *Absolute difference*, *Laplacian* and *Canny without custom ROI* produced generally about the same number of detections, although *Canny without custom ROI* returned slightly less detections from 1800 to 2400 frames.

The *Absolute Difference* algorithm yields the highest number of detections. This is so because the *Absolute Difference* algorithm obtains the absolute value of the result of the subtraction of the pixels in the current image frame from the corresponding pixel in the previous one. The operation of the Absolute Difference algorithm is given by equation (5)

$$Diff = |I_i - I_{i-1}| \tag{5}$$

Where:

- $Diff$ is the absolute difference.
- And $I_i$ is the pixel value of a pixel in the current image.
- And $I_{i-1}$ is the pixel value of a pixel in the previous image.

Any change in pixel values due to factors such as noise will yield a difference which will be detected as motion by the *Absolute Difference* algorithm. Thus, this algorithm is highly affected by noise and yields the most number of detections (including false detections). Despite its low noise tolerance, the *Absolute Difference* algorithm is simple and thus requires minimal computation and minimal memory.

The *Laplacian* algorithm is more complex than the *Absolute Difference* algorithm because it uses a Laplacian of Gaussian (LoG) operation to find the areas of rapid change (i.e. the edges) in a video frame. The LoG operation is a two-step process that first performs a smoothing operation on the image using a Gaussian filter to reduce noise and then performs the *Laplacian* operation. The *Laplacian* operator is essentially a derivative in the x and y pixel directions. In the LoG, a video frame $I(x, y)$ is convolved by applying the Gaussian kernel using equation (6).

$$g(x, y, t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/(2t)} \tag{6}$$

The kernel is applied at a given scale $t$ to give the scale space representation in equation (7).

$$L(x, y, t) = g(x, y, t) * I(x, y) \tag{7}$$

Then, the *Laplacian* operator is computed with equation (8).

$$\nabla^2 L = L_{xx} + L_{yy} \tag{8}$$

The above form of the *Laplacian* operator usually results in strong edges for dark blobs and weak edges for bright blobs of similar size. Thus, it can be said that it is partial to extreme intensities. A major challenge faced when applying the *Laplacian* operator at a fixed scale, however, is that its detection depends on the size of the BLOB structures in the image with respect to the size of the Gaussian kernel used for pre-smoothing process. In order to automate the detection of BLOBs of any size in the image domain, a multi-scale approach is, therefore, necessary. A straightforward way to obtain a multi-scale blob detector with automatic scale selection is to consider the scale-normalized *Laplacian* operator:

$$\nabla^2{}_{norm} L(x, y, t) = t\big(L_{xx} + L_{yy}\big) \tag{9}$$

The *Laplacian* algorithm is more tolerant to noise than the *Absolute Difference* algorithm also and yields less false detections than the *Absolute Difference* algorithm. Thus, the *Laplacian* algorithm would yield slightly less detections than the *Absolute Difference* algorithm when the images contain minimal noise. However, the difference in the number of detections between both algorithms will

increase as more noise is introduced to the images. The complexity of the *Laplacian* algorithm makes it slower and more memory demanding than the *Absolute Difference* algorithm. Figure 7 shows the performance of the four algorithms in terms of true positives. It is worthy to note that all the four algorithms yielded an equal number of true positives. No algorithm was found to yield a higher number of detections of images containing actual moving objects than the other. However, the *Canny with custom ROI* algorithm produced less false detection than the other algorithms as shown in Figure 8.

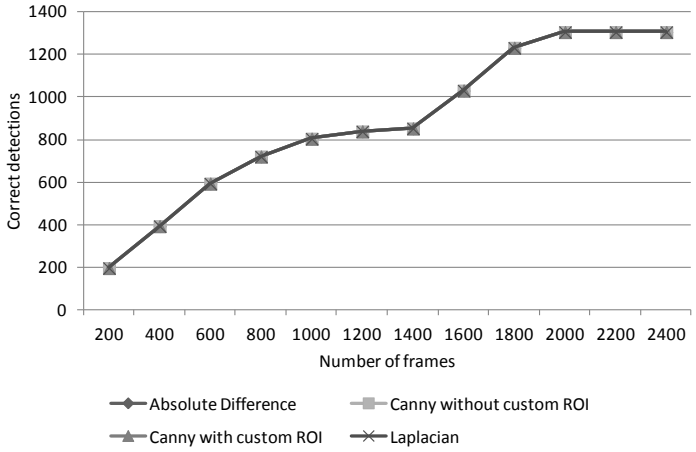


Figure 7: True positives.

