National College of Ireland

MSc Web Technologies

September 2015

Name: Denis Stepanenko

Student Number: x13100793

Email: denis.step88@gmail.com

Evaluation of Background Segmentation Algorithms on Embedded Devices

Declaration of Authorship

I hereby certify, that this material which I now submit for assessment leading to the award of master of science in web technology is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

____

Date:

____

Abstract

The CCTV monitoring operators miss up to 95% of intrusions when monitoring multiple cameras simultaneously (Hyenkyun, et al., 2010). In order to tackle this problem, many monitoring centres utilize various motion detection approaches, as opposed to constantly watching the screens. These approaches work well in illuminated environments or indoors, but outdoors these tend to generate multiple false alarms. The problem is even worse when the camera utilizes Infra-Red lighting, as noise motion such as drops of rain, cobwebs and flies are much more visible. Since every detection needs to be investigated, operators are able to monitor less cameras than they would otherwise, which increases the monitoring service price.

One potential solution to false detections is the installation of motion sensors, such as PIRs. However, these tend to be expensive, require additional wiring and hardware, and can be extremely unreliable.

This research investigates if the latest background segmentation algorithms can be utilized to suppress noise objects efficiently enough, to be utilized on embedded devices such as those used by IP cameras. To answer this question, a number of utilities were developed to allow the testing of problematic feeds and analysing results. In addition, a number of post-detection filters were built to suppress false detections even further.

The tests were carried out on a variety of video feeds containing intruders and noise objects. These tests were carried out on a laptop and various embedded devices such as Raspberry PI. The research methodology used was quantitative. The analysis of the data shows that a significant amount of noise objects can indeed be suppressed, with acceptable decrease in FPS rate.

Acknowledgements

I would like to thank all the people who helped me throughout this project.

To Mr Vikas Sahni, my supervisor, thank you for all your time, especially for when you helped even when you were away on your holiday in India.

To Marc Van Droogenbroeck, who provided me with ViBe source code.

To everyone in Inventise Business Solutions for your help, support and encouragement.

To my wife Kseniya, for all your help, support and patience. Without your support I wouldn't have finished it.

Table of Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures vii

List of Tables ix

Abbreviations x

Terminology xi

1 Introduction 1

1.1 Literature Review 1

1.1.1 Motion Detection and Analysis with Four Motion Detectors 3

1.1.2 Video Analytics Algorithm for Detecting Objects Crossing Lines In Specific Direction Using Blob-Based Analysis 5

1.1.3 Adaptive Motion Detection Algorithm using Frame Differences and Dynamic Template Matching Method 6

1.1.4 Environmentally Robust Motion Detection for video surveillance . 7

1.1.5 The OBSERVER: An Intelligent and Automated Video Surveillance System 9

1.1.6 ViBe: A universal background subtraction algorithm for video sequences 11

1.1.7 Background Subtraction Based on a Robust Consensus Method 14

1.1.8 Motion Detection for Video Surveillance 16

1.1.9 A new motion detection algorithm based on Σ-Δ background estimation 17

1.1.10 The Pixel-Based Adaptive Segmenter (PBAS) 19

1.2 Research Question 21

1.3 Conclusion 21

1.3.1 Chapter 2 22

1.3.2 Chapter 3 22

1.3.3 Chapter 4 22

1.3.4 Chapter 5 22
2 Theory and Background..23
2.1 Definition of "real-time" 23
2.2 Video Processing Devices..24
2.3 Low-Resolution Frame Processing..24
2.4 Computer Vision Framework 25
2.5 Types of Motion 26
2.6 Background Subtraction 28
2.7 Adaptive Background Subtraction 28
2.8 Performance Evaluation 29
2.9 Conclusion..30
3 Segmentation Tester..32
3.1 Testing Harness..33
3.1.1 Frame Acquisition..34
3.1.2 Test Results..34
3.1.3 Performance Testing 35
3.2 Target Devices..35
3.2.1 Motion Detection Algorithms..37
3.2.1.1 Algorithm Configuration 37
3.2.2 Post-detection Filters..38
3.2.2.1 Transparency Removal Filter 38
3.2.2.2 Colour-based Filter 39
3.2.2.3 Heat Map Filter 39
3.2.2.4 Custom Background Model..40
3.2.3 Ground Truth Comparison..41
3.3 Test Result Analysis Tool 42
4 Results and Analysis..44
4.1 Methodology..44
4.2 Model..44
4.3 Data Sets..45
4.4 Conditions Tested 45
4.5 Conditions Excluded..47
4.6 Experiment Results 47

4.7 Conclusion....56
5 Conclusion and Future Work....58
5.1 Conclusion....58
5.2 Future Work....59
References....60
Appendix 1 - Test Results....64
Appendix 2 – User guide....67
Appendix 3 – Compilation Process....69
Appendix 4 – Testing Harness Command Line Arguments....72
Appendix 5 – Segmentation Algorithms Configuration....75
Appendix 6 – CSV Log Sample....77

List of Figures

Figure 1: Outdoor Infra-Red sample 2
Figure 2: Indoor Infra-Red sample (Drinkwater, 2010) 2
Figure 3: The OBSERVER motion segmentation (Duarte, et al., 2006) 10
Figure 4: ViBe's background detection illustration (Olivier & Marc, 2011) 12
Figure 5: ViBe PCC rates achieved by the authors (Olivier & Marc, 2011) ... 13
Figure 6: ViBe FPS rates achieved by the authors (Olivier & Marc, 2011) ... 14
Figure 7: SECON Flow Diagram (Hanzi & David, 2006) 15
Figure 8: PCC results comparison between ViBe and PBAS algorithms achieved by Hofmann, et al 20
Figure 9: Wide angle frames with snow too close to the lens(Anon., 2010) . 25
Figure 10: Contrast difference between a genuine moving and over-saturated object 27
Figure 11: Semi-transparent cobweb over saturating the frame 27
Figure 12: Wide angle frames with snow too close to the lens (Anon., 2010) 27
Figure 13: Test Result Analysis Tool screenshot 42
Figure 14: FPS comparison from low and high resolution tests 48
Figure 15: Post-detection filter PCC rate comparison 49
Figure 16: FPS rates comparison between algorithms and filters. Test conducted on a Raspberry PI. 51
Figure 17: FPS rates comparison between algorithms and filters. Test conducted on a Orange PI. 51
Figure 18: FPS rates comparison between algorithms and filters. Test conducted on a Banana PI. 52
Figure 19: Saturation Remover filter PCC rates comparison 53
Figure 20: PCC rates achieved from feeds containing cobwebs 54
Figure 21: PCC rates achieved from "Change Detection Challenge" data sets 55
Figure 22: Noise object suppression example (no intruder) 55
Figure 23: Noise object suppression example (intruder present) 56

List of Tables

Table 1: PCC rates achieved by Zulaikha, et at. 5
Table 2: Device Specification 36
Table 3: List of conditions used for each test run 47
Table 4: FPS increases proportional to PCC increases for Heatmap filter on Orange PI 50
Table 5: FPS increases proportional to PCC increases for Transparency Removal filter on Orange PI 50
Table 6: FPS increases proportional to PCC increases for Saturation Remover filter 54
Table 7: FPS and PCC comparison from low and high resolution tests conducted on a laptop 64
Table 8: Post-detection filters PCC rate comparison 64
Table 9: Saturation Remover filter PCC rates comparison 64
Table 10: FPS rates comparison between algorithms and filters. Test conducted on a laptop 65
Table 11: FPS rates comparison between algorithms and filters. Test conducted on a Raspberry PI 65
Table 12: FPS rates comparison between algorithms and filters. Test conducted on a Orange PI 65
Table 13: FPS rates comparison between algorithms and filters. Test conducted on a Banana PI 66
Table 14: PCC rates achieved from "Change Detection Challenge" data sets 66
Table 15: PCC rates achieved from feeds containing cobwebs 66

Abbreviations

PCC	Percentage of Correct Classification
PBC	Percentage of Bad Classification
GUI	Graphic User Interface
RTSP	Real Time Streaming Protocol
RGB	Red Green Blue
XML	Extensible Markup Language
CSV	Comma Separated Values
CCTV	Close-circuit Television
PIR	Passive Infra-Red (sensor)
IP	Internet Protocol
DVR	Digital Video Recorder
MP	Mega Pixel

Terminology

Foreground	Moving object in the field of view
Foreground Mask	Bit mask where value 1 represents foreground and 0 background pixels
Background	Pixels in the view that are typically static and don't represent the moving object
Ghost Object	Object mistakenly identified as foreground
Noise Object	Object identified as foreground but which the monitoring operator does not need to know about (i.e. cobweb, rain)
Segmentation	The process of separating the foreground from the background

1 Introduction

CCTV monitoring operators miss up to 95% of intrusions when monitoring multiple cameras simultaneously (Hyenkyun, et al., 2010). In order to tackle this problem, many monitoring centres utilize various motion detection approaches, as opposed to constantly watching the screens. These approaches work well in illuminated environments or indoors, but outdoors these tend to generate multiple false alarms. The problem is even worse when the camera utilizes Infra-Red lighting, as noise motion such as drops of rain, cobwebs and flies are much more visible.

One potential solution to false detections is the installation of motion sensors, such as Passive Infra Red (PIR). However, these tend to be expensive, require additional wiring and hardware, and can be extremely unreliable.

The following literature review looks into various approaches of the foreground/background segmentation to identify algorithms efficient enough to be run on embedded devices, such as those used in IP cameras or DVRs, while capable of suppressing the detection of noise objects.

1.1 Literature Review

Mishra et al conducted a study which showed that there are three sensible ways to detect motion in the series of frames; those are background subtraction, optical flow and temporal differences (Mishra, et al., 2011). However, because optical flow algorithms require special hardware assistance (Widyawan & Muhammad, 2012), this type of algorithms will not be investigated further as the target devices required for applications such as IP cameras, are limited in resources and processing power.

The background subtraction is performed by comparing new frames against a particular static frame. The temporal differences approach is very similar, with the exception that there are multiple static frames available to subtract the incoming frame from. The incoming frame is not typically subtracted from all of the frames in the set, but instead the algorithm selects dynamically one frame that should be used for subtraction (Mishra, et al., 2011).

In both processes, the pixels similar to both frames (incoming and reference) are removed, resulting in a bit map, where some of the pixels could represent a moving object and others different parts of the background. This approach is reasonably simple to implement and it is considered to consume the least number of CPU cycles due to lack of complex mathematical calculations. However, although its accuracy can be acceptable indoors, it might not be acceptable in the outdoors environment. This is primarily due to the fact that indoor feeds tend to have clearer images than outdoors, where rain, snow, dust, sudden light intensity changes and other environmental noise constantly changes the frame contents, resulting in false detections. Furthermore, because the outdoor scenes generally cover larger areas, the resulting Infra-Red image can be much darker outdoors than indoors, as can be compared below.

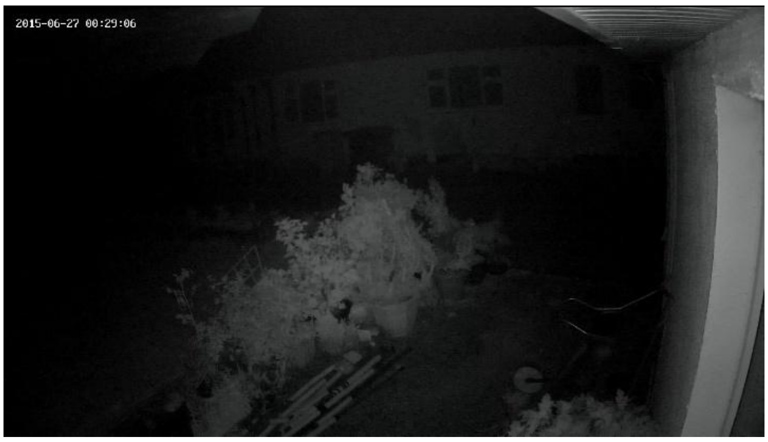


Figure 1: Outdoor Infra-Red sample

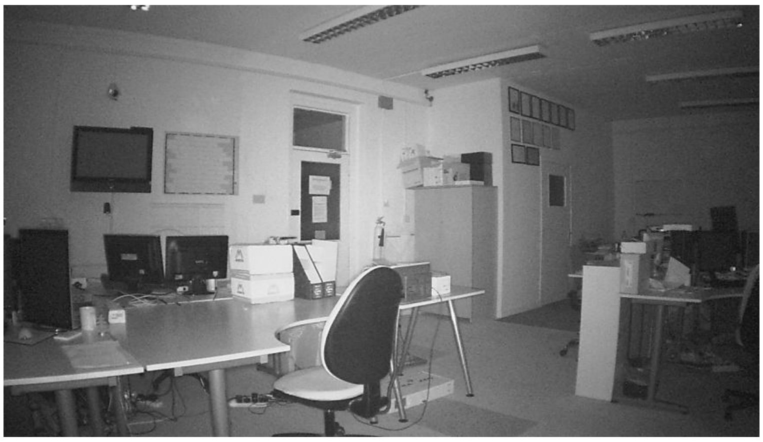


Figure 2: Indoor Infra-Red sample **(Drinkwater, 2010)**

1.1.1 Motion Detection and Analysis with Four Motion Detectors

Ching, et al. developed a motion detection system utilizing four of the most popular algorithms, which were: current vs. previous frame subtraction, pixilation, blob counter and morph filter. The algorithms were modified in order to optimize their performance. These algorithms were implemented in C# and Matlab programming languages. (Ching, et al., 2011)

The aim of their research was to determine which of the four algorithms would yield best results with the least CPU load. However, although the authors stated that certain approaches proved to be the most accurate and CPU-friendly, a limited amount of data was included in the paper.

The developed system was able to estimate detected object's speed, which was then used to determine the alarm level. However, although the speed can certainly indicate whether the detected motion is environmental noise or real motion object (e.g. intruder), the program, theoretically, would have been able to provide much more accurate results if the alarm level was calculated based on object's speed and size together. This is because the smaller the real object is, the slower it would move as opposed to a drop of rain, which depending on the distance from the lens, could be about the same size as the intruder, but would move much faster. Zulaikha, et al. took a similar approach in order to tolerate noise, but have observed that lower frame rates can have an adverse effect on this kind of algorithm (Zulaikha, et al., 2012).

The first approach used in this paper is current vs. previous frame, which subtracts one grey-scaled frame from another. Following this, the erosion filter was applied to remove sparse pixels which don't normally represent an object (Heijmans & Ronse, 1990). Then the remaining pixels are counted and if the total exceeds a specified threshold, the alarm was triggered (Stanley R. Sternberg, CytoSystems Corporation, 1983). However, according to the research conducted by Ching Yee Yong, et al., the problem with this algorithm is that it cannot detect motion if the object is moving too slow. Another slight variation of this algorithm is to compare the current frame to the first frame acquired from the video sequence. The only difference is that the first frame is gently changed one colour tone at a time. This is required to ensure that over time the foreground doesn't become too different from the real background, as otherwise a ghost object would be detected (Tetsuya, et al., 2010).

The second approach used in this paper is the pixilation filter, which evaluates the mean saturation value for pixels within a fixed size square. As a result, the frame would become a grid of saturation means, which is then compared against a grid from the previous frame. This approach can potentially save CPU cycles as the mean doesn't have to be calculated for every pixel, but for every *n*th pixel (where *n* is a variable), as according to (Duarte, et al., 2006) and (Birmohan, et al., 2014), neighbouring pixels tend to be similar. However, as described in the paper, this algorithm has a potential flaw. Because the adjacent HUE indices can signify completely different colours, the mean saturation can cause false alarms.

The third approach used in this paper is the blob counter, which extracts pixels that fall into a particular range of colours. The blob denotes connected motion pixels in motion map, which can also be acquired using simple background subtraction techniques. This approach is a popular choice in applications where object tracking is required (Zulaikha, et al., 2012). Once the blob is identified, its shape and size is checked to determine if it matches the shapes defined by an operator beforehand, and if it does, the tracking can start and alarm can be raised.

The final approach used by (Ching, et al., 2011) was the morph filter, but there was not enough information given as to how it was utilized and the results it produced.

The test results of the system showed that it was able to separate the background from the foreground, although the maximum frame rate achieved was approximately 6 FPS. The accuracy of the detection was not specified.

It was observed by the authors that faster frame rates (circa 35fps) can be handled by the system using the morph filter.

1.1.2 Video Analytics Algorithm for Detecting Objects Crossing Lines In Specific Direction Using Blob-Based Analysis

Zulaikha, et al. conducted a study and developed a system for detecting, tracking and notifying surveillance operators of potential perimeter breaches. The system was mainly based on simple background subtraction and blob counting algorithms. The system was able to outperform one of the commercial products by approximately 5-10% in terms of accuracy of detection. However, the results cannot be considered very accurate as very little is known about the commercial system and no other reference data was provided. Furthermore, the comparison between the results achieved in clear and raining weather is not clear as there is no mention whether the test was conducted during the night or day.

	Day	**Night**	**Clear Weather**	**Raining**
Proposed	97.48%	81.45%	90.59%	76.28%
Commercial	90.34%	68.01%	80.93%	59.27%

Table 1: PCC rates achieved by Zulaikha, et at.

This is important as the authors observed that Infra-red (IR) lighting, bad weather and bad weather together with IR lighting causes both systems (commercial and proposed) to produce significantly more false alarms, than video sequences acquired during the day with good weather. (Zulaikha, et al., 2012)

The performance tests showed that the proposed approach was able to achieve real-time frame rates (over 25 FPS), but the frame size and host machine parameters were not specified.

One strong feature of the approach taken by Zulaikha, et al. is the virtual boundary functionality, which acts as a mask, so that only pixels within the boundary are processed. As a result, the amount of data to be processed had decreased, yielding better performance. Furthermore, virtual boundaries can be utilised to mask noisy areas of the view, such as moving water or trees, in effort to reduce false detections. The system also implemented the "*tripwire*" functionality which in essence is just a virtual line on the frame, which once crossed, causes the alarm to go off. But unlike the "*tripwire*" algorithm proposed by (Yun & Arun, 2009), it was configurable to only raise alarms if the tripwire was crossed in only one direction. Therefore, the algorithm proved to be efficient in areas of dense traffic.

