University of Southern Queensland

Faculty of Engineering & Surveying

An Intelligent Vision System For Wildlife Monitoring

A dissertation submitted by

Ashton Fagg

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Computer Systems)

Submitted: October, 2012

Abstract

The understanding of animal behaviour in response to human development is vital for sustainable management of ecosystems. Existing methods of monitoring wildlife activity fall short in facets pertaining to accuracy, accessibility, cost and practicality.

To address this level of crudity, recent technological advances have led to the development of electronic, autonomous wildlife monitoring solutions. Whilst these developments improve the overall experience in some areas, there is still much to be desired. This dissertation aims to outline the development of an accessible, affordable, intelligent vision-based technique which addresses limitations of existing monitoring methods.

A signal processing methodology was investigated, developed and implemented. The development of this methodology included the investigation of two distinct facets of computer vision - image segmentation and event classification.

The existing literature was explored, and several image segmentation techniques were explored. Upon further investigation, the Gaussian Mixtures Model was selected in two forms - per pixel modelling (Zivkovic 2004) and a compressive sensing based method (Shen, Hu, Yang, Wei & Chou 2012). Each method was evaluated in terms of real time capabilities and accuracy to provide basis for recommendation of the method presented in the prototype.

Upon evaluation, it was discovered that the proposed compressive sensing based method was a suitable prototype and recommendations regarding the implementation and commissioning of the system were made. Furthermore, possible avenues for further research and development were explored.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Ashton Fagg

0050101441

_____

Signature

_____

Date

Acknowledgments

It is a great pleasure to be given the opportunity to thank the wonderful people with whom I have been given the honour to work with on this project.

Firstly, I would like to acknowledge the support and advice of my two supervisors, Dr Navinda Kottege and Dr John Leis. This project and dissertation would not have been possible without their abundant knowledge, wisdom and support.

I must also acknowledge the invaluable assistance of:

- Dr Wen Hu (CSIRO)
- Mr Junbin Liu (CSIRO)
- Dr Darryl Jones (Griffith University)
- Dr John Griffiths (CSIRO)

On a personal note, I'd like to thank my friends and family for their invaluable support throughout my studies here at USQ and the process of this project. Last, but certainly not least, my thanks to Lucy - for her love, support, encouragement and endurance of my many hours of focus.

Ashton Fagg

University of Southern Queensland
October 2012

Contents

List of Figures

List of Tables

Nomenclature

GMM	Gaussian Mixture Model
CSGMM	Compressive Sensing GMM
FPU	Floating Point Unit
ROC	Receiver Operating Characteristic
VPN	Virtual Private Network
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative
MDF	Morphological Dilation Filter
MEF	Morphological Erosion Filter

Chapter 1

Vision Systems for Wildlife Monitoring

1.1 Introduction

At it's essence, computer vision aims to augment the magic of human perception with the autonomic nature of computation. The development of vision systems for use in applications which rely upon human cognition and judgement for event classification can have great advantages in efficiency and cost. Recent advancements in signal processing techniques and potential target hardware has further enhanced the attraction of vision systems as an effective platform for solution development.

For many centuries, scientists have been interested in the activities of wildlife within their natural environments. With increasing human development, the preservation of these habitats is an important challenge with respect to sustainable development practices and for the future of our ecosystem. One of the biggest threats to animal species is the development of roads which disturbs or segments their natural habitat. This can lead to an increase in road kill, and has the potential to inflict significant repercussions on the ecosystem. To allow animals to better cope with changes in their environment, the development of fixed animal crossing structures, such as specifically purposed road underpasses, has presented new challenges in research areas targeting

the understanding of the impact of human development upon fauna.

The use of fixed-structure wildlife crossings has been well discussed in previous work. Such examples have taken the form of wildlife bridges (Bond & Jones 2008) and tunnelling systems (Mata, Hervs, Herranz, Surez & Malo 2008) which provide a safe path to surrounding areas and minimising the impact upon the ecosystem. The use of these structures has formed a basis for the preservation of wildlife in newly developed areas where a risk to the wildlife is involved (i.e. risk of injury or death due to contact with traffic). In order to monitor the effectiveness of the implementations, road kill surveying (Mata et al. 2008) (Bond & Jones 2008) in conjunction with activity monitoring within the structures is employed to gauge an overview of the approximate usage statistics. The road kill surveys performed provide a method of determining whether the development of such structures has an effect in reducing the number of animals which are killed or injured due to contact with traffic.