From figure 8 it can be observed that the *Canny with custom ROI* algorithm produced zero false positives while the other three algorithms produced significantly a high number of false positives. It can also be observed that the number of false positives detected remained constant at zero for the *Canny with custom ROI* algorithm. However, for the other three algorithms, the number of false positives increased gradually as the number of frames is increased.

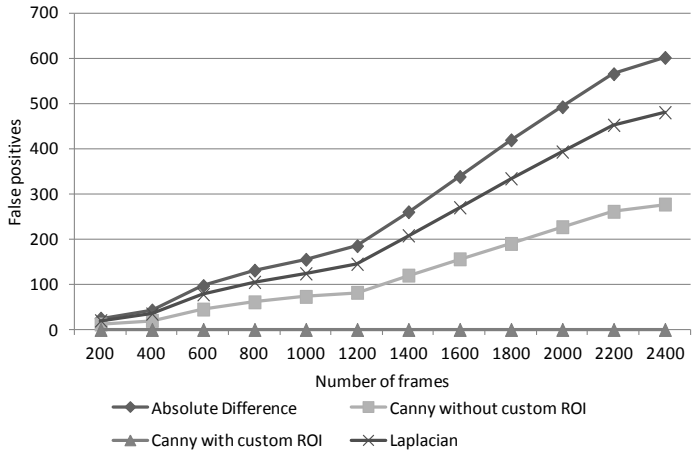


Figure 8: False positives.

A false negative is a video frame which contains a moving object, but which was not detected by the motion detection algorithm. It is important to determine the number of false negatives (if any) made by the *Canny with custom ROI* algorithm and compare it with the other three algorithms. Figure 9 shows the false negatives for all four algorithms.

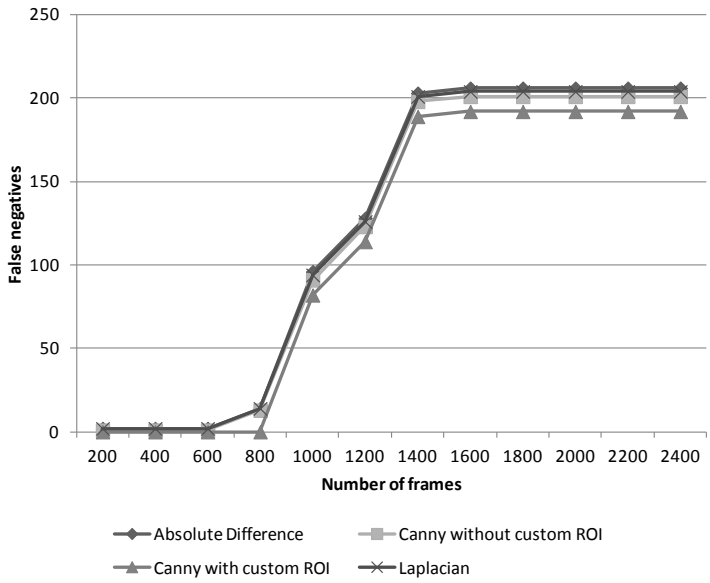


Figure 9: False negatives.

From Figure 9, the most obvious observation that can be made is that the numbers of false negatives returned by all the four algorithms were quite low (about 1 to 2 occurrences) between 200 to 600 frames. Beyond 600 frames, the numbers of false negatives increases rapidly and eventually settle at about an average of 200 occurrences at 1400 frames. The number of false negatives remains mostly unchanged till 2400 frames.

It can also be observed that the introduction of the custom ROI had no effect on the number of false negatives. Thus the *Canny with custom ROI* algorithm and the *Canny without custom ROI* algorithm both produced an equal number of false negatives. The *Absolute Difference* and the *Laplacian* algorithms produced the least number of false negatives.

## 4.4 Precision and Recall in Motion Detection

The precision (also called positive predictive value) is the fraction of detected video frames that are relevant. A high precision implies that an algorithm retrieves substantially more relevant detections than irrelevant ones. In classifying the detections, the precision for a group of detections is the number of true positives (i.e. the number of detections which contain moving objects that are of interest) divided by the total number of detections which contain moving objects which are relevant or irrelevant (i.e. the sum of true positives and false positives, which are video frames incorrectly detected as containing moving objects) [17].