The background modelling was performed by selectively updating the average value of each pixel that wasn't part of a moving blob. The background model was constantly updated. When a pixel within a given threshold reappeared in frames to follow, it was considered to be part of the blob, and was given a value of 1, alternatively, the pixel was given a value of 0. This resulted in a pixel map, denoting moving blobs, which were then passed on to the object tracking algorithms. This approach helped eliminate false detections caused by rain or other fast-moving noise objects.

In order to track each blob of pixels, a simplified "*particle filter*" algorithm was utilized (Sze, et al., 2010), where each blob was assigned at least one tracker, which would estimate object's position on the frame.

1.1.3 Adaptive Motion Detection Algorithm using Frame Differences and Dynamic Template Matching Method

Widyawan & Muhammad have conducted a research into the development of a Dynamic and Adaptive Template Matching (DATM) algorithm, which is essentially an enhancement of the Dynamic Template Matching (DTM) algorithm. The Dynamic Template Matching approach is defined as a process for determining the reference image dynamically. The resulting algorithm achieved an improvement in accuracy of approximately 5%, when compared against the DTM approach. (Widyawan & Muhammad, 2012)

The functional difference between DTM and DATM algorithms is that DATM is able to constantly adapt to scene changes, such as sudden illumination. In order to do this, the reference frame is constructed by calculating average RGB values for each pixel. Subsequent frames would be compared against the reference frame, which would return a percentage of changes as a result. It should be noted that the reference frame would not be updated in areas where a moving object is being detected. This is required to prevent the object from slowly fading into the background.

The testing metric used by Widyawan & Muhammad was True Positives (TP) and False Positives (FP), where TP is the condition where the object appeared in the view and was detected successfully. The FP on the other hand was when a ghost object was detected.

The algorithm was tested using an IP camera, however, the camera and computer specifications, and CPU load were not disclosed. That said, frame size used during test was 256x192 pixels at 1 frame per second, therefore, the CPU load would've been reasonably low, even on a previous generation machine, such as Pentium 4 1.6 GHz. Nevertheless, despite of the low resolution, the algorithm was able to achieve 95.5% detection rate, which showed that large frames and fast frame rates might not be required for the approach to be effective.

1.1.4 Environmentally Robust Motion Detection for video surveillance

Hyenkyun, et al. observed that CCTV monitoring operators miss between 45-95% of actual intrusions when monitoring numerous cameras simultaneously. Therefore, the authors proposed a motion detection algorithm which would help operators notice motion without having to look for it. This algorithm is able to automatically adjust its parameters based on the environment, in order to suppress false alarms, by ignoring the noise caused by bad weather and sudden illumination. The primary objective of the proposed approach was to detect moving objects, but the exact shape and border of the object was not of the main concern. (Hyenkyun, et al., 2010)

The authors claim that the proposed approach is suitable for devices with constrained resources, as it doesn't require fast CPU, nor a large RAM. However, although paper's benchmark showed that the algorithm was able to process video sequences at 125fps, the frame size was 160x120 pixels. The host machine used to perform the test was a Pentium 4 3.0GHz PC. This casts doubts on authors' claims in relation to suitability for resource-constrained devices, as an average IP camera has a CPU of approximately 433-533MHz and resolution of at least 640x480 pixels(Business Wire, 2010).

The authors also claim that statistical background segmentation approaches are not always suitable because they require re-tuning between day and night environments, which is why the variational energy approach was chosen. This method is a complex equation often used in quantum mechanics.

"*Detection of moving objects is performed by thresholding the difference between two consecutive images. These methods work very well in day time since there is high intensity contrast among objects, while, at night, false alarms and detection failures are generated by low contrast and relatively high noise induced by poor lighting conditions. To overcome these problems, various statistical methods [9]–[14] are applied, but it is still hard to adapt abrupt changes of illumination without manual changes of their parameters. It is often necessary to adjust parameters of the model for satisfactory performance when environmental conditions are changed such as illumination due to light.*"

(Hyenkyun, et al., 2010)

"*To overcome such difficulties, we introduce a variational energy model with low dependency on parameters and robust to environmental changes and variation in signal-to-noise ratio*."

(Hyenkyun, et al., 2010)

The algorithm proved to be weak in scenes with swaying trees or fountains, in other words, scenes that have constant moving objects without a consistent shape. As a result, statistical background estimation approach was required, which defeated the purpose of the complex variational energy algorithm.

"*For more reliable motion detection algorithm, we need to adjust the background estimation following statistical changes in image sequences.*"

(Hyenkyun, et al., 2010)

That said, object detection masks could be set to ignore motion in certain areas using approach similar to that used by Zulaikha, et al.

In conclusion, although the algorithm proposed by Hyenkyun, et al. might look like a feasible option for accurate motion detection, more simplistic approaches are still required, which can potentially provide good performance and accuracy without the need for complex and potentially heavy algorithms.

1.1.5 The OBSERVER: An Intelligent and Automated Video Surveillance System

Duarte, et al. conducted a study for the development of a new approach for detecting humans' abnormal behaviours based on adaptive background subtraction algorithm. The research showed that criminal activity can indeed be predicted from real-time CCTV footage. The algorithm consisted of the following steps: detect moving object, detect shadows, highlights and ghosts, remove shadows, highlights and ghosts from the motion mask and merge them back into the background. (Duarte, et al., 2006)

Before any analysis can begin, the motion object should be detected and segmented. The segmentation logic should be able to accurately determine object's border and therefore shadows, highlights and ghosts should be removed before determining the border. To do this, motion mask is generated using the difference between the background and current frames, which then is passed to the algorithms for removal of shadows and highlights. The shadow and highlight algorithms also generate a bit mask, but unlike the background subtraction, this happens in HUE colour space. The resulting masks are then subtracted from the initial motion mask. At this stage, noise and ghost objects might still be present in the new mask, therefore the noise is removed first, by applying a thresholding filter, which removes small sparse objects.

In this paper, the ghost object was defined as:

"false positives originated by displacements of background objects"

(Duarte, et al., 2006)

In order to detect ghosts, a border of an object has to be determined. For this each pixel relative to the motion map is compared with its four neighbouring pixels and if their similarity is higher than a predefined threshold, it is considered to be part of the edge. Following this, the edge of the object is compared against the background and if the displacement is higher than 10%, the object is considered as ghost.

The following image shows the different stages of motion detection process.

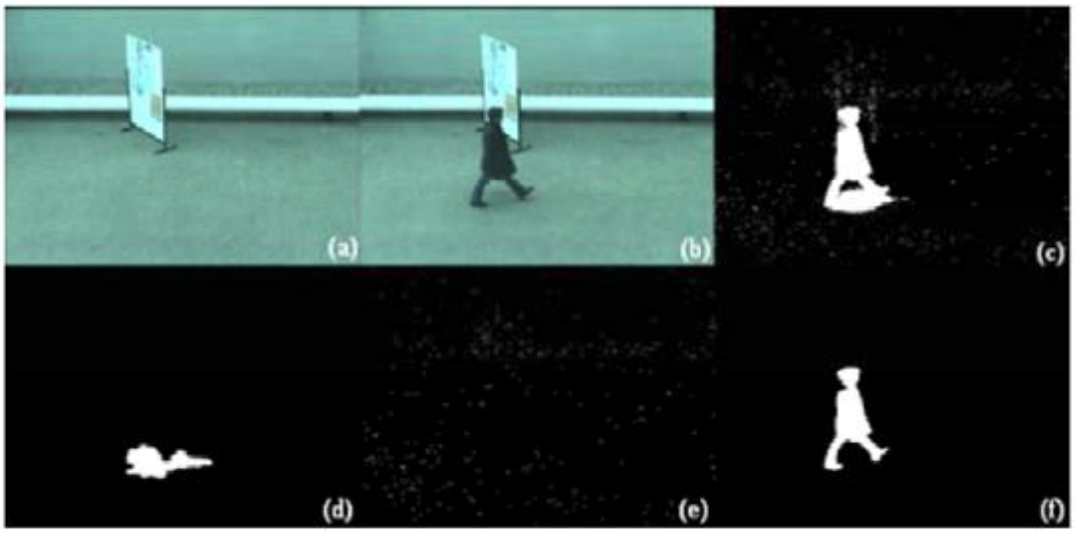


Figure 3: The OBSERVER motion segmentation (Duarte, et al., 2006)

(a) Background Image; (b) Current Image; (c) Primary Motion Mask; (d) Shadow Mask; (e) Highlight Mask; (f) Filtered Motion Mask

Following the detection of the object, it is passed onto the tracking logic, where a tracker is attached to the object, which is required by the behaviour detection and prediction. The behaviour detection and prediction logic utilizes a database where various object attributes are stored. These are classified by operators and then used to compare newly detected objects. If the new object is found in the database, then the probability of the event becoming abnormal is calculated based on attributes such as object speed, size, perimeter, area. If the probability exceeds the predefined threshold, an alarm event is raised.

The test results of this approach showed that it could be utilized in real-time surveillance, however the test parameters were not given. In addition, the algorithm was not tested with real CCTV footage, and more importantly, footage obtained from IR cameras, as many background segmentation algorithms would fail to perform well under such conditions (Hyenkyun, et al., 2010).

1.1.6 ViBe: A universal background subtraction algorithm for video sequences

Olivier & Marc conducted a research into the development of a robust and universal algorithm for the foreground subtraction, which makes no assumptions related to the scenes or its attributes, instead it focuses more on what a potential object might be, which achieves a high degree of accuracy in object detection. (Olivier & Marc, 2011)

"ViBe makes no assumption regarding the video stream framerate or color space, nor regarding the scene content, the background itself, or its variability over time."

(Olivier & Marc, 2011)

The authors state that most of the background subtraction approaches are based on pixel *A* vs. pixel *B* comparison given the threshold *T*, however, absolute comparison of the two pixels with a particular threshold is not always accurate, especially over time. Therefore the threshold *T* should be dynamic and should be relevant to the statistical pixel change rate.

The algorithm builds a background model where each pixel has a set of history pixels from previous frames. Every incoming pixel is compared against the pixels of the set using the Euclidean distance. Where the number of history pixels that are identified to be within a radius *R,* is less than a constant threshold, the incoming pixel is considered to be background. The following figure shows this logic in Euclidean colour space, where *v(x)* represents an incoming pixels and v_n a sample. The radius *R* represents a threshold.

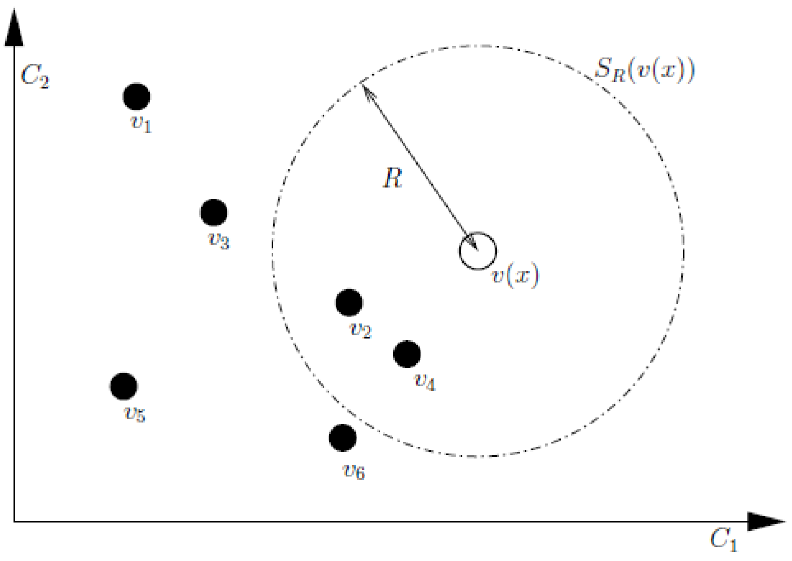


Figure 4: ViBe's background detection illustration (Olivier & Marc, 2011)

The background model is built from the first frame of the sequence, allowing the detection to begin from the second frame, however, because sudden light changes in the scene might significantly alter the chromaticity component of the frame, in such cases the background model would be discarded completely and the latest frame would then be used to rebuild the background model. It should be noted that the background model is not always discarded due to sudden illumination, but only early in the cycle. The background model initialization is based on assumption made by (Pierre-Marc, et al., 2007), where the neighbouring pixels hold very similar temporal distribution.

It was observed by the authors that in case the background model is initialized from a frame containing a moving object, the approach would introduce a ghost object, which would eventually be dissolved due to constant model updates.

The algorithm uses a "*conservative update*" approach taking into account the temporal spacial distribution assumption described by Pierre-Marc, et al. (Pierre-Marc, et al., 2007), in order to update the background model. The conservative update means that a pixel belonging to the foreground should never be used to update the background. However, according to the authors, this approach could lead to ghost objects. To remedy this, temporal spacial distribution of neighbouring pixels is used to update background pixels covered by the foreground object. According to the authors, most of the background update approaches simply remove the oldest frame, which is not correct as just because a particular pixel is old, doesn't mean it changed. Therefore, the pixels inside the pixel model are replaced randomly, although this approach might still lead to wrong pixels being replaced. It should be noted that neighbouring pixel models are also randomly updated.

The tests were conducted on two different types of video sequences containing complex backgrounds, such as moving trees and bushes. Then the results were compared against 2 simple and 5 advanced algorithms. The comparison was made using Percentages of Correct Classification framework, which determines the percentage of correctly identified pixels. The results clearly showed that the rate of correct detections increased from approximately 88% to 99% as the number of samples increased from 0 to 20, after that the Percentage of Correct Classification (PCC) rate remain roughly the same. Given the fact that normal video frame rate is about 24 fps (Chaney, n.d.), it would take the algorithm close to 1 second to build reliable background model, and as a result, the additional complexity introduced with building the model from the first frame is largely redundant.

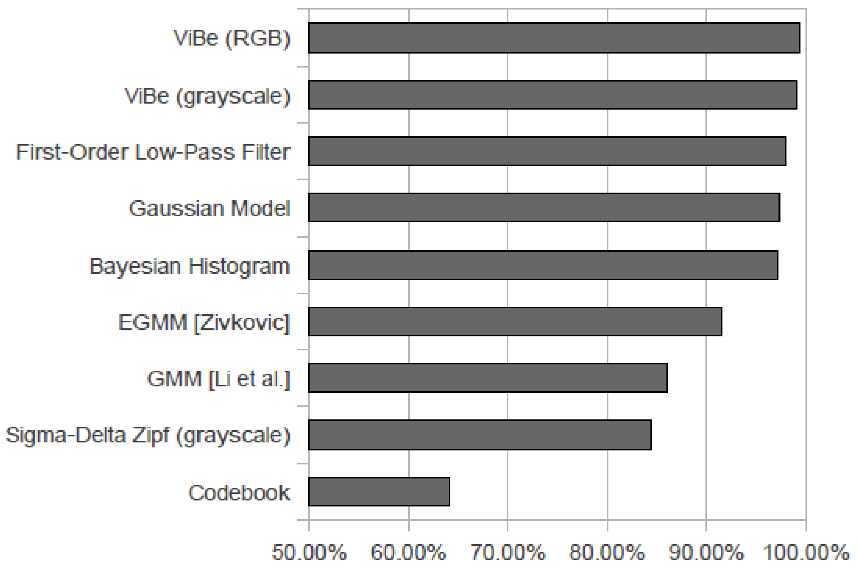


Figure 5: ViBe PCC rates achieved by the authors (Olivier & Marc, 2011)

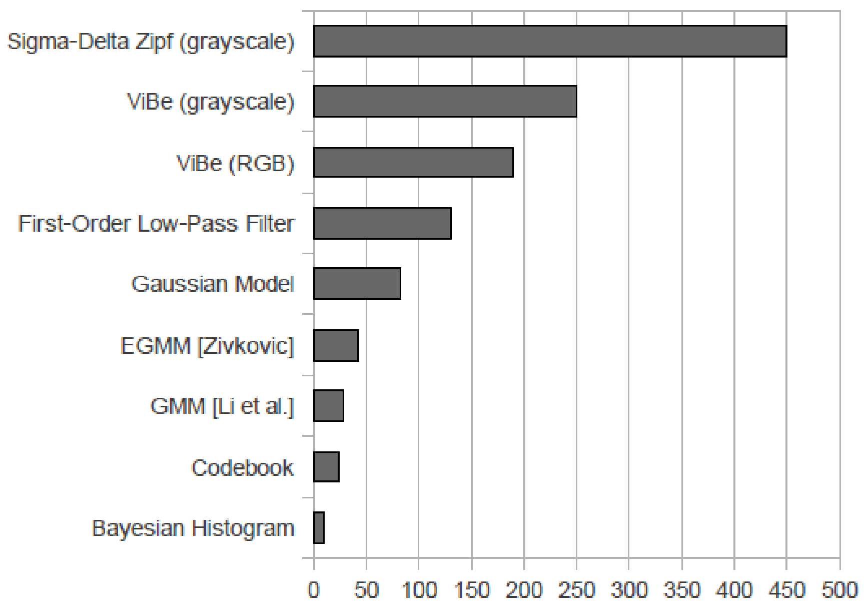


Figure 6: ViBe FPS rates achieved by the authors (Olivier & Marc, 2011)

The tests conducted against other algorithms showed that ViBe was able to outperform all of them and was able to ignore moving trees. That said, the maximum frame rate achieved by ViBe is 250 FPS, which puts it on a second place after sigma-delta algorithm (Antoine & Julien, 2007). The host machine used in a test was a Core i7 2.67GHz with 6GB of RAM; the frame size was 640x480 pixels. The MODE algorithm (Birmohan, et al., 2014) utilized very similar approach to that of ViBe, in fact it borrowed some of the concepts but the test was conducted on a significantly weaker machine, where the test returned only 7fps. Therefore high frame rate achieved in ViBe's test could be the result of a very powerful machine rather than well optimized algorithm.