Figure 1.1: Wildlife crossing tunnels developed at Gap Creek Road, Pullenvale. Source: Ashton Fagg

The primary objective of wildlife monitoring is the collection of data by means which minimise disturbances to the surrounding environment. Non-autonomous methods have been used exclusively for many years. In the context of fixed-structure wildlife crossings, several methods are preferred. The use of a porous substrate, such as sand (Bond & Jones 2008) or finely grained marble (Mata et al. 2008) to record footprints of species

as they transition is a typical method. Other methods include the use of scat tracking (Bond & Jones 2008) and hair sampling (Harris & Nicol 2010).

The methods proposed offers a limited level of accuracy in regards to quantitative and categorical surveying. Whilst a broad overview of the species utilising the wildlife crossings is available, it does not guarantee that the evidence of a present species will be recorded or preserved. Nor does such a method provide a definite quantity of the number of inhabitants of the area in question.

Perhaps the most obvious method which addresses these shortcomings is a manual survey, which is high in cost and tedious. However, electronic monitoring methods (such as small, low power cameras) are beginning to make an entrance into the field in coexistence with "tried and true" methods such as footprint recording and hair trapping.

The use of an electronic system aims to improve accuracy and reliability whilst still alleviating the tediousness of the task by autonomously performing the survey with little human interaction. Current electronic systems are not capable of distinguishing between different types of motion, and can often record events which are not of interest to the survey at hand. In order to keep monetary and power costs low, hardware components are often not fast enough to adequately record fast-moving objects.

Thus, it is clear that the existing methods of animal monitoring and wildlife detection may not be ideal or optimised for comparable performance with a manual survey. It is believed that the development of an electronic, vision-based technique can provide significant performance increases over existing methods. This project aims to explore suitable methods and make recommendations for a prototypical vision system concept for the purposes of autonomous & unsupervised wildlife monitoring. These limitations will be investigated in detail in later chapters.

1.2 Project Aim

The primary aim of this project is design, develop and implement of an unsupervised, intelligent, prototypical software-based vision system which addresses the limitations of existing wildlife monitoring systems. The project will involve the creation of image processing software. The software system must also provide means for recording the captured frames in a way which is easily accessible to the user. In the interests of user accessibility, the system must be fully autonomous in operation once it has been configured. The system must be capable of being deployed to relevant locations and be able to work at a rate which is equal to or faster than real time.

This project aims to investigate, compare and recommend existing image processing techniques and algorithms for use in this application. These techniques will involve the development of test software and the evaluation and validation of algorithms by way of existing ground truths, and the performance evaluated in terms of speed and accuracy.

Upon completion of the project, it is intended that the recommendations and prototypical software can be further developed into a production system for installation upon fixed wildlife crossing structures for the generation of knowledge and research in ecological and biological fields.

1.3 Project Objectives

The project aim was assessed and broken into a number of deliverables for completion.

- Identification of limitations with existing monitoring methods.
- Investigation of image processing methods which address these shortcomings.
- Development of software prototypes which allow the performance of methods to be compared and recommendations made for the prototype.
- Recommendations for supporting hardware & software systems.
- Analysis and evaluation of the system performance.

1.4 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 discusses the target application and existing solutions to the target problem. Furthermore, shortcomings with existing systems are identified and defined as a basis for development of the new system. Image and signal processing techniques are discussed and evaluated in terms of suitability for the application.

Chapter 3 defines suitable methodologies for research and software development. This chapter also includes a risk and hazard identification and classification and appropriate mitigation strategies. Resources which are required for this project are also identified and their attainability assessed.

Chapter 4 details the design, development and implementation of the software system. This includes details of processing algorithms and the details of each system component.

Chapter 5 identifies performance metrics, test and validation strategies used to qualify the system. The outcomes of these tests were used to recommend the prototypical design for submission.

Chapter 6 summarises the work performed and identifies further avenues for research and development.

Chapter 2

Literature Review

2.1 Chapter Overview

It is only in more recent times that electronic means have been used with this type of application in mind. With technological advances made over the last couple of decades, the implementation of a computer-based system to replace existing methods has developed into a feasible, affordable and sensible option.

This chapters examines existing methods of wildlife monitoring, and their underlying aims. The feasibility of an electronic system to perform this task is examined, and existing literature which targets specific implementation challenges put forth by this domain was analysed. This includes the exploration of image processing techniques which aid in the extraction of meaningful data.

2.2 Wildlife Monitoring

As evidenced in previous work (Bond & Jones 2008), temporal trends in animal activity can be significant to the research at hand. The use of methods outlined in Section 1.1 cannot provide adequate temporal data of sufficient granularity. Thus, it was necessary to examine current solutions which offer the ability to place events on a temporal scale.