Thus the precision is given by the equation (10).

$$Precision = \frac{Number\ of\ true\ positives}{Number\ of\ true\ positives + Number\ of\ false\ positives} \qquad (10)$$

Figure 10 shows the precision results of the four algorithms from which it can be observed that the precision was highest when the custom ROI was applied to the *Canny* algorithm. However, without the custom ROI, the precision of the *Canny* algorithm was significantly lower. Low precision values were recorded for both the *Absolute Difference* and *Laplacian* algorithms. It should be noted that the lowest precision was obtained for the *Absolute Difference* algorithm. It can also be observed that the precision remains stable at a value of 1 regardless of the number of frames when the custom ROI was applied to the *Canny* algorithm. However, there was a general decline in the precision for other three algorithms as the number of frames was increased.

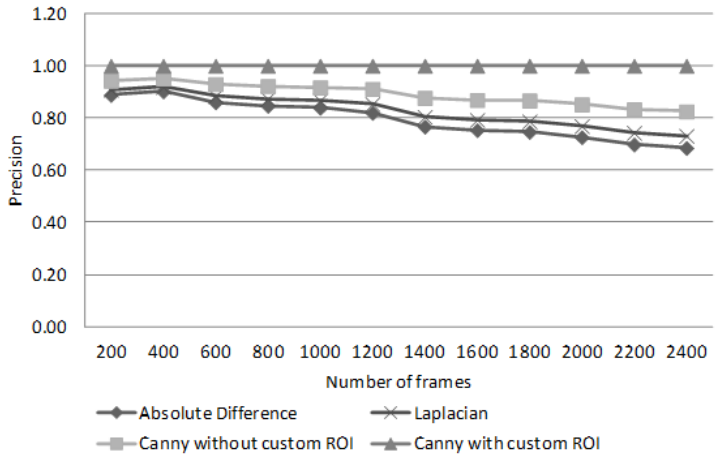


Figure 10: Precision in motion detection.

The recall (also known as sensitivity) is the fraction of relevant detections that are made. It is also a measure of completeness or quantity of the obtained results. A high recall means that an algorithm detectes most of the relevant moving objects in the scene. Recall in this context is defined as the number of true positives divided by the total number of detections that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are video frames which were not detected as containing moving objects , but actually do) [17]. Thus the recall is given by the equation (11).

$$Recall = \frac{Number\ of\ true\ positives}{Number\ of\ true\ positives + Number\ of\ false\ negatives} \qquad (11)$$

Figure 11 shows the recall results of the four algorithms from which it can be observed that the recall values are slightly higher when the custom ROI is applied to the *Canny* algorithm. However, this margin is just about 0.15. It was also observed that there was a gradual decline in the recall values for all the four algorithms. The lowest algorithm recall values were obtained at 1400 frames. The recall values increase gradually but eventually settle at about 0.86 at 1800 frames.

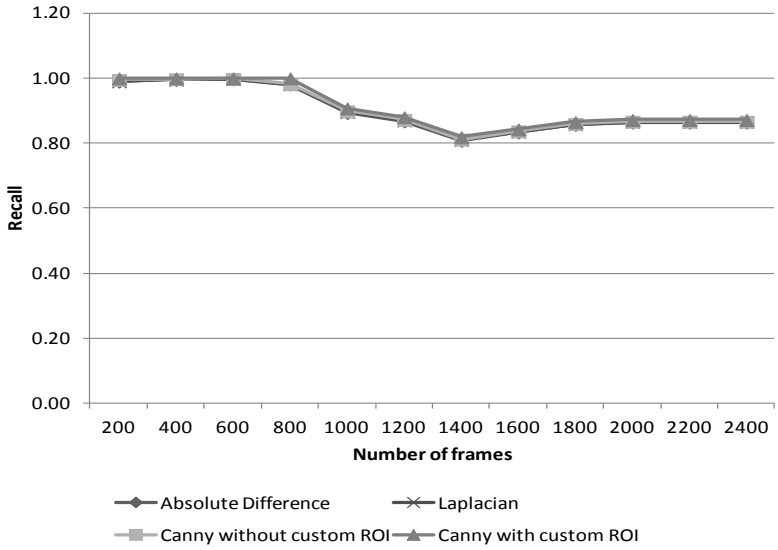


Figure 11: Recall in motion detection.