1.1.7 Background Subtraction Based on a Robust Consensus Method

Hanzi & David conducted a research into the development of the SAmple CONsensus (SECON) algorithm, which is an adaptive background subtraction algorithm that can be utilized on both, static and dynamic scenes. The algorithm utilizes the colour normalization technique in order to detect/suppress shadows and handle illumination changes. In this approach, the chromaticity and luminance of the RGB colour space are separated and luminance is ignored, however, (Birmohan, et al., 2014) critique this approach, stating that it causes too much detail to be ignored and that it performs poorly in low-intensity frames, causing false detections. (Hanzi & David, 2006)

The video sequence processing flow is the following:

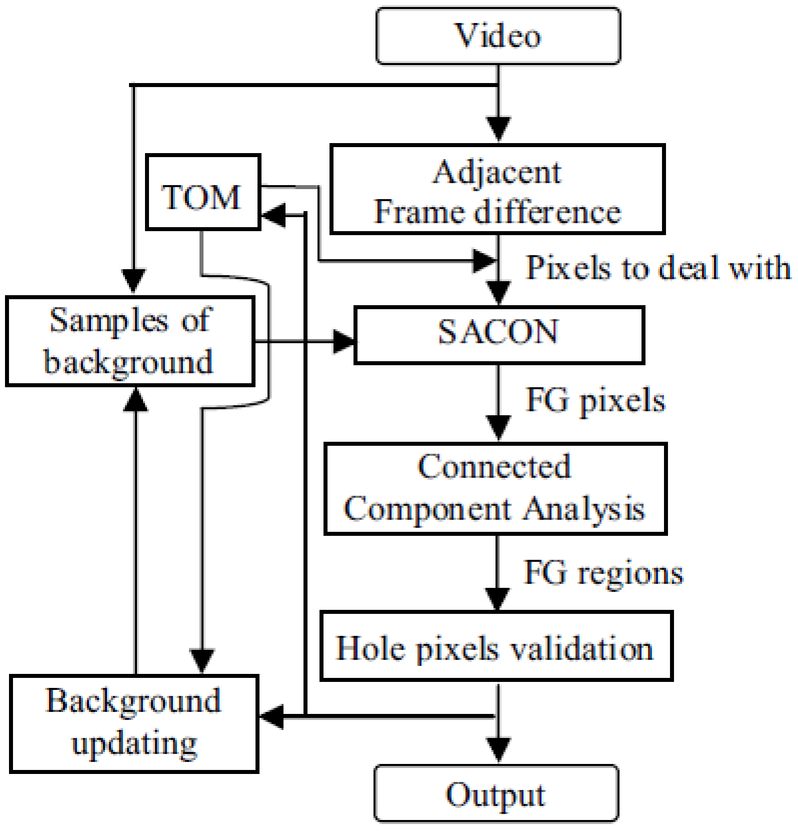


Figure 7: SECON Flow Diagram (Hanzi & David, 2006)

In the flow chart above, TOM stands for Time Out Map, which is a map of pixels where the value for each pixel is increased every time the pixel is marked as a foreground. The SECON box represents the algorithm that classifies pixels into background or foreground based on background samples. The result of this algorithm is a map of Foreground Pixels, which might contain holes in cases where the moving object contains similar colours to the background. At this stage, the pixels in holes (if any) have to be validated and the background updated with pixels that turned out to be background. The pixels that turned out to be representing the foreground would update TOM to keep it up-to-date for the sequences to follow. The hole pixels validation logic compares the pixels of a hole against the pixels in the background frame that are in the same location. If the difference between the colours is greater than a constant threshold, then the pixels are considered to be background pixels.

The test was conducted using False Positive (FP), False Negative (FN) and Total Error (TE) metrics. The Wallflower benchmark (Kentaro, et al., 1999) conditions were utilized and tested indoors, although some of the tests involved switching lights on and off. It should be noted that the algorithm wasn't tested with IR lighting and in outdoors environment.

The results showed the Total Error rate for different configurations and different tests, but, although the paper states that it outperformed other state-of-the-art algorithms, not enough data was provided to support this claim. The frame rate achieved was between 6 and 10 FPS for 160x120 pixels frame on Pentium M 1.6 GHz machine. This showed that the approach would not be suitable for real-time surveillance, which requires the frame rate to be up to 24fps, unless of course even smaller resolution would be sufficient for accurate detection. However, such condition was not tested and even if it was, it is unlikely that better results would've been achieved as the frame size of 160x120 pixels is close to the minimum resolution required to identify moving objects, especially on wider angle views.

1.1.8 Motion Detection for Video Surveillance

Birmohan, et al. developed a novel approach to background subtraction which is independent from bootstrapping, illumination changes, noise and dynamic variations in scenes. In this paper bootstrapping is defined as:

"The process of initializing the background model in which foreground objects are also available"

(Birmohan, et al., 2014)

This paper also presents a new colour model for the detection of illuminations and shadows, and a new object tracking technique. Unlike SECON approach (Hanzi & David, 2006), where the RGB luminance component is completely ignored, the authors suggested to reduce its importance in order to reduce false alarms, while suppressing shadows.

The algorithm borrowed background initialization and updating techniques from SECON (Hanzi & David, 2006) and ViBe (Olivier & Marc, 2011) algorithms, therefore, although the algorithm is based on statistical pixel data, it is able to initialize its model from the first frame, where the incoming pixels would be compared against neighbouring pixels of a background frame. The authors admit that initially this approach might lead to false detections, until a larger set of statistical data is available, which in case of ViBe is about 20 frames, and therefore translates to approximately 1 second for initialization. It should be noted that the background frame cache is constantly updated by completely replacing some of the frames (starting from second most recent).

In order to detect whether the pixel belongs to the foreground or the background, Euclidean distance is utilized. The calculated distance between a particular pixel in background and foreground represents a rate of colour change between the pixels, where the shorter distance means that the two pixels are similar. The bit mask is then constructed where ones represent the foreground and zeros the background.

The model update was performed randomly, where each pixel to be updated was selected by summing the pixel indexes and frame number and then applying modulus constant. If the resulting value was zero, then the pixel, along with its neighbour pixels were updated. The random pixel selection algorithm used was very similar to the one used in ViBe (Olivier & Marc, 2011).

The testing of the algorithm was performed using two data sets of approximately 40 video sequences in total, covering a wide range of scenarios, such as sudden illumination, dynamic background and shadows, among others. The test results showed that the MODE algorithm performed better than any other tested algorithms, however, it was only able to provide a frame rate of 7fps on a 1.4GHz CPU with 1GB of RAM. The frame sizes were not included in the data provided and the data sets used could not be found.

1.1.9 A new motion detection algorithm based on Σ-Δ background estimation

Antoine & Julien conducted a research into the development of a surveillance system that would automatically detect moving objects without having to frequently configure or adjust its parameters (Antoine & Julien, 2007). The main requirements of the system are that the system should be able to build the model from initial frames, handle sudden illumination and to be lightweight in order to be used in devices with limited resources, while processing feeds in real-time. In order to satisfy all of these requirements, the authors eliminated most of the popular approaches, such as first vs. current frame comparison, temporal average of the background, methods using histogram analysis, linear prediction, kernel density estimation and principal component analysis.

The core of the proposed algorithm (pronounced as sigma-delta) is to increment or decrement the pixel representation of the model when building or updating the model. This results in a statistical model, which is then subtracted from the incoming frames. The values used to increment or decrement the pixel model are Gaussian distributions. However, (Olivier & Marc, 2011) and (Srivastava, et al., 2003) argue that Gaussian based statistical model is not always relevant to the natural images. In addition, (Olivier & Marc, 2011) stated that Gaussian based models cannot handle high speed moving objects, which wasn't tested by the authors of sigma-delta.

The time variance of the pixel was calculated in order to classify pixels as foreground only if they remain in the view for a prolonged period of time. As a result, this algorithm should be able to cope with rain drops on IR feeds, as the drop would only remain in the view for a couple of frames only. The tests showed that this algorithm is able to accurately detect moving objects surrounded by constantly moving background, such as grass.

Following the detection of the foreground, the noise and ghost objects are removed using spatiotemporal regularization and binary morphology, which were claimed to be significantly complex but at the same time efficient.

The performance of the algorithm was tested on an artificial retina device of 200x200 8-bit pixels with a 25MHz processor, which achieved 2.25ms per frame, where only 0.75ms was spent on segmentation and the rest on image acquisition. This strongly suggests that the approach is well capable to process high resolution images on embedded devices, but the accuracy of the approach was not tested. That said, (Olivier & Marc, 2011) had tested their algorithm against this exact implementation of the algorithm which left it on approximately third place in terms of accuracy and on the first place (by a long shot) in terms of speed of processing measured in FPS.

1.1.10 The Pixel-Based Adaptive Segmenter (PBAS)

Hofmann, et al. conducted a research into the development of a non-parametric motion detection algorithm where some of the parameters for each pixel are adjusted dynamically at runtime (Hofmann, et al., 2012). This algorithm is very much based on ViBe algorithm, so this approach can be seen as a potential improvement of ViBe.

Just like in ViBe (Olivier & Marc, 2011) and SECON (Hanzi & David, 2006), this algorithm represents a background pixel as a "history of *N*", where *N* represents an incoming pixel. It uses a random update rule to update the background model, similar to that in ViBe. The main difference between ViBe and PBAS is in the randomness approach taken by PBAS. Vibe uses a fixed randomness and threshold parameters for all pixels, whereas PBAS adapts these parameters dynamically at runtime for each pixel separately. Therefore, some pixels are not guaranteed to be updated at all, which could present a problem for mistakenly identified pixels.

The algorithm detects whether the pixel is foreground or background by comparing it to the set of history pixels at the same coordinate. This comparison is made using the Euclidean distance, just like in ViBe, but the distance threshold is determined dynamically for every pixel *N* separately. The minimum number of pixels that must be closer than the distance threshold for classifying the incoming pixel as foreground or background, remained a constant value.

Another dynamic parameter that PBAS has introduced, is the probability rate for updating the background model. Where the probability rate is higher than a dynamic threshold, the incoming pixel is used to update the history set of pixels, unlike the approach taken by ViBe, where this value is a non-parametric constant. The size of the history set does not change, therefore some old background pixels need to be removed to make room for the incoming pixel. The approach taken by PBAS is to choose randomly the pixel in the history set to be replaced with the incoming pixel. However, this approach could result in the removal of pixels that more closely represent the background. Besides, by the time the background model needs to be updated, Euclidean distance would've been already calculated, therefore the resulting values could've been reused without further calculations to select the furthest (e.g. the least similar) pixel.

Another one contrasting feature of the algorithm is that it would "eat-up" the foreground objects over time, in order words, the foreground objects would be slowly merged into the background from outside. The authors claims that this logic allows incorrectly detected foreground objects to be moved into the background quickly, at the price of slowly "eaten-up" real foreground objects. It is unknown how fast this would happen, therefore it is unknown if this is likely to cause problems with slow moving objects.

The algorithm is able to learn which parts of the frame are foreground and which are background, therefore, more "active" parts of the frame result in background to be updated more frequently, which could lead to performance improvements as less background pixels need to be updated, especially given the fact that most of the time, most of the pixels represent a background.

The performance tests were carried out against the "*Change Detection Challenge*" (Pierre-Marc, et al., 2010) data set, which contained a variety of different scenes. The metric used to record the accuracy of the approach was the Percentage of Bad Classification.

Algorithm	**Percentage of Bad Classification**
PBAS	1.7693
ViBe	3.1178

Figure 8: PCC results comparison between ViBe and PBAS algorithms achieved by Hofmann, et al

The test results showed that the algorithm outperformed all of the state-of-the-art algorithms tested, with accuracy improvements of up to 51% percent. However, the processing frame rate was not specified.

1.2 Research Question

The primary use of the foreground segmentation in CCTV monitoring centres is to detect intruders on numerous simultaneous feeds without having to look for them. This would not only decrease the chance of overlooking an intrusion, but could also decrease the amount of staff required to perform the monitoring.

The task of segmenting the feeds can generally be located in two places, on a remote site or inside the monitoring station. The remote CCTV devices are generally low-powered and resource constrained, which typically deal with a smaller number of feeds (between 1 and 16). The monitoring station on the other hand, can be a powerful computer, however, it would need to be able to process hundreds of feeds simultaneously as it might be connected to numerous sites.

Given these constraints and that many monitoring customers opt-in for the IR types of cameras, this research attempted to answer the following question:

Can latest background subtraction algorithms be utilized on embedded devices, in order to successfully detect moving objects on IR video sequences, while ignoring environmental noise such as cobwebs or rain?

1.3 Conclusion

This chapter provided a brief overview of the different motion detection and background segmentation approaches, most of which could be suited for the surveillance purposes, however, none of these approaches have been tested against outdoor IR feeds. These kinds of feeds tend to cause a lot of false alarms in monitoring centres due to bad weather or insects living on the cameras.

The performance of the reviewed algorithms varied between acceptable to unacceptable for real-time detection on embedded devices, bearing in mind that most of the feeds were tested against extremely low resolution of about 0.2MP, using which on wide angle feeds can have an adverse effect.

Therefore, this paper looks into the development of a tool which would allow testing various motion detection algorithms against problematic and generic night feeds, on a range of devices. In addition, the tool attempts to suppress noise objects through purpose built post-detection filters.

The remainder of the document is structured as follows:

1.3.1 Chapter 2

This chapter describes the problems and benefits of motion detection, libraries available for video processing and state-of-the-art algorithms that could potentially be suitable for the real-time monitoring on resource-constrained devices.

1.3.2 Chapter 3

This chapter describes in detail the testing harness, post-detection filters, target devices and result analysis tools, developed in order to conduct tests and inspect the results of various combinations of segmentation algorithms, filters and devices.

1.3.3 Chapter 4

This chapter describes the methodology, data sets and conditions used to perform the testing. Following this, the test results are analyzed and summarized.

1.3.4 Chapter 5

This chapter describes the results that have been achieved, the strengths and weaknesses of the approach chosen and draws a conclusion.

2 Theory and Background

Operators miss up to 95% of true alarms/incidents when monitoring multiple cameras simultaneously (Hyenkyun, et al., 2010). Most situations require that the incidents are dealt with in a timely manner (Kevala & Sasse, 2006), therefore any potential intrusion has to be detected in real-time.

Companies such as Group4 Security and Netwatch have a multitude of cameras that have to be constantly monitored and every detection has to be investigated. In order to stay competitive, monitoring centres often charge as little as €6.00 per 24 hours of monitoring of up to 16 cameras. Therefore, for the business to make a profit, the cameras to staff ratio needs to be high. This presents a problem when bad weather or cobwebs constantly set the alarm off, increasing the chance of missing an intrusion.

The problem often can be solved by installing motion detection sensors, such as Passive Infra-Red or Point-to-point Infra-Red sensors, but these sensors are expensive and for various reasons can be extremely unreliable. The motion detection, on the other hand, is the perfect solution, but it requires good lighting. However, due to high electricity costs, the Infra-Red lighting is often chosen. The IR light results in a black and white picture, making it hard for motion detection algorithms to determine whether the pixel is background or foreground, resulting in many false detections.

The development of a new motion detection algorithm or filter that could detect and/or suppress "noise" objects is crucial for today’s businesses, as it would allow increasing the cameras-per-person ratio, while decreasing the risk of missing an important event.

2.1 Definition of "real-time"

The real-time frame rate value varies as it is bound to a context where it's used, for example in movies at least 24FPS would typically be required, however, in CCTV monitoring environments, anything from approximately 12 FPS can be deemed acceptable, because the video feed would still be seen as "live", as opposed to "slide show" effect resulted from much lower FPS. Furthermore, the vast majority of CCTV systems operate with frame rates between 6 and 10 FPS (Honovich, 2011), although the article recommends frame rates of at least 12 FPS to remove erratic motion. Therefore, for the remainder of this work the "*real-time*" frame rate would refer to at least 12 FPS.

2.2 Video Processing Devices

The foreground/background segmentation is a highly versatile process, which is used for many different purposes and devices, such as phones, security equipment, still shot cameras and others. Many of such devices have high-end processors and often dedicated graphics chips. The power of these devices tends to drive their price, which often is too high for CCTV purposes as normally at least 4 cameras are needed to secure the perimeter. As a result, lower end processors are generally used for CCTV video recorders and cameras. Therefore, for the purposes of this research, development boards Raspberry Pi 1 Model B (Raspberry PI Foundation, n.d.), Orange PI (Xunlong Software CO.,Limited, 2015) and Banana PI (Banana Pi, 2015) were used. This development board is equipped with a 700MHz CPU and 512MB of RAM, among other peripherals such as USB host and an Ethernet adapter.

2.3 Low-Resolution Frame Processing

Many of the existing algorithms are CPU-intense (such as those based on histograms or Gaussian models). This forces the processing to be based on much smaller frames, approximately 256x192 pixels. Although multiple research papers suggest that such small frame size is sufficient for accurate detection [(Widyawan & Muhammad, 2012), (Hyenkyun, et al., 2010), (Hanzi & David, 2006)], it is arguable whether the detection rate would be acceptable on wide angle scenes, as the scene features would be much smaller. As such, it is possible that a single drop of rain could be larger than a person walking in front of the camera, making it practically impossible to tune down the sensitivity of the motion detection algorithm to suppress false detections.

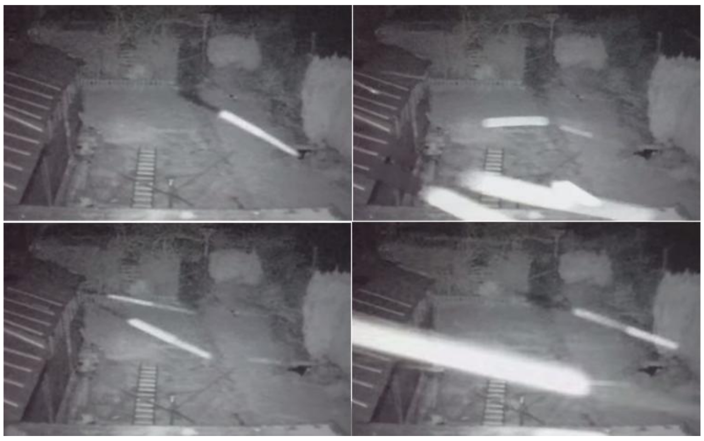


Figure 9: Wide angle frames with snow too close to the lens(Anon., 2010)

Another feature that is widely used, is the boundary mask. When the user marks certain areas of the frame as to be ignored, those pixels usually would not be iterated over, therefore speeding up the process of foreground segmentation. It should be noted though, that the primary use-case of this feature is to ignore certain parts of the frame, the performance benefits are collateral.

2.4 Computer Vision Framework

The video processing and analysis is generally very complex, even in simple application that for example, only retrieve frames and iterate over pixels. In order to avoid "*re-inventing the wheel*", the OpenCV library (Open Source Computer Vision Library, 2015) has been used to aid the development of the prototype and testing harness. Perhaps one of the most important features of this library, is the automatic memory management, which guards against memory leaks (to some extent). Memory management aside, the OpenCV library also provides a number of motion detection algorithms, effective pixel manipulation helpers/utilities and frame analysis tools.

The BSGLibrary (Andrews, 2015) was also utilized as it implements a number of motion detection algorithms, which can segment video feeds, but it is not able to compare the foreground against the ground truth or calculate results. Furthermore, this utility is compiled for windows environment and therefore would not run on a Linux development board.

2.5 Types of Motion

When monitoring a large number of cameras in a setting such as a CCTV monitoring centre, it is crucially important that the moving objects are accurately detected, as otherwise false-positives would trigger alarms, distracting the operators and causing them to become less vigilant. Many algorithms are well able to perform this task exceptionally well in cases where the scene is well illuminated by either street lights or day light. However, when Infra-Red light is used (e.g. during the night), moving objects such as dust, rain, snow, cobwebs, flies, etc. reflect much more (infrared) light than the rest of the background, causing the noise objects to be much more visible than they would be during the day, which triggers false alarms. From the perspective of the motion detection algorithms, these alarms cannot be classified as false detections, because they are actual moving objects, but the monitoring centre operator does not need to know about them.

The most typical types of motion are the following:

- Actual moving objects, such as a human or an animal
- Background movement, such as that caused by waiving trees and running water
- Small flying objects, such as rain or snow
- Cobwebs and spiders in front of the camera

As can be seen from the following images, the objects of interest tend to have less brightness, the cobweb are often transparent and the rain drops typically don't remain in the view for a prolonged period of time. Where rain could potentially be tolerated by advanced motion detection algorithms, such as those based on the temporal differences or statistical approaches (Manzanera & Richefeu, 2007), the cobwebs would still be detected, because they remain in the view much longer and often cover larger view area.

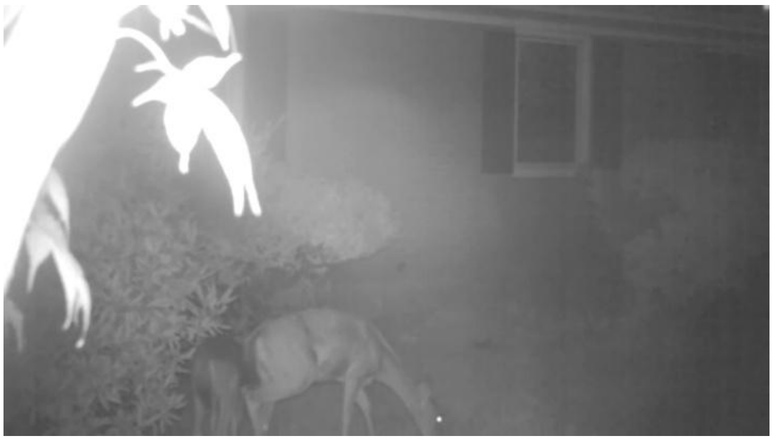


Figure 10: Contrast difference between a genuine moving and over-saturated object

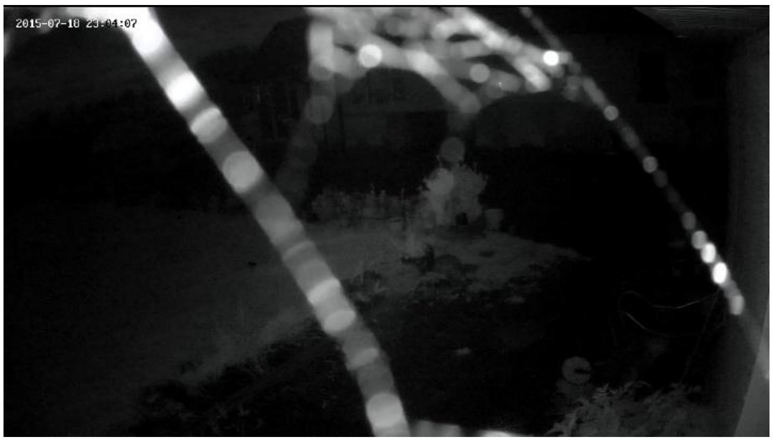


Figure 11: Semi-transparent cobweb over saturating the frame

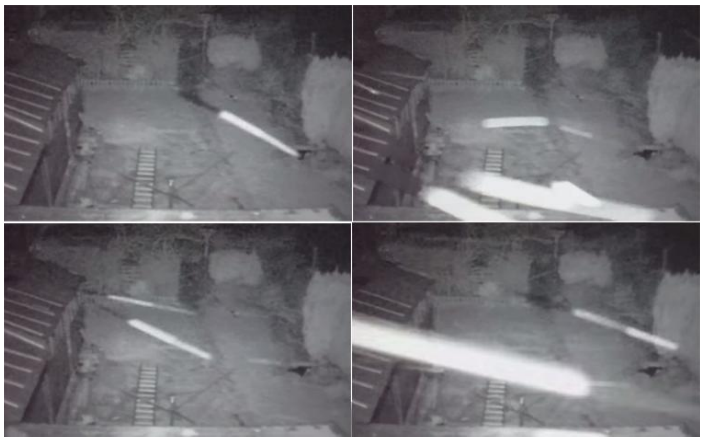


Figure 12: Wide angle frames with snow too close to the lens (Anon., 2010)

2.6 Background Subtraction

The background subtraction technique is perhaps the simplest technique available for detecting motion. It generally works by subtracting pixels in one frame from another. The resulting set of pixels is considered to be the motion object. This approach is very popular due to its simplicity and low processing power requirements, however, it performs poorly in the outdoors environment with low lighting. In addition, this algorithm can be fooled, if the object is moving too slow, especially with the higher frame rate, in which case the difference between frames would always be minuscule (Ching, et al., 2011) (Olivier & Marc, 2011). In order to remedy this, a number of algorithms were developed.

As already mentioned, the most popular approach is the simple background subtraction, where two grey-scaled frames (background and incoming) are subtracted from each other, then, sparse pixels are removed using the erosion filter and finally, the remaining pixels are counted, determining whether the alarm should be triggered. Then, the background frame is discarded and the incoming frame becomes the new background.

2.7 Adaptive Background Subtraction

This approach is slightly more complex but typically more powerful. Many simple algorithms just discard the old background frame, however, just because there is a more up to date frame, it doesn't mean that all of the pixels in the older frame no longer represent the background. As such, the main principle with this kind of approach is to adapt the estimated background to the actual background by gradually merging the incoming frames into the background. This prevents slowly moving objects from being undetected as the background is not completely discarded every time.

The adaptive background subtraction approach is often based on statistics, where every frame coordinate has a collection of pixels from previous frames. The incoming frame pixels at the same coordinates are then compared against these collections, and generally the incoming pixel would be considered as foreground if it deviates too much from the pixels in the collection.

The adaptive background subtraction algorithms used in this research were:

- PBAS (Hofmann, et al., 2012)
- Sigma-Delta (Antoine & Julien, 2007)
- ViBe(Olivier & Marc, 2011)

The Pixel-Based Adaptive Segmenter (PBAS) algorithm was chosen primarily because it is based on ViBe algorithm, but also because it showed a significant improvement in segmentation accuracy. The test results described in PBAS paper suggest that the approach is highly efficient and that it outperforms many of the state-of-the-art approaches, including ViBe, but testing hardware specifications and achieved FPS rate was not specified.

The ViBe algorithm was chosen as according to its authors, the algorithm is capable of achieving very high FPS and PCC rates. In addition, the performance of ViBe can be seen in action on authors' website in form of a video, which shows that the algorithm is able to cope very well with sudden illumination, temporary motion and fast-moving objects. (Anon., n.d.)

The Sigma-Delta algorithm was chosen because it outperformed ViBe in terms of CPU load by almost double, however, a lower PCC rate was achieved, with only 85%, whereas ViBe achieved close to 100% (Olivier & Marc, 2011). It can also adapt well to the constantly moving backgrounds and other noise objects.

2.8 Performance Evaluation

The selected algorithms were evaluated against the "*Change Detection Challenge*" database of videos (Pierre-Marc, et al., n.d.), which features a number of different scenes and scenarios, such as bad weather, night videos, dynamic background and many others. In addition to the database, a number of short clips captured by a local CCTV camera were used. These clips feature infrared-illuminated scenes, where cobwebs are moved by wind or spiders and/or falling drops of rain.

All of the algorithms were tested with and without custom filters developed as part of this research in order to determine their effectiveness. The ground truth was established manually and was used in accuracy measurement (i.e.

PCC). The frame rate achieved during processing was also measured and recorded.

The performance testing was carried out on a number of development boards, similar to the Raspberry PI. The tests were carried out with different frame sizes in order to determine if the frame size used for detection has an effect in accuracy. The Percentage of Correct Classification metric was used to measure algorithm’s accuracy and the FPS rate to determine if the algorithm would be suitable for real-time processing.

2.9 Conclusion

Every approach has its benefits, and many are able to accurately separate foreground from the background, but because the “noise” motion objects are actual objects, it is very hard to suppress them.

Out of all of the reviewed papers related to motion detection, very little was aimed at detecting motion on IR frames and no test video database was found that would contain less perfect video clips, such as that where time to time a spider crawls in front of the camera.

After observing closely the characteristics of the motion to be suppressed, a couple of key properties were detected. As it can be seen in *Figure 11,* the noise shape has a number of transparent sections. The operator can assume that an intruder would not be transparent, therefore, the background features identified on the provisional foreground objects can be removed from the foreground, leaving only solid colour object.

Another problem that can be observed on the frame from *Figure 10: Contrast difference between a genuine moving and over-saturated object*, is that objects located too close to the camera/lens reflect too much infra-red light, over-saturating the frame. These objects often become so over-saturated, that they appear in pure white colour. Therefore, the operator could make an assumption that parts of the foreground that have a value higher than a particular threshold, should be ignored. This is because the objects of interest would normally reflect less light, and therefore would appear as grey.

The background segmentation on its own is quite error prone when it comes to black and white feeds (Ching, et al., 2011) (Tetsuya, et al., 2010), therefore additional video filters are required to filter out what the surveillance operator does not need to know about. These filters can be based on object speed, colour, direction, starting point and heading direction, movement patterns and others.

3 Segmentation Tester

Motion detection is becoming a more and more popular choice for security companies specializing in CCTV monitoring, because it allows detecting intruders without having to install special hardware, such as PIR sensors, which is expensive, require additional maintenance and can be tampered with without the operator knowing about it (e.g. sensor could be pushed to point to another direction). Motion detection on the other is much harder to tamper with and doesn't require additional hardware. However, it can be a nuisance for an operator in cases where the monitored area is lit up by Infra-Red light. The main problem with IR is that most cameras have the IR LEDs located around the lens, and therefore, objects that are located very closely to the lens would reflect too much IR light, over-saturating the image. Furthermore, the IR light attracts spiders, which then weave the web around the lens resulting not only in over-saturated frames, but also constant false detections due to web movements.

In order to address this issue, a utility was developed which allows testing a number of latest motion detection algorithms against problematic feeds. These algorithms were tested on normal computers and embedded devices, such as Raspberry PI. The reason for performing tests on Raspberry PI is because IP cameras are becoming more and more popular as they are able to deliver high resolution video for the price of analogue cameras and they utilize hardware with similar architecture to Raspberry Pi. As a result, the motion detection process is shifting from Digital Video Recorder devices to IP cameras, which tend to run on cut-down versions of Linux and on processor architectures such as ARM.

The test results of this utility were used to identify motion detection algorithms which could perform at real-time frame rate on embedded devices, while accurately separating the foreground from the background on IR video feeds. This also included classifying moving cobwebs (and other noise objects) appropriately and removing them from the foreground.

In order for the testing harness to determine whether the algorithm is suitable for the task, the segmentation results must be collected from feeds containing scenarios where:

- an intruder appears in the view, while no noise objects are visible/moving
- an intruder appears in the view, while noise objects are visible/moving
- there is no intruder in the view but the noise objects are visible/moving

The above condition was also tested on full-size frames in order to determine whether more frame detail can produce more accurate results.

3.1 Testing Harness

The testing harness encapsulates multiple motion detection algorithms and post-detection filters. Most of the implementations for the algorithms used were taken from a BGSLibrary project, with the exception of ViBe algorithm, which was acquired directly from the authors (Olivier & Marc, 2011).

The harness was designed to run only in command line mode because the graphic user interface (GUI) adds more load on the embedded device as Linux X Server and the Window Manager would also need to be running. However, for debugging purposes it is possible to enable some GUI so that incoming and processed frames are shown, although this would only work on operating systems with Window Managers running (e.g. not SSH or command line).

The harness expects a number of command line parameters, which are used for configuring the test run. These parameters allow specifying the source of the video (can be a path to a file, a URL to a video stream or device ID), test results destination path, ground truth source, the algorithm to be used for motion detection, optional post-detection filters, frame size and others (full list of parameters can be found in Appendix 4).

3.1.1 Frame Acquisition

The testing harness is developed to be able to read frames from a variety of sources, such as video files, sequence of image files or video streams such as RTSP or raw streams, captured from a camera directly connected to the computer running the test utility.

The OpenCV library is able to automatically detect and read all types of sources mentioned above, however, it is not able to automatically read sequences of images as it wouldn't know which file name represents which frame.

Therefore, since the file sequences provided by "*Change Detection Challenge* " use the file name format such as "in000001.jpg", this format was "hard-coded" in the testing harness logic. The logic for reading this kind of sequences would record last read frame number and increment that value by one, in order to construct the file name of a next frame to be read when requested next. The same logic is employed for ground truth frames.

Before the read frame is segmented, it is resized proportionally to the width specified as a command line argument. Then, the resized frame is converted to greyscale. Both of these operations are required to optimize performance and to test whether processing full-size frames can increase the detection accuracy. The only exception is the Sigma-Delta algorithm, which requires colour frames, therefore in this case the greyscale conversion would happen after segmentation. The greyscale frame processing was enforced for most algorithms because IR feeds are always black and white anyway.

3.1.2 Test Results

In order to be able to accurately analyze the results of the combination of settings for each test run, each frame would be saved onto the hard drive (or network drive) after the segmentation process. The file name would be composed of a frame number followed, by the type of frame (e.g. input, output). In order words, the file name of each of the files would be in the following format:

1234-motion.jpg

1234-input.jpg

1234-ground.jpg

These file names would then be written to the CSV file used as output. All this data would be inspected at a later stage by the test result analyzer program, which is described further down.

It should be noted that "input" and "ground" types of frames would be copied from the source directory to destination even if they are stored in the source directory as a sequence of files, as opposed to a video file. This is required in order to be able to test the algorithms using footage saved in video file formats such as AVI, or live video feeds (although in this case no ground truth would be available).

The frame processing rates, measured in frames per second (FPS) were written to the standard output stream every second and as an average at the end of processing.

The testing harness would also measure and record the percentage of correct classification (PCC) for each of the processed frames in the CSV file.

The detailed CSV file format can be found in the Appendix 6.

3.1.3 Performance Testing

In order to achieve accurate results, it was decided to only measure the performance of frame segmentation and filtering procedures, while excluding the time taken by reads, writes and PCC calculation. This is required because otherwise some streams could be encoded, adding more load on the CPU when decoding such frames. In addition, network problems could slow down data transfer, resulting in slower reads or writes. Therefore, measuring only the time taken by the segmentation processes would result in most accurate figures and the target devices would not need to be run on a separate (low traffic) networks.

3.2 Target Devices

The test harness is developed to be run under Windows, Linux and Linux running on ARM architecture. However, although compiling for Windows and Linux is a reasonably quick procedure, compiling for Raspberry PI (and other development boards) required the use of "distcc" (Pool, n.d.) tool because the development boards perform much slower when it comes to compilation, unlike standard computers. Therefore, to utilize the power of a standard CPU, the "distcc" tool sends the source code files to a machine with a faster CPU, where they're compiled and the resulted object file is returned back to the development board.

The linking stage is then completed on the development board, which is a relatively quick process. More details on compilation are available in Appendix 3

The following table contains the specifications for the devices used in the test:

Device	RAM	CPU	GPU	OS	Price
Raspberry PI	512MB	Broadcom 700MHz	Broadcom VideoCore IV	Linux (Raspberrian)	€32.00
Banana PI	1GB	ARM Cortex-A7 1GHz (dual core)	Mali400MP2 GPU	Linux (Raspberrian)	€32.00
Orange PI 2	1GB	ARM Cortex-A7 1.6GHz (quad core)	Mali400MP2 GPU - 600MHz	Linux (Raspberrian)	€35.00
Lenovo ThinkPad x220	8GB	Intel i5 vPro 2.6GHz (2 core, 4 threads)	Intel Graphics 3000 - 650MHz	Windows 8	

Table 2: Device Specification

3.2.1 Motion Detection Algorithms

The BGSLibrary contains a number of different implementations of motion detection algorithms, which is why it was decided to reuse the source code available in GitHub instead of re-implementing the algorithms to be tested. The latest version of the source code available at the time of writing of this thesis was used, which is 1.9.2. The only exception is the PBAS algorithm, which was removed from the later versions of the library, and therefore it was taken from an earlier branch. The only problem with earlier version of the library is that algorithms only returned the foreground mask and not the background model, as a result, transparency removal filter could not be utilized. To overcome this, a custom background model filter was developed, which is described further on. Modifying the PBAS algorithm was not possible as the change would be too involved due to the way the background model is persisted. In addition, in would add not only complexity, but also extra load on the CPU as the existing multi-layer background model would need to be converted into a single layer.

The source code for the ViBe algorithm was acquired from its authors as it's not part of the BGSLibrary. This source code was modified in order to expose the background model. Although PBAS algorithm is based on ViBe, internally ViBe's background model is also kept in form of a single-layer frame, which allowed expositing it without extra processing. In addition, the algorithm was wrapped in a generic class so it can be called polymorphically, just like the other BGS algorithms.