## 5. Conclusion

In this paper, we have presented a parallel implementation of the *Canny* algorithm for high performance motion detection in surveillance videos. The parallel motion detection system run on a cluster of inexpensive computing devices. Custom region of interest was implemented to enhance the performance and accuracy of the Canny algorithm. Performance evaluation results showed that the enhanced algorithm achieved higher accuracy in motion detection with reduced execution time in computation.

In order to further automate the system, a learning algorithm can be developed to dynamically generate which parts of the scene need to be excluded without a need for a manual custom region of interest selection. Since most of the unwanted detection are likely to be caused by trees, the algorithm may be designed to identify and exclude regions containing high concentration of green pixels. This may add to the overheads but will certainly reduce the time required to deploy the motion detection system.

Acknowledgement

The author would like to acknowledge the Research Fund received from the School of Law, Criminal Justice and Computing, Canterbury Christ Church University, UK.

## References

1. R. Szeliski, Computer Vision: Algorithms and Applications, 2010, London: Springer.
2. CCTV User Group, 2011, UK,
3. R. Cucchiara, C. Grana, M. Piccardi, A. Prati, Detecting Moving Objects, Ghosts, and Shadows in Video Streams, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 10, pp. 1337 - 1342, 2003.
4. T. Thongkamwitoon, S. Aramvith, T. H. Chalidabhongse, An adaptive Real-Time Background Subtraction and Moving Shadows Detection, Proceedings of the International Conference on Multimedia and Expo (ICME), pp. 1459-1462, 2004.
5. F. Hu, Y. Zhang, L. Yao, An Effective Detection Algorithm for Moving Object with Complex Background, Proceedings of the International Conference on Machine Learning and Cybernetics, pp. 5011-5015, 2005.
6. S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, N. S. O'Brien, Iridis-Pi: A Low-Cost, Compact Demonstration Cluster, Cluster Computing, vol. 17, no. 2, pp. 349-358, 2014.
7. L. Dalcin, MPI4Py, https://mpi4py.scipy.org/docs/mpi4py.pdf [Last Accessed on 21 November 2015]
8. R. Maini, J. S . Sohal, Performance Evaluation of Prewitt Edge Detector for Noisy Images, GVIP Journal. vol. 6, no. 3, pp.39-46, 2006, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.3340&rep=rep1&type=pdf [Accessed: 21/11/2015].
9. J. Canny, A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679-698, 1986.
10. J. Shen and S. Castan, An Optimal Linear Operator for Step Edge Detection, Journal of CVGIP: Graphical Models and Image Processing, vol. 54, pp. 112-133, 1992.
11. Y. Ming, B. Yang, A. Men, Y. Guo, Background Subtraction under Single Varying Illumination, Proceedings of the 2nd IEEE International Conference on Broadband Network and Multimedia Technology, pp. 143-146, 2009.
12. M. Piccardi, Background Subtraction Techniques: A Review, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. vol. 4. pp. 3099-3104, 2004.
13. D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, Towards Robust Automatic Traffic Scene Analysis in Real-Time, Proceedings of the 33rd IEEE Conference on Decision and Control, vol. 4. pp. 3776-3781, 1994.
14. C. Wren, A. Azarbayejani, T. Darrell and A. Pentland, Pfinder: Real-Time Tracking of the Human Body, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 780–785, 1997.

15. R. Athilingam, K.S. Kumar and G. Kavitha, Neuronal Mapped Hybrid Background Segmentation for Video Object Tracking, Proceedings of the International Conference on Computing Electronics and Electrical Technologies (ICCEET), pp. 1061-1066, 2012.
16. D. Gangodkar, P. Kumar and A. Mittal, Segmentation of Moving Objects in Visible and Thermal Videos, Proceedings of the International Conference on Computer Communication and Informatics (ICCCI), pp. 1-5, 2012.
17. J. Kaur, S. Gupta, S. Kundra, A k-means Clustering Based Approach for Evaluation of Success of Software Reuse, Proceedings of International Conference on Intelligent Computational Systems (ICICS'2011), pp. 1-4, 2011.