3.2.1.1 Algorithm Configuration

It is important to keep the algorithm configuration separate from the compiled code - a requirement well taken care of by the BSGLibrary. The library stores the configuration in XML files, which makes it easy to modify it even at run-time, if required. The configuration used for the algorithms is available in Appendix 5.

As mentioned previously, the ViBe algorithm was wrapped in a generic class so it can be treated in code just like the other BGS algorithms. The so called "interface" (located in IBSG.h file) contains methods for loading and saving configuration of the algorithm. These were implemented in the wrapper so that the configuration is also saved in XML format, just like other BGS algorithms. It should be noted that, although method for updating the setting "number of samples" is available, the actual method is not implemented and therefore the default value is always be used.

Additional configuration can be supplied with command-line arguments, however, this configuration is mainly specific to post-detection filters, rather than motion detection algorithms. The full list of available options is described in Appendix 4.

3.2.2 Post-detection Filters

The post-detection filters are filters that are applied to the segmentation output and perform their task only on pixels that were identified as foreground. These filters were developed to fix a specific problem, which generic motion detection algorithms can't.

3.2.2.1 Transparency Removal Filter

It was observed that very often a moving cobweb changes a large blob of pixels causing the blob to be classified as foreground. It was also observed that this blob of pixels very often looks transparent as it would often be out of focus. Although classifying this type of blob as a foreground would be correct from the perspective of the motion detection algorithm, this type of object would need to be suppressed, as the operator does not need to be aware of it. In order to do this, the incoming frames, foreground and background models are scanned to detect any feature points. Following the detection, the coordinates of each of the feature points in the foreground are compared against those in the background. If the distance between the coordinates is within a given threshold, the section covered by the feature point is removed from the foreground mask. The section to be removed would have a square shape with side lengths configured through command-line parameter "*Removal Shape Size*". The feature point displacement threshold was also made dynamic, and therefore can be changed through command-line parameter "*Distance Threshold*". This detection method assumes that the intruder object would not have the same feature points in the same location as the background.

One of the important points about this approach is to ensure that the foreground of interest (i.e. intruder) is not merged into background, as otherwise this would cause the feature points to be located within the displacement threshold, causing the motion object to be removed from the foreground mask.

The feature detection algorithms used to detect feature points were those available in the OpenCV library. Although all of the tested algorithms showed similar results in terms of accuracy, the "FastFeatureDetector" proved to be the fastest and most accurate, therefore it was selected as the default detector. Other detectors can be selected dynamically using the command-line arguments.

In order to optimize performance of this filter, the only pixels that were processed by the detector were those that were present in the motion mask.

3.2.2.2 Colour-based Filter

It was observed that in certain situations the moving objects in front of the camera reflect so much light, that the object becomes completely white (see *Figure 10: Contrast difference between a genuine moving and over-saturated object*).

The intruder on the other hand would normally reflect far less light and therefore would normally appear in grey colour.

Given these two rules, the filter was developed so that pixels brighter than certain threshold are removed from the motion mask.

3.2.2.3 Heat Map Filter

It was also observed that noise objects would either have the same motion pattern or would be present in the frame temporarily, although it is enough to be detected as foreground by some motion detection algorithm. In order to take care of this scenario, the heat map filter was developed, which would classify a pixel as foreground only in situations where the same pixel was detected as foreground in a number of previous frames subsequently.

The pixels that are detected as a foreground cause the relative pixels in the heat map model to be incremented (up to a value of 255), alternatively those pixels are decremented until the value reaches zero.

The configuration of this filter is done through command-line arguments. The options available for this filter are the minimum and maximum heat values that the heat map should reach in order to classify the motion pixel as foreground. Where the heat value of a pixel is under the minimum or over the maximum value, the pixel is not classified as foreground and is removed from the motion mask, but the heat value is still incremented or decremented.

This filter works based on the assumption that the noise objects move too fast, whereas intruders would move slower as they're further away from the lens.

Similar approach is used by the Sigma-Delta algorithm (Manzanera & Richefeu, 2007) where a statistical model is built by incrementing or decrementing related pixels. However, the core difference between the two approaches are that the Sigma-Delta algorithm increments and decrements Gaussian distributions, whereas the heat map filter operates on bit masks that represent foreground pixels.

3.2.2.4 Custom Background Model

Some algorithms, such as Sigma-Delta or PBAS, either don't have or don't expose the background model used internally, which makes it impossible for the transparency filter to perform it's task. In order to overcome this limitation, a custom background model was developed, which can be enabled on demand using the command-line argument. The model works by merging incoming pixels into the existing model by selecting pixels randomly, with the exception of the pixels that were detected as a foreground - these are ignored.

The available command line arguments are the "*pixel update step*" and "*frame update step*". The first argument must be more than or equals to one, as it is used to determine which pixels should be skipped and which should be updated. At runtime, this value is used to get the random integer that is less than or equals to this value as otherwise some pixels would never be updated. If the value of this parameter is one, it would cause every pixel to be updated.

The second argument indicates how many frames should be skipped before the update happens and therefore the value must be more than zero in order to skip frames. The logic to skip frames is only required to improve performance.

3.2.3 Ground Truth Comparison

The ground truth is represented by a mask for each of the video frames captured by the camera. The pixel values of this mask are as follows:

- 0 - Static
- 50 - Hard Shadow
- 85 - Outside the region of interest
- 170 - Unknown Motion, such as distortion caused by a drop of rain on a lens
- 255 - Motion

The ground truth comparison was performed between the foreground mask and the related ground truth frame. Some of the ground truth sets were supplied by the " *Change Detection Challenge* ". The ground truth for the footage recorded using a local camera was established manually, which followed the same convention as in the " *Change Detection Challenge* " data set.

The results of this comparison would be a percentage of correct classification (PCC) and it would be written to a CSV file along with the frame number, input and ground truth file names.

The pixels were classified as correctly identified only when at least one of the following conditions held true:

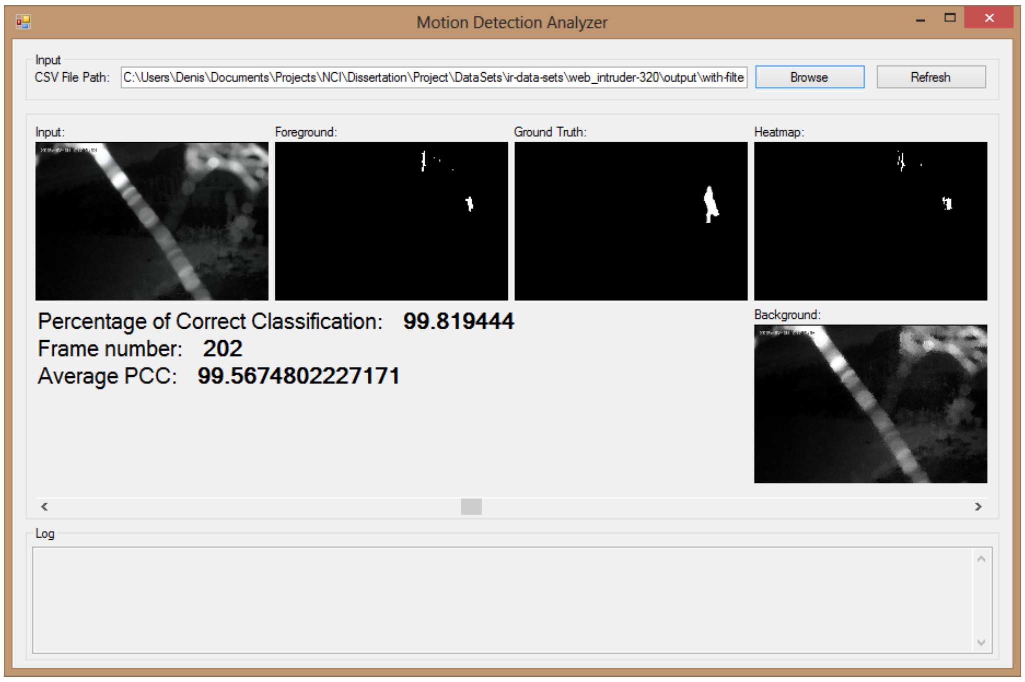
- Ground truth pixel has a value of 50 (Hard Shadow)
- Ground truth and motion mask pixels both have a value of 255 (Motion)
- Ground truth and motion mask pixels both have a value of zero

The pixels matching the first condition above would essentially be ignored. This is because the shadow could be caused by an actual moving object, but not every motion detection algorithm is able to separate shadow pixels.

3.3 Test Result Analysis Tool

This GUI tool (MotionDetectionAnalyzer) was developed to facilitate the analysis of the results generated by the testing harness. It was built using the Windows Forms .NET technology and therefore can only be run under Windows environment. The tool allows the user to select a CSV file containing the mappings between the different frame types and the PCC rates. This tool works under the assumption that the CSV file and the files referred to in the CSV file are located within the same directory.

When the CSV file is loaded, each of the files referenced in the first line are loaded into a corresponding section of the GUI window. The scroll bar below the frames can be moved back and forward in order to fast-forward or rewind the video sequence. This allows the user to carefully inspect each of the frames (and their different states) in order to understand better the weaknesses of a particular approach of configuration.



CSV File Path: C:\Users\Denis\Documents\Projects\NCI\Dissertation\Project\DataSets\ir-data-sets\web_intruder-320\output\with-filte

Percentage of Correct Classification: **99.819444**
Frame number: **202**
Average PCC: **99.5674802227171**

Figure 13: Test Result Analysis Tool screenshot

The “Percentage of Correct Classification” label contains the percentage of pixels that were correctly detected for the frame identified by the “Frame Number” label, whereas the value next to “Average PCC” label contains the average PCC across all frames.

4 Results and Analysis

The aim of this research is to identify background segmentation algorithms capable of accurately separating the foreground from the background on Infra-Red feeds. These algorithms should be able to process video streams in real-time, on resource-constrained devices. CCTV Monitoring centres often use motion detection as a way of detecting intrusions, but it normally leads to numerous false detections caused by rain, flies and cobwebs. This causes operators to become less vigilant, increasing the chance of intruders gaining access undetected.

4.1 Methodology

This research focused on a number of motion detection algorithms, which according to their authors were able to accurately segment video sequences, while producing high FPS rate. Each of the algorithms was tested on a large data set containing a variety of night scenes. Furthermore, each of the algorithms was also tested with different post-processing filters. Each of the test runs produced two main values, the FPS and PCC rates which were used to determine the suitability of a particular approach for the problem in hand. Therefore, the methodology chosen for this research was quantitative.

4.2 Model

A number of scripts were developed which encapsulated various configurations for the Segmentation Tester utility, such as input and output paths to video feeds, working width, algorithms and filters to be used and filter configurations. One script was developed per data set, containing various conditions to be tested. The Segmentation Tester tool would be called from within the script, which would iterate sequentially over the frames in the data set, separate the foreground from the background and then compare the resulting foreground mask against the ground truth model. This would then produce a PCC value for each iteration. The tool would also accurately measure the time consumed by the segmentation process, excluding the time spent on frame acquisition and other pre-processing tasks. The time spent by the segmentation logic was then used to generate the FPS rate.

Each test run produced one CSV file, which contained the PCC rate. The FPS rate was not included in the CSV file, but instead was output to standard output (e.g. the command line window), which was then captured in a LOG file.

4.3 Data Sets

In order to accurately measure the performance of a particular algorithm, 10 different data sets were used. Three of these data sets were recorded using a local CCTV camera. These contained footage of the same scene before and after the cobweb appeared in the view, and with intruder present and absent.

In order to determine whether higher resolution frames has any positive effect on accuracy of detection, the first 3 data sets were produced with resolution of 720x405 pixels. For lower resolution testing, each of the frames was resized on-the-fly by the testing utility.

The remaining 7 data sets were acquired from "*Change Detection Challenge*" website. This data set contained night outdoor footage of public places, such as a street, bridge and motorway. The frames sizes and aspect ratios in these data sets varied, therefore, in order to achieve accurate test results, the working width was adjusted so that the total amount of pixels processed by the Segmentation Tester would be similar between all tests.

4.4 Conditions Tested

There are many variables involved in the configuration of each algorithm and filter. Since testing each one of them by "*brute force*" would be practically impossible, each algorithm was configured to optimum settings where the real motion object could be detected as close as possible to the ground truth, disregarding the level of environmental noise detected along the way. The contents of the configuration XML files for each of the algorithms tested can be found in Appendix 5.

Once the segmentation algorithms were set, post detection filters were configured to suppress environmental noise with minimum knock-on effect on objects of interest. The configuration options for each of the filters are available in Appendix 4.

Although the primary focus of this research were algorithms ViBe, PBAS and Sigma-Delta, other algorithms available as part of BGSLibrary project were also tested for comparison.

Each test run was made against 7 algorithms, 10 night-time data sets and 4 types of devices. Every run was different, with variations in working width and filters applied. The following lists show the algorithms, data sets and variations used for every run.

Algorithms Tested

- Adaptive Background Learning
- Adaptive Selective Background Learning
- Weighted Moving Mean
- Static Frame Difference
- PBAS
- Sigma Delta
- Vibe

Data Sets Tested

- back yard without cobwebs and with intruder
- back yard with cobwebs and with intruder
- back yard with objects over-saturating the view
- winter street
- tram station
- street corner at night
- fluid highway
- busy boulevard
- bridge entry

Test Variations

Working width proportional to 320x180 pixels	Working width proportional to 720x405 pixels	Transparency Filter Enabled	Custom Back-ground Model Enabled	Heatmap Filter Enabled
Y	N	N	N	N
Y	N	Y	N	N
Y	N	Y	Y	N
Y	N	N	N	Y
N	Y	N	N	N
N	Y	Y	N	N
N	Y	Y	Y	N
N	Y	N	N	Y

Table 3: List of conditions used for each test run

4.5 Conditions Excluded

Tests with working width proportional to 720 pixels and those that used PBAS algorithm were excluded from embedded devices and were only performed on a laptop against the data sets recorded by a local camera. This is because the data acquired from the first 3 tests carried out on the development machine was sufficient to answer the related question.

In addition, tests on wide angle feeds was not carried out as such feeds could not be acquired.

4.6 Experiment Results

The experiment was conducted initially on the first 3 data sets recorded with a local camera. The first data set contained a clear image, without cobwebs and with intruder appearing in the view after approximately 220 frames. This test was performed in order to acquire the base PCC and FPS rates, which was then compared with the results from other data sets and algorithms of the same scene. The "*Change Detection Challenge*" data set was used to acquire results from a wider range of scenarios, which don't necessarily have a cob-web problem.

The first test was executed on a development machine using high and low resolution feeds. The primary focus of this experiment was to determine whether the higher resolution images can have a positive effect on segmentation accuracy. Therefore, no post-detection filters were utilized during this test.

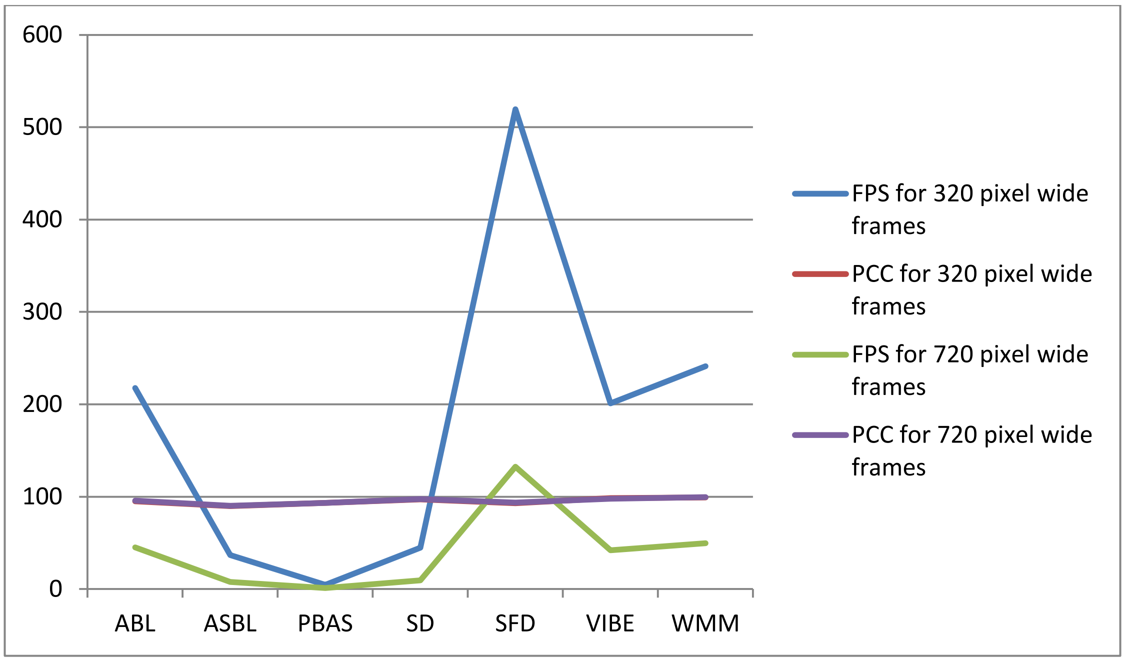


Figure 14: FPS comparison from low and high resolution tests

The test result showed that increasing the resolution of the feed does indeed increase the PCC rate, but this increase was negligible, especially when taking into account the dramatic decrease in FPS rate, as can be seen in *Table 7*. The frame sizes for low and high resolution tests were 320x180 and 720x405 pixels respectively.

Following the high resolution tests, feeds with lower resolutions were tested on all devices and against all data sets. As anticipated, the PCC rate did not change between tests conducted on different platforms, therefore the chart below shows the average data acquired by all 4 platforms.

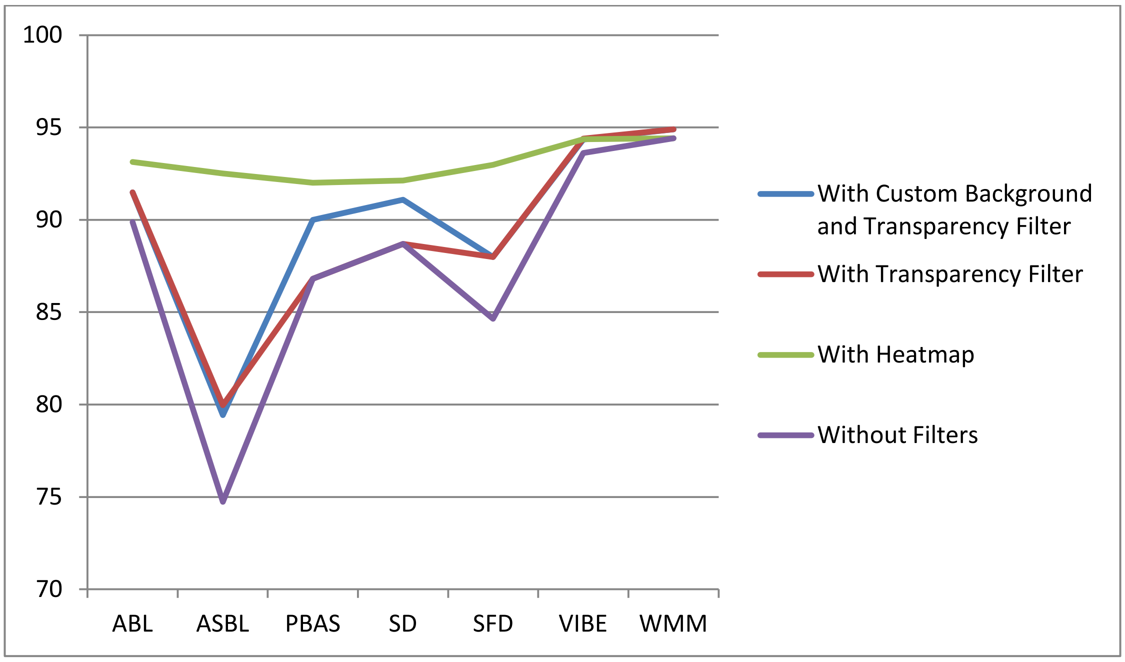


Figure 15: Post-detection filter PCC rate comparison

The results showed that every post-detection filter increased the PCC rate, although the heat map filter was able to provide the highest and the most consistent result. The transparency filter and custom background model also showed an improvement in detection accuracy, although the custom background model barely had any effect when used in conjunction with algorithms that expose their own background model. This indicates that the custom background model is at least as efficient as that in other state-of-the-art algorithms.

The algorithms identified as the most effective were ViBe, Weighted Moving Mean and Adaptive Background Learning. The FPS rate achieved by these 3 algorithms on Banana PI showed that only ViBe algorithm was capable of processing video feeds in real-time, whereas results obtained from Raspberry PI showed that none of these algorithms were able to perform fast enough. The Orange PI met the minimum requirement for real-time processing (approximately 12 FPS) for all 3 algorithms and 3 filters. However, the increase in FPS doesn't justify the increase in PCC in most cases, with exception of Adaptive Background Learning, as can be seen in the following tables.

	Adaptive Back-ground Learning	ViBe	Weighted Moving Mean
FPS	4.32%	5.56%	5.4
PCC	3.5%	0.78%	0%

Table 4: FPS increases proportional to PCC increases for Heatmap filter on Orange PI

	Adaptive Back-ground Learning	ViBe	Weighted Moving Mean
FPS	64.2%	67.02%	72.11%
PCC	1.8%	0.03%	0.51%

Table 5: FPS increases proportional to PCC increases for Transparency Removal filter on Orange PI

The transparency filter was not able to achieve the minimum requirement for real-time processing on Banana PI, The heatmap filter, on the other hand, was able to achieve approximately 12.56 FPS on this board, as can be seen from the following charts.

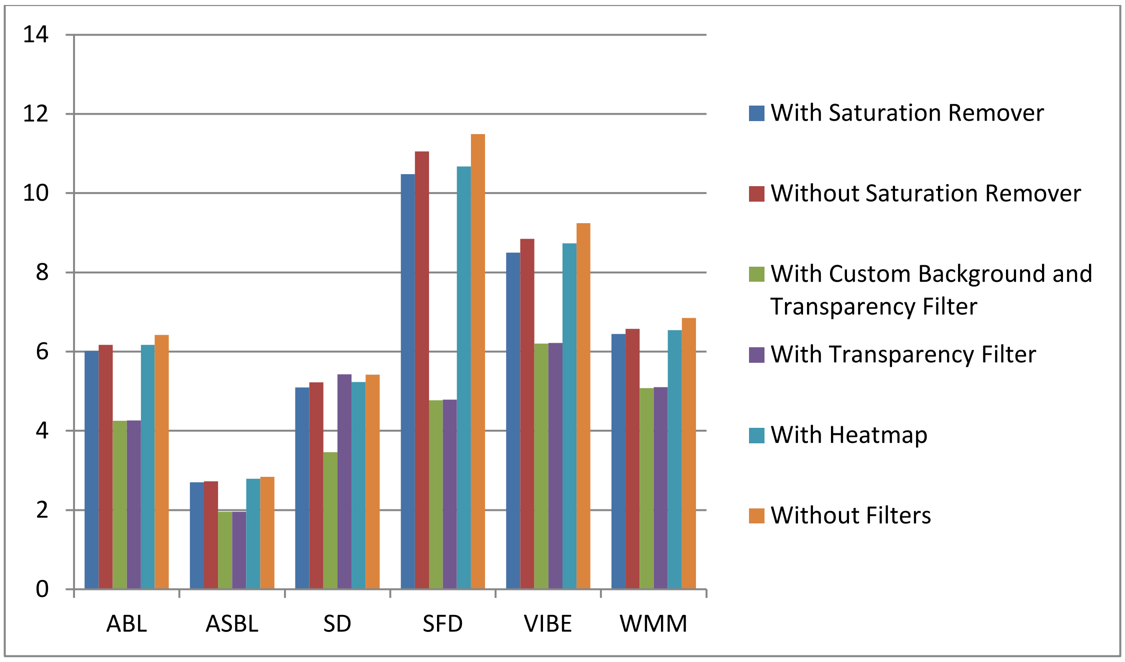


Figure 16: FPS rates comparison between algorithms and filters. Test conducted on a Raspberry PI.

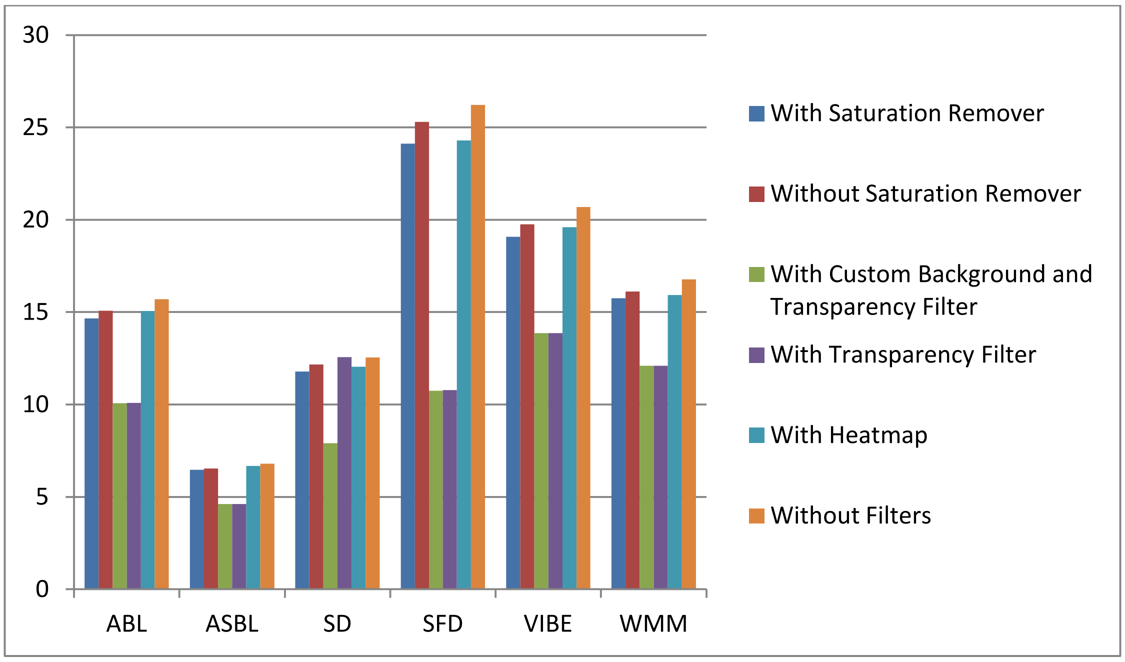


Figure 17: FPS rates comparison between algorithms and filters. Test conducted on a Orange PI.

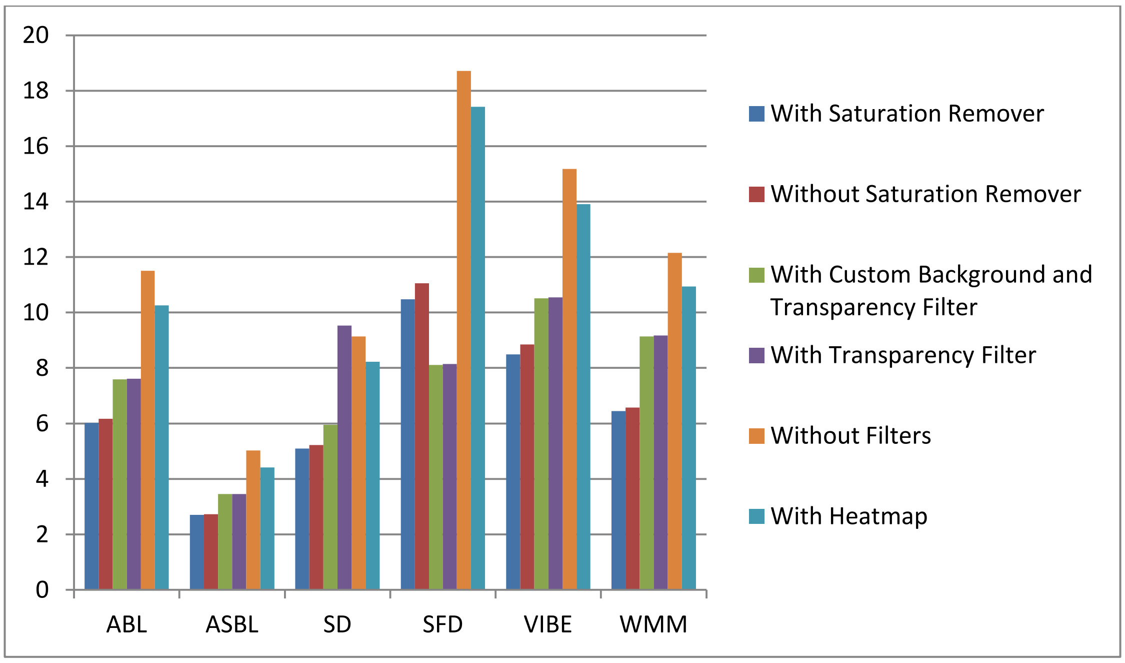


Figure 18: FPS rates comparison between algorithms and filters. Test conducted on a Banana PI.

The Saturation Remover filter was developed specifically to ignore parts of the motion mask which represents pixels that are much brighter than those of a potential intruder. Therefore, this filter cannot be classified as universal, unlike other filers and algorithms investigated in this research. As a result, this filter had to be tested using a feed that contained over-saturated pixels.

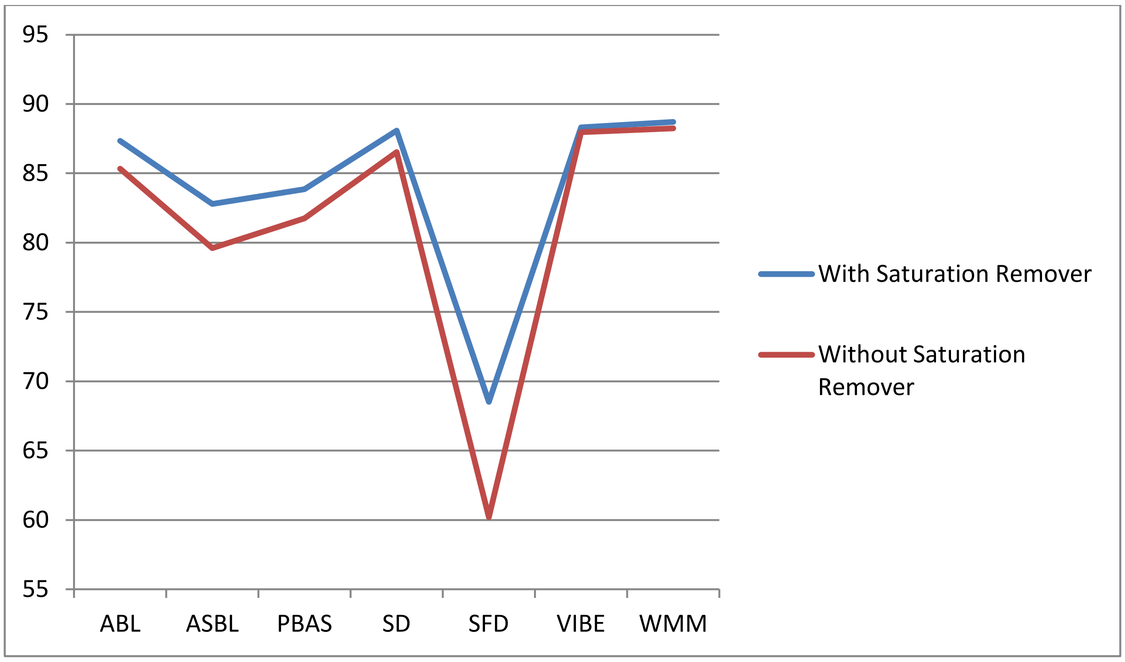


Figure 19: Saturation Remover filter PCC rates comparison

The test results showed an increase in accuracy by almost 14%, as it is the case with Static Frame Difference algorithm. Other algorithms were also more accurate with this filter, but the increase in PCC was not as big.

Orange PI was the only board capable of achieving real-time FPS rate with algorithms Adaptive Background Learning, Static Frame Difference, ViBe and Weighted Moving Mean. However, the increase in PCC didn't justify the increase in FPS for ViBe and Weighted Moving Mean algorithms, as can be seen from the following table.

Increase	Adaptive Background Learning	Static Frame Difference	ViBe	Weighted Moving Mean
FPS	2.83%	4.66%	3.35%	2.3%
PCC	2.35%	13.78%	0.39%	0.5%

Table 6: FPS increases proportional to PCC increases for Saturation Remover filter

The accuracy of detection was also measured separately for the two kinds on data sets.

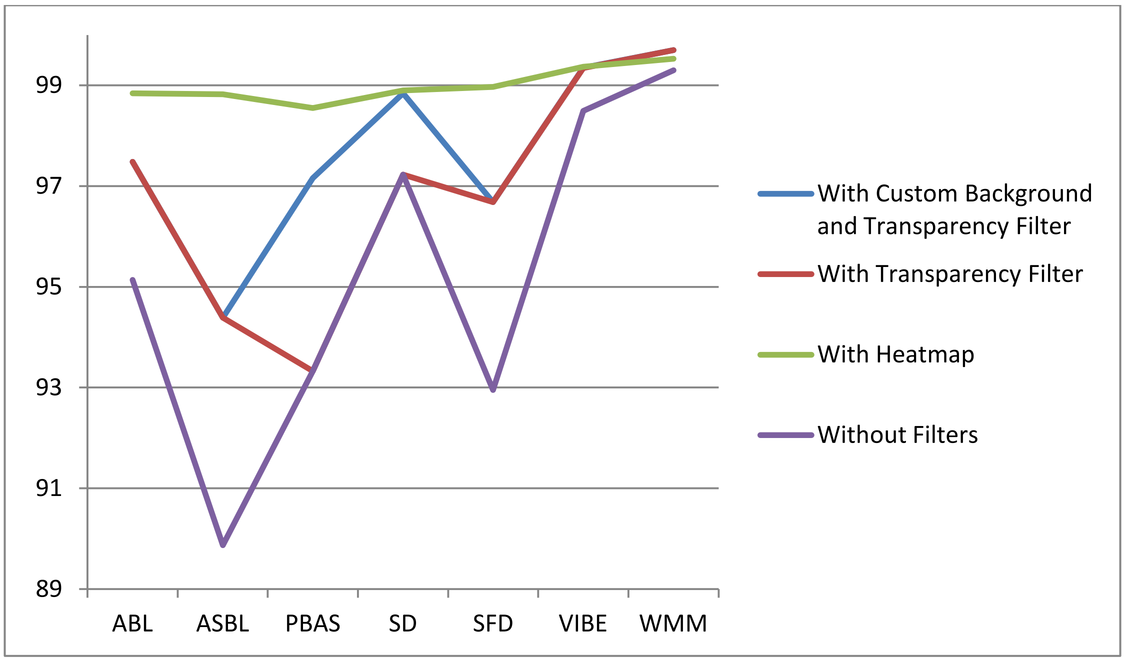


Figure 20: PCC rates achieved from feeds containing cobwebs

The data in the above chart showed that the algorithms ViBe and Weighted Moving Mean were able to detect over 99% of pixels correctly on IR feeds. The 3 filters also showed a significant increase in PCC rates for some algorithms. However, the chosen algorithms and post-detection filters were not so effective against feeds from "*Change Detection Challenge*" data sets, as can be seen in the following chart.

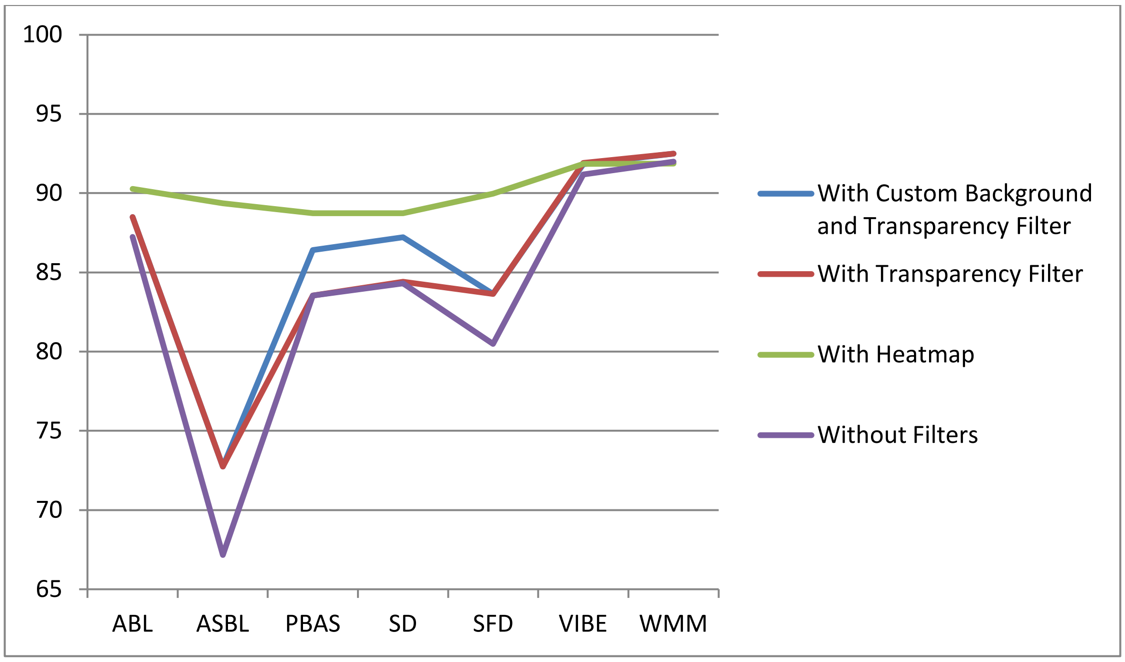


Figure 21: PCC rates achieved from "Change Detection Challenge" data sets

Apart from the PCC rates, foreground and ground truth masks were inspected, which showed a significant decrease in detection of unwanted objects, whilst the moving object of interest was not affected badly.

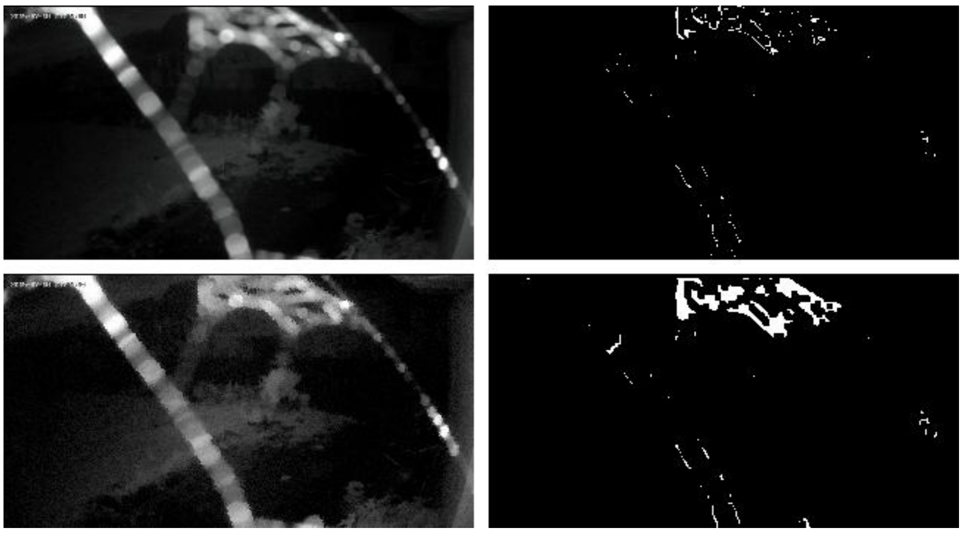


Figure 22: Noise object suppression example (no intruder)

Top-left: input, Top-right: heatmap mask, Bottom-left: background model, Bottom-right: ViBe foreground mask

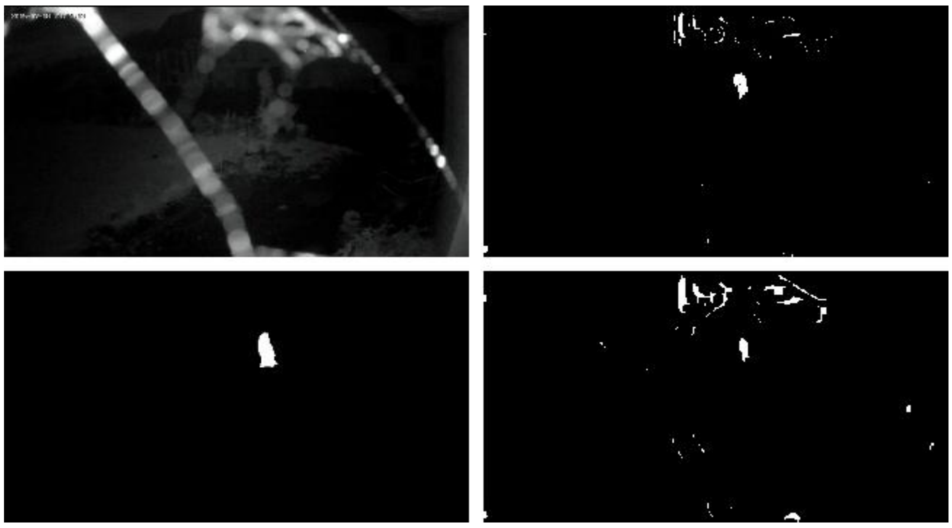


Figure 23: Noise object suppression example (intruder present)

Top-left: input, Top-right: heatmap mask, Bottom-left: ground truth, Bottom-right: ViBe foreground mask

It should be noted that the IR feed captured with a local camera was darker than normal, especially when compared to feeds as in *Figure 10*. As a result, more accurate detection happened when the moving object moved closer to the camera as it became brighter. This caused a decrease in PCC rate, meaning that with brighter feeds, the results could've been much better.

4.7 Conclusion

The 7 algorithms and 4 filters were tested on 4 different devices, using 10 different data sets. Algorithms such as Adaptive Background Learning, ViBe and Weighted Moving Mean were able to segment the foreground from the background most accurately. However, the CPU on Raspberry PI board was not powerful enough to process video feeds in real-time.

The Banana PI board was able to process video feeds in real time using algorithms ViBe, Weighted Moving Mean and Heatmap filter. The Orange PI board was capable of processing video feeds in real-time using all 3 algorithms. The 4 filters also reached the real-time requirements, but only for ViBe and Weighted Moving Mean, as Adaptive Background Learning was not able to process feeds in real-time with Transparency Filter.

Inspecting the resulting segmentation masks showed a noticeable decrease in incorrectly detected pixels, but none of the algorithms or filters were able to completely remove the noise objects. That said, the feed containing noise objects was too dark, therefore brighter IR feeds could potentially result in more accurate detections as a higher threshold could be used without affecting foreground objects.

Alternative data sets (without noise objects) were also tested in similar manner, which showed that the filters had a negligible increase in PCC rates, but significantly increased the FPS rates.

High resolution feeds were also tested, which showed that increasing the frame size did not significantly increase the PCC rate, but did decrease the FPS rate substantially.

5 Conclusion and Future Work

5.1 Conclusion

The incorrectly detected objects on CCTV feeds is a big problem for monitoring centres and security businesses as it increases the amount of people required to monitor customer sites. This increases monitoring costs to consumers. With the growing popularity of IP cameras, the segmentation process is moving away from DVRs or monitoring centres to IP cameras, which are run on embedded devices. This research attempted to identify background segmentation algorithms capable of accurately segmenting video feeds, while being run on cheap and low-power boards, such as Raspberry PI, Banana PI and Orange PI. In order to do this, a utility was created, which allowed performing tests against various types of video feeds using different variations of algorithms, filters and configurations.

In order to verify the effectiveness of different approaches, the test utility was run against data sets containing noise objects located too close to the camera and clear feeds, mainly featuring streets and boulevards without any noise objects. The results showed that although none of the tested algorithms were able to completely remove noise motion, algorithms such as ViBe and Weighted Moving Average were the most accurate. The results also showed that simple filters, such as Heatmap, can significantly increase the accuracy rates without serious knock-on effects on FPS rates.

This research also investigated whether increasing the frame size could increase the accuracy of the detection. The test was conducted against data sets with 320x180 and 720x405 pixel frames. The results showed that the FPS rate increased 5 times, but practically no increase in PCC was registered.

The boards tested as part of this research showed that Raspberry PI is not suitable for surveillance purposes as it was not able to reach real-time frame rate with any of the algorithms. The results acquired from boards Banana PI and Orange PI showed that they could be suitable for surveillance purposes as they were able to reach real-time frame rate with most of the algorithms.

5.2 Future Work

There is a multitude of studies done in an attempt to develop accurate foreground segmentation algorithms, however, the testing of new algorithms is often performed against data sets containing images with clearly visible differences. Such tests can show promising results, but in real life and especially with IR feeds, could prove to be inefficient. This is why more research is required in this field, which needs to be conducted against problematic video feeds.

This research evaluated the performance of boards containing more than one CPU core, however, it is unknown whether the algorithms tested utilized more than one core. The development of solutions capable to be run on multiple cores simultaneously can be complicated, but could potentially double or quadruple FPS rates on boards such as Banana PI and Orange PI, as these have 2 and 4 cores respectively.

The smallest frame size used to conduct testing was 320x180 pixels, which was above the average frame size tested by other researchers. Since this research showed that increasing frame size has no noticeable effect on PCC rates, further investigation is required to determine the minimum size of the frame that could be used in foreground segmentation. If the amount of pixels decreases, the FPS rate would increase, allowing for more complex calculations to be performed instead.

This research also showed that simple filters, such as Heatmap can make a big difference in PCC without big decrease in FPS. The Heatmap filter "remembers" the most triggered pixels or patterns and tries to suppress them. However, suppressing such pixels can have an adverse effect on intruders which appear in the same area, as these could also be suppressed. Therefore, further research is required to determine whether pattern databases can be utilized to determine whether the detected shape is reoccurring in a specific location too often, and in case it is, it could be suppressed. This differs from the Heatmap filter because new shapes, even if appearing in areas of suppressed shapes, could be detected as objects of interests, until classified otherwise.

References

Andrews, S., 2015. *A Background Subtraction Library.* [Online]
Available at: https://github.com/andrewssobral/bgslibrary
[Accessed 01 06 2015].

Anon., 2010. *YouTube - Infrared Snow.* [Online]
Available at: https://www.youtube.com/watch?v=ereevhGu7UI
[Accessed 15 08 2015].

Anon., n.d. *ViBe - a powerful technique for background detection and subtraction in video sequences.* [Online]
Available at: http://www2.ulg.ac.be/telecom/research/vibe/
[Accessed 27 05 2015].

Antoine, M. & Julien, C. R., 2007. A new motion detection algorithm based on R–D background estimation. *Pattern Recognition Letters,* 28(3), pp. 320-.

Banana Pi, 2015. *Banana Pi.* [Online]
Available at: http://www.bananapi.com/
[Accessed 27 08 2015].

Birmohan, S. et al., 2014. Motion Detection for Video Surveillance. *International Conference on Signal Propagation and Computer Technology,* pp. 578 - 584.

Business Wire, 2010. *Ambarella A5s IP Camera Platform Delivers Industry-Changing Combination of Image Quality, HD H.264 Multi-Streaming, CPU Flexibility, and Low Power Consumption.* [Online]
Available at:
https://ezproxy.ncirl.ie/login?url=http://ezproxy.ncirl.ie:2103/docview/443533009?accountid=103381
[Accessed 27 03 2015].

Chaney, M., n.d. *Video Frame Rates (24p, 25p, 30p, 60i).* [Online]
Available at: http://www.steves-digicams.com/knowledge-center/video-frame-rates-24p-25p-30p-60i.html
[Accessed 04 04 2015].

Ching, Y. Y., Rubita, S. & Kim, M. C., 2011. Motion Detection and Analysis with Four Different Detectors. *Third International Conference on Computational Intelligence, Modelling & Simulation.*

Drinkwater, J., 2010. *The NW Blog.* [Online]
Available at: http://www.networkwebcams.com/ip-camera-learning-center/2010/12/22/sony-snc-ch160-ip-security-camera-review-evaluation/
[Accessed 14 08 2015].

Duarte, D., Henrique, S. & Paulo, C., 2006. The OBSERVER: An Intelligent and Automated Video Surveillance System. *Image Analysis and Recognition,* pp. 898 - 909.

Hanzi, W. & David, S., 2006. Background Subtraction Based on a Robust Consensus Method. *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06).*

Heijmans, H. & Ronse, C., 1990. The algebraic basis of mathematical morphology. I. dilations and erosions. *Computer Vision, Graphics, and Image Processing,* 50(3), pp. 245 - 295.

Hofmann, M., Tiefenbacher, P. & Rigoll, G., 2012. Background Segmentation with Feedback: The Pixel-Based Adaptive Segmenter. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops,* pp. 38 - 43.

Honovich, J., 2011. *Average Frame Rate Used for Recording.* [Online]
Available at: http://ipvm.com/updates/1100
[Accessed 16 08 2015].

Hyenkyun, W., Yoon, M. J., Jeong-Gyoo, K. & Jin, K. S., 2010. Environmentally Robust Motion Detection for video surveillance. *IEEE TRANSACTIONS ON IMAGE PROCESSING,* 19(11).

Kentaro, T., John, K., Barry, B. & Brian, M., 1999. Wallflower: Principles and Practice of Background Maintenance. *Proceedings of the Seventh IEEE International Conference on Computer Vision,* Volume 1, pp. 255 - 261 .

Kevala, H. U. & Sasse, M. A., 2006. Man or a Gorilla? Performance Issues with CCTV Technology in Security Control Rooms.

Manzanera, A. & Richefeu, J., 2007. A robust and computationally efficient motion detection algorithm based on sigma-delta background estimation. *Pattern Recognition Letters,* 28(3).

Massimo, P., 2004. Background subtraction techniques: a review. *IEEE International Conference on Systems, Man and Cybernetics,* pp. 3099-3104.

Mishra, S., Mishra, P., Chaudhary, N. & Asthana, P., 2011. A Novel Comprehensive Method for. *International Journal of Scientific & Engineering Research,* II(4).

Olivier, B. & Marc, V. D., 2011. ViBe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image Processing,* 20(6), pp. 1709-1724.

Open Source Computer Vision Library, 2015. *OpenCV.* [Online]
Available at: http://opencv.org/
[Accessed 01 06 2015].

Pierre-Marc, J., Janusz, K., Prakash, I. & Fatih, P., 2010. *Change Detection Challenge.* [Online]
Available at: http://www.changedetection.net/
[Accessed 05 27 2015].

Pierre-Marc, J., Janusz, K., Prakash, I. & Fatih, P., n.d. *www.changedetection.net.* [Online]
Available at: http://www.changedetection.net/
[Accessed 05 27 2015].

Pierre-Marc, J., Max, M. & Janusz, K., 2007. Statistical Background Subtraction Using Spatial Cues. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* 17(12), pp. 1758-1763.

Pool, M., n.d. *distcc GitHub.* [Online]
Available at: https://github.com/distcc
[Accessed 20 07 2015].

Raspberry PI Foundation, n.d. *Raspberry PI 1 MODEL B.* [Online]
Available at: https://www.raspberrypi.org/products/model-b/
[Accessed 01 06 2015].

Srivastava, A., Lee, A. B., Simoncelli, E. P. & Zhu, S., 2003. On advances in statistical modeling of natural images. *Journal of Mathematical Imaging and Vision,* 18(1), pp. 17 - 33.

Stanley R. Sternberg, CytoSystems Corporation, 1983. Biomedical Image Processing. *IEEE .*

Sze, L. T., Zulaikha, K., KimMeng, L. & Mei, K. L., 2010. Hybrid Blob and Particle Filter Tracking Approach for Robust Object Tracking. *International Conference on computational Science (ICCS),* 1(1), p. 2549–2557.

Tetsuya, M., Makoto, N., Tomohiro, Y. & Shinji, T., 2010. Comparison of Color Space in Extraction of a Hand Region for Computer Human Interface Using Color Image. *Technical report of IEICE. PRMU 98(528).*

Widyawan & Muhammad, I. Z., 2012. *Adaptive Motion Detection Algorithm using Frame Differences and Dynamic Template Matching Method.* Daejeon, s.n.

Xunlong Software CO.,Limited, 2015. *Orange Pi.* [Online]
Available at: http://www.orangepi.org/
[Accessed 27 08 2015].

Yun, Z. & Arun, H., 2009. Virtual Boundary Crossing Detection without Explicit Object Tracking. *IEEE International Conf. on Advanced Video and Signal Based Surveillance (AVSS),* pp. 518-522.

Zulaikha, K. et al., 2012. Video Analytics Algorithm For Detecting Objects Crossing Lines In Specific Direction Using Blob-Based Analysis. *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia,* pp. 306-312 .

Appendix 1 - Test Results

	FPS for 320 pixel wide frames	PCC for 320 pixel wide frames	FPS for 720 pixel wide frames	PCC for 720 pixel wide frames
ABL	217.6	95.14	44.94	95.7
ASBL	36.69	89.87	7.72	90.19
PBAS	4.72	93.33	0.95	93.28
SD	44.65	97.23	9.34	97.54
SFD	519.45	92.95	132.39	93.63
VIBE	200.98	98.49	42.11	97.64
WMM	241.18	99.3	49.47	99.38

Table 7: FPS and PCC comparison from low and high resolution tests conducted on a laptop

	With Custom Background and Transparency Filter	With Transparency Filter	With Heatmap	Without Filters
ABL	91.48	91.48	93.13	89.87
ASBL	79.43	79.96	92.51	74.73
PBAS	89.99	86.8	92.01	86.8
SD	91.09	88.69	92.12	88.69
SFD	87.98	87.98	92.97	84.64
VIBE	94.34	94.39	94.36	93.62
WMM	94.9	94.9	94.42	94.42

Table 8: Post-detection filters PCC rate comparison

	With Saturation Remover	Without Saturation Remover
ABL	87.343147	85.337051
ASBL	82.797825	79.608691
PBAS	83.847845	81.760345
SD	88.072619	86.533222
SFD	68.510458	60.21152
VIBE	88.318311	87.971589
WMM	88.695623	88.254402

Table 9: Saturation Remover filter PCC rates comparison

	With Saturation Remover	Without Saturation Remover	With Custom Background and Transparency Filter	With Transparency Filter	With Heatmap	Without Filters
ABL	200.94	214.555	114.47	113.25	217.11	219.2
ASBL	52.09	52.9412	24.88	24.65	47.03	45.92
PBAS	4.70	4.76244	3.89	4.2	4.19	4.14
SD	36.31	46.4043	35.45	48.01	46.94	47.06

SFD	342.69	519.757	98.03	100.14	441.76	537.65
VIBE	173.25	176.836	135.47	130.491	193.15	206.56
WMM	228.30	233.607	187.83	180.71	229.88	242.2

Table 10: FPS rates comparison between algorithms and filters. Test conducted on a laptop

	With Saturation Remover	Without Saturation Remover	With Custom Background and Transparency Filter	With Transparency Filter	With Heatmap	Without Filters
ABL	6.01901	6.17106	4.25	4.26	6.17	6.42
ASBL	2.70356	2.72901	1.96	1.96	2.79	2.84
SD	5.09535	5.22297	3.46	5.43	5.23	5.42
SFD	10.4779	11.0537	4.77	4.79	10.67	11.49
VIBE	8.49478	8.84635	6.2	6.22	8.73	9.24
WMM	6.44796	6.57187	5.08	5.1	6.54	6.85

Table 11: FPS rates comparison between algorithms and filters. Test conducted on a Raspberry PI

	With Saturation Remover	Without Saturation Remover	With Custom Background and Transparency Filter	With Transparency Filter	With Heatmap	Without Filters
ABL	14.653	15.0794	10.06	10.08	15.05	15.7
ASBL	6.4577	6.5342	4.61	4.61	6.67	6.8
SD	11.7769	12.1708	7.9	12.56	12.04	12.54
SFD	24.1185	25.2959	10.75	10.78	24.29	26.22
VIBE	19.0848	19.746	13.86	13.86	19.59	20.68
WMM	15.7459	16.1169	12.09	12.1	15.92	16.78

Table 12: FPS rates comparison between algorithms and filters. Test conducted on a Orange PI

	With Saturation Remover	Without Saturation Remover	With Custom Background and Transparency Filter	With Transparency Filter	With Heatmap	Without Filters
ABL	6.01901	6.17106	7.59	7.61	10.26	11.5
ASBL	2.70356	2.72901	3.46	3.46	4.41	5.03
SD	5.09535	5.22297	5.95	9.53	8.22	9.14
SFD	10.4779	11.0537	8.11	8.14	17.42	18.71

VIBE	8.49478	8.84635	10.51	10.55	13.91	15.18
WMM	6.44796	6.57187	9.14	9.17	10.94	12.15

Table 13: FPS rates comparison between algorithms and filters. Test conducted on a Banana PI

	With Custom Background and Transparency Filter	**With Transparency Filter**	**With Heatmap**	**Without Filters**
ABL	88.48	88.48	90.27	87.23
ASBL	72.75	72.75	89.35	67.16
SD	86.41	83.53	88.74	83.53
SFD	87.22	84.41	88.73	84.31
VIBE	83.64	83.64	89.97	80.49
WMM	91.83	91.91	91.85	91.18

Table 14: PCC rates achieved from "Change Detection Challenge" data sets

	With Custom Background and Transparency Filter	**With Transparency Filter**	**With Heatmap**	**Without Filters**
ABL	97.48	97.48	98.84	95.14
ASBL	94.39	94.39	98.82	89.87
SD	97.16	93.33	98.55	93.33
SFD	98.84	97.23	98.9	97.23
VIBE	96.68	96.68	98.97	92.95
WMM	99.35	99.34	99.37	98.49

Table 15: PCC rates achieved from feeds containing cobwebs

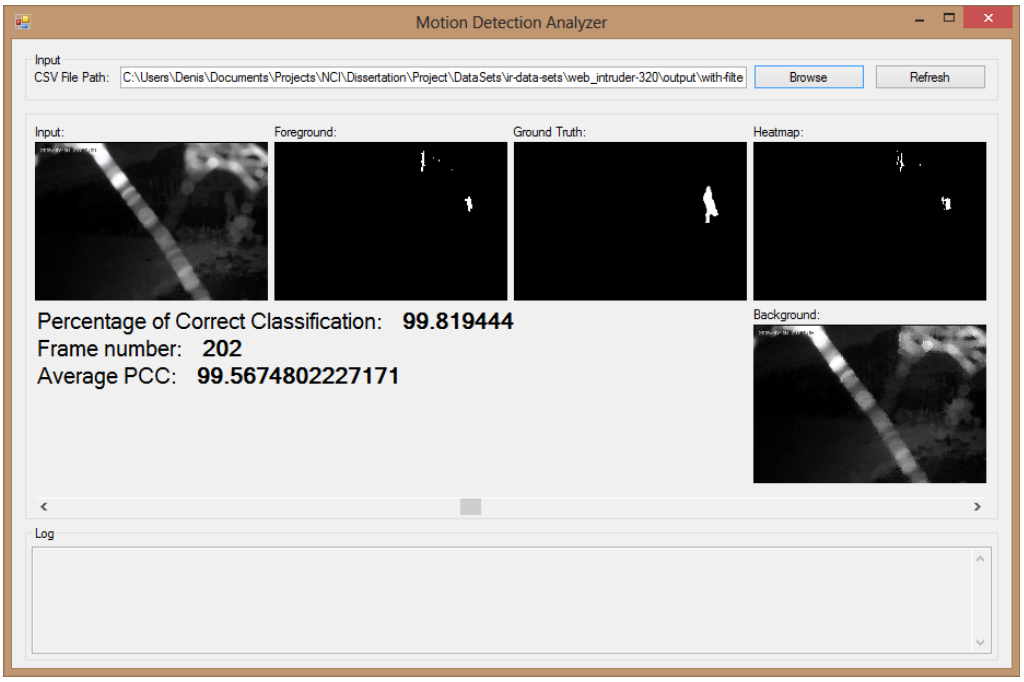
Appendix 2 – User guide

In order to start testing motion detection algorithms using the Segmentation Tester, navigate to the directory of the segmentation tester executable file and simply execute the following command:

```
ir-motion-filter md-algo VIBE work-width 320 show-gui true source "C:\bridgeEntry\input"
source-type sequence ground-source "C:\Users\bridgeEntry\groundtruth" enable-transp-
filter FFD 5 5 enable-heatmap-filter 2 5 enable-custom-background-model 5 2 enable-
saturation-remover 160 destination "C:\temp\output"
```

Once the executable finished running, the directory specified in “destination” argument, in this case it’s “C:\temp\output”, would contain input and output files generated by the segmentation tester. These can be inspected manually, or using the “Test Results Analysis Tool”.

In order to inspect the results using the results analysis tool, find and start the “MotionDetectionAnalyzer.exe” executable. Once started, the following screen would come up:



CSV File Path: C:\Users\Denis\Documents\Projects\NCI\Dissertation\Project\DataSets\ir-data-sets\web_intruder-320\output\with-filte

Percentage of Correct Classification: **99.819444**
Frame number: **202**
Average PCC: **99.5674802227171**

Click the “Browse” button and select the CSV file located in the directory as specified in the “destination” command line argument of the segmentation tester tool. This would load the metadata of the test and the states of the segmentation process. In order to fast-forward or rewind the video sequence, drag the horizontal scrolling slider.

Appendix 3 – Compilation Process

The following describes the processes for compiling the segmentation tester for different platforms, such as Windows, Linux and Linux running on ARM architecture.

Compilation on Windows

The compilation process on Windows environment is simple, but requires Visual Studio. To compile the application, open the solution in Visual Studio and press F6 or select Build Solution from the Build menu.

Compilation on Linux

The compilation process on Linux environments requires a number of development libraries to be installed first. The following commands were tested on Ubuntu and Rasbian operating systems. These commands configure the development environment and allow the compilation process to be performed on a remote machine, as compiling large amount of code on development boards is very slow.

The following commands must be executed on a development machine (e.g. Ubuntu)

```
#checkout toolchain for RPi
git clone https://github.com/raspberrypi/tools.git --depth=1
cd tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/
ln -s arm-linux-gnueabihf-gcc gcc
ln -s arm-linux-gnueabihf-g++ g++
ln -s arm-linux-gnueabihf-gcc cc
ln -s arm-linux-gnueabihf-c++ c++
ln -s arm-linux-gnueabihf-cpp cpp

export PATH=/your-path-to-tools/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-
raspbian/bin:$PATH


sudo apt-get install subversion autoconf automake python python-dev

#install libiberty, which is required for distcc
wget https://toolbox-of-eric.googlecode.com/files/libiberty.tar.gz
sudo tar -xvf libiberty.tar.gz
cd libiberty
sudo ./configure
sudo make
sudo make install

sudo apt-get install libgtk2.0-dev

cd ..

svn checkout http://distcc.googlecode.com/svn/trunk/ distcc-read-only
cd distcc-read-only
./autogen.sh
./configure --with-gtk && make && sudo make install

#start distcc daemon
# --jobs sets the number of jobs this machine can take in parallel
# --allow sets allows the machine with the given ip address to connect
# ( for more than one IP, call --allow again, i.e.: --allow 192.168.1.1 --allow 192.168.1.2 ...
# --log-stderr redirect the errors in the standard error output
# --no-detach don't detach from this terminal, so we can track the log in real time
distccd --daemon --jobs 4 --allow 192.168.178.25 --verbose --log-stderr --no-detach
```

The following commands must be executed on a development board (e.g. Raspberry PI)

```
sudo apt-get install subversion autoconf automake python python-dev

#install libiberty, which is required for distcc
wget https://toolbox-of-eric.googlecode.com/files/libiberty.tar.gz
sudo tar -xvf libiberty.tar.gz
cd libiberty
sudo ./configure
sudo make
sudo make install

sudo apt-get install libgtk2.0-dev

cd ..

svn checkout http://distcc.googlecode.com/svn/trunk/ distcc-read-only
cd distcc-read-only
./autogen.sh
./configure --with-gtk && make && sudo make install

sudo ln -s /usr/local/bin/distcc /usr/local/bin/gcc
sudo ln -s /usr/local/bin/distcc /usr/local/bin/cc
sudo ln -s /usr/local/bin/distcc /usr/local/bin/g++
sudo ln -s /usr/local/bin/distcc /usr/local/bin/c++
sudo ln -s /usr/local/bin/distcc /usr/local/bin/cpp
export PATH=/usr/local/bin:$PATH
```

```
#SETTING ENV VARS
# The remote machines that will build things for you. Don't put the ip of the Pi unless
# you want the Pi to take part to the build process.
# The syntax is : "IP_ADDRESS/NUMBER_OF_JOBS IP_ADDRESS/NUMBER_OF_JOBS" etc...
# The documentation states that your should set the number of jobs per machine to
# its number of processors. I advise you to set it to twice as much. See why in the test para-
graph.
# For example:
export DISTCC_HOSTS="192.168.178.31/2"

# When a job fails, distcc backs off the machine that failed for some time.
# We want distcc to retry immediately
export DISTCC_BACKOFF_PERIOD=0

# Time, in seconds, before distcc throws a DISTCC_IO_TIMEOUT error and tries to build the
file
# locally ( default hardcoded to 300 in version prior to 3.2 )
export DISTCC_IO_TIMEOUT=3000
# Don't try to build the file locally when a remote job failed
export DISTCC_SKIP_LOCAL_RETRY=1



git clone --depth=1 git://code.opencv.org/opencv.git
cd opencv
mkdir redist && cd redist
cmake -DCMAKE_BUILD_TYPE=DEBUG -DBUILD_JASPER=ON -DBUILD_OPENEXR=ON -
DBUILD_JPEG=ON -DBUILD_PNG=ON -DBUILD_TIFF=ON -DBUILD_ZLIB=ON -
DBUILD_DOCS=OFF -DBUILD_PERF_TESTS=OFF -DBUILD_TESTS=OFF -
DBUILD_opencv_gpu=OFF -DWITH_CUDA=OFF -DWITH_CUFFT=OFF -DWITH_OPENCL=OFF ..
&& time make -j4
time make -j4

#update the following file in order for the system to find openCV's libraries when executing
our program: /etc/ld.so.conf.d/opencv.conf
#this file should contain a line which points to the location of the *.so files.
#after directories have been added, run the following:
sudo ldconfig -v
```

This concludes the set-up on the environments. In order to compile the segmentation tester on either platform (Ubuntu or Raspbian) execute the following script:

```
~/your-path-to-sources/ir-motion-filter/ir-motion-filter/compile.sh
```

Appendix 4 – Testing Harness Command Line Arguments

This section describes the all of the available command-line options of the testing tool. It should be noted that the sub-parameter order is extremely important where more than one sub-parameter must be specified.

the argument values must follow the argument, casing important

- **source** - **Mandatory:** path to the video source, can be a video file, video stream, physical device name or a directory containing each frame as a separate file

  Example: source "c:\temp\video.avi"

- **source-type** - **Mandatory:** indicates whether the source path is a video file/stream or a directory

  Example: source-type sequence

  - ***sequence*** - used when source is a directory with a list of files
  - ***video*** - used when source is a video file or stream

- **ground-source** - path to a directory containing a list of frames that represent the ground truth

  Example: ground-source "c:\temp\ground-truth"

- **destination** - path to a directory where each input, output and ground truth frames, and the CSV log file would be written to

  Example: destination "c:\temp\dest"

- **show-gui** - when set to "true", each of the frames would be shown on a screen as they travel through each of the layers of the application. It should be noted that this configuration has no effect on the motion detection algorithms, which have similar configuration option in their XML files

  Example: show-gui true

- **work-width** - the width to which each of the frames would be resized to before processing. The height of the frame would be proportional to the width.

  Example: work-width 320

- **md-algo** - **Mandatory:** the motion detection algorithm to be used, only one must be specified

  Example: md-algo ABL

  - ***ABL*** **-** Adaptive Background Learning
  - ***ASBL*** **-** Adaptive Selective Background Learning
  - ***WMM*** - Weighted Moving Mean
  - ***SFD*** - Static Frame Difference
  - ***PBAS*** - Pixel-Based Adaptive Segmenter
  - ***SD*** - Sigma-Delta
  - ***VIBE*** - VIsual Background Extractor (ViBe)

- **enable-transp-filter** - enables transparency filter. A feature detector, the distance threshold and removal shape size must be specified.

  Example: enable-transp-filter FFD 5 5

  Feature Detectors: required, only one must be specified

  - ***FFD*** - Fast Feature Detector
  - ***GFTTD*** **-** Good Features To Track Detector
  - ***MFD*** - Mser Feature Detector
  - ***STRFD*** - Star Feature Detector
  - ***DFD*** - Dense Feature Detector
  - ***SBD*** **-** Simple Blob Detector
  - ***GAFD*** - Grid Adapted Feature Detector

  Distance Threshold: required, must be a whole number

  Removal Shape Size: required, must be a whole number.

- **enable-heatmap-filter** - enables the heat map filter; the heat map minimum and maximum values must be specified

  Example: enable-heatmap-filter 2 5

  Minimum: required, must be a whole number.

  Maximum: required, must be a whole number

- **enable-custom-background-model** - enables custom background model, requires two parameters: Pixel Update Step and Frame Update Step. If the first parameter is set to zero, every pixel would be updated, if set to 1, then every second pixel would be updated. The second parameter indicates how many frames should be skipped before the background model is updated, therefore, if zero is specified, then every frame would be used to update the background, alternatively, if 1 is specified, then every second frame would be used.

  Example: enable-custom-background-model 5 2

- **enable-saturation-remover** - enables saturation remover filter, requires one parameter to be specified: Max Brightness Value.

  Example: enable-saturation-remover 150

Sample command used in final tests:

```
ir-motion-filter md-algo VIBE work-width 320 show-gui true source "C:\bridgeEntry\input"
source-type sequence ground-source "C:\Users\bridgeEntry\groundtruth" enable-transp-
filter FFD 5 5 enable-heatmap-filter 2 5 enable-custom-background-model 5 2 enable-
saturation-remover 160 destination "C:\temp\output"
```

Appendix 5 – Segmentation Algorithms Configuration

The following XML is used to configure segmentation algorithms. The XML should be saved in a file named after the algorithm name, with extension ".xml", for example: "ViBe.xml". These XML files must be located in the same directory as the Segmentation Tester executable.

ViBe:

```
<?xml version="1.0"?>
<opencv_storage>
<matchingNumber>2.</matchingNumber>
<matchingThreshold>20.</matchingThreshold>
</opencv_storage>
```

Adaptive Background Learning

```
<?xml version="1.0"?>
<opencv_storage>
<alpha>5.0000000000000003e-002</alpha>
<limit>-1</limit>
<enableThreshold>1</enableThreshold>
<threshold>15</threshold>
<showForeground>0</showForeground>
<showBackground>0</showBackground>
</opencv_storage>
```

Adaptive Selective Background Learning

```
<?xml version="1.0"?>
<opencv_storage>
<learningFrames>90</learningFrames>
<alphaLearn>1.4999999999999999e-001</alphaLearn>
<alphaDetection>5.0000000000000003e-002</alphaDetection>
<threshold>5</threshold>
<showOutput>1</showOutput>
</opencv_storage>
```

Sigma-Delta

```
<?xml version="1.0"?>
<opencv_storage>
<ampFactor>1</ampFactor>
<minVar>15</minVar>
<maxVar>255</maxVar>
<showOutput>1</showOutput>
```

```
</opencv_storage>
```

Static Frame Difference

```
<?xml version="1.0"?>
<opencv_storage>
<enableThreshold>1</enableThreshold>
<threshold>15</threshold>
<showOutput>1</showOutput>
</opencv_storage>
```

Weighted Moving Mean

```
<?xml version="1.0"?>
<opencv_storage>
<enableWeight>1</enableWeight>
<enableThreshold>1</enableThreshold>
<threshold>15</threshold>
<showOutput>0</showOutput>
<showBackground>0</showBackground>
</opencv_storage>
```

Appendix 6 – CSV Log Sample

Input	Foreground	Background	Heatmap	GroundTruth	PCC
1-input.jpg	1-foreground.jpg	1-background.jpg	1-heatmap.jpg	1-ground.jpg	90
2-input.jpg	2-foreground.jpg	2-background.jpg	2-heatmap.jpg	2-ground.jpg	92
3-input.jpg	3-foreground.jpg	3-background.jpg	3-heatmap.jpg	3-ground.jpg	91