2.2.1 Existing Autonomous Methods

The use of a small, low power wireless sensor network to monitor animal activity has been proposed in a number of scenarios. Solutions used in previous studies have relied upon the use of Passive Infra-red Detectors (PID). Such a system consists of small, low power micro-controller boards equipped with a low power radio transceiver. The individual sensor nodes communicate with a base node, which is typically acting as a static data logger (Langbein, Scheibe, Eichhorn, Lindner & Streich 1996) or as a gateway to an external storage medium (Mainwaring, Culler, Polastre, Szewczyk & Anderson 2002). Additionally, environmental sensors may be included on the nodes to provide a broader level of environmental awareness (Mainwaring et al. 2002).

The use of a sensor network, and the ideals behind remote monitoring via single point data aggregation (Mainwaring et al. 2002) was considered to be a basis of design for the vision system. However, the use of a PID for motion detection and data collection does not offer the ability to detect the species, merely providing a quantitative count.

Some systems address this issue with the inclusion of a camera. The PID is used as a trigger to signal the camera to record a still image (Brown & Gehrt 2009). The cameras included in such systems are low power and infra-red sensitive to enable the use of the equipment for monitoring of nocturnal species. The use of a trigger method allows for temporally dynamic events to be captured in entirety. However such systems are considered passive in that the camera is placed in a low power state, essentially limiting the response time of the system and potentially allowing for missed events. The rate at which the PID is polled can also have an effect on accuracy. The poll rate provides a delicate balance between power saving and accuracy and incorrect sample rate decisions can result in missed events or inefficient use of observation time (Langbein et al. 1996).

The reliance upon a PID presents a number of issues of reliability. Sources of infrared radiation can pose interference risks, and subsequently can cause large numbers of spurious triggers. For example, dense vegetation is able to trap heat, and subsequently can present as a stimulus of the PID. Thus, a method which is able to minimise these spurious events was considered highly advantageous, as less data is generated requiring manual classification. Thus, the use of a PID as a primary means of event detection

was discounted for the vision system. (Mainwaring et al. 2002)

Alternatively, camera based system have been used to take a still image at a given interval (Ganick & Little 2009) and are therefore unaware of sources of motion. Such systems are capable of generating large amounts of data, and require human intervention to classify data as relevant. The system devised in (Ganick & Little 2009) presents an almost real-time data retrieval and remote monitoring method which utilises an Internet connection to deposit the retrieved data to a remote server, which is subsequently viewable by the user in a Web Browser. This is similar to the method discussed in (Mainwaring et al. 2002) as it provides an accessible means of data retrieval and monitoring for use in remote deployment locations. However, assuming that wildlife events are temporally sparse, the generation of large amounts of data is undesirable due to factors of bandwidth, storage and the labor requirements to review the collected data.

2.2.2 Vision System Requirements

With these findings in mind, it was defined that a suitable vision system was to be able to cope with sparse animal activity across a wide range of environmental and application conditions. The vision system was to be autonomous, unsupervised and require minimal human intervention. Thus, a vision system which encompasses a static camera and the hardware and software support to process the incoming frames was required. The system was to only return data which is relevant and meaningful, and subsequently required the ability to make an "intelligent" decision with respect to the validity of the event. Thus, the requirement for an intelligent, autonomous and unsupervised vision system was defined.

The advantage of an unsupervised, autonomous vision system is the facility to return data in near real-time and the remote monitoring and configuration abilities. This was considered to be an improvement over existing systems and methods in many facets.

2.3 Signal Processing for Wildlife Monitoring

With the need for a vision system identified, it was necessary to examine the existing literature pertaining to suitable signal processing techniques.

To operate within a natural environment, the system was to be tolerant of a wide variety of operational circumstances. The nature of this operation can be difficult to define, as it is imperative that the method of operation makes as few assumptions about the surrounding environment as possible. The lack of assumptions which can be made allows for a more tolerant system, as the signal processing techniques must be independent of any external stimuli. Thus, the system was to be designed in such a way which made little assumption about the operational conditions it would face.

The natural environment presents several distinct challenges for many vision-based systems. Perhaps the most important is the variability of illumination. Not only must the system be able to operate successfully, independent of a guaranteed degree of illumination, the system must also cope with the natural progression of lighting throughout the day. Additionally, the natural environment offers a degree of image noise which can corrupt incoming samples and create difficulty for processing methods. The most detrimental noise to the purposes of this application, presents as spurious motion caused by environmental factors (kinetic noise). A simple and relevant example of kinetic noise is vegetation in motion due to wind. Whilst this is motion within the frame, it is not a valid event. A vision system for wildlife monitoring relies upon the introduction of new entities to the frame. With these considerations in mind, the following aspects were identified for investigation:

1. Techniques which allowed the vision system to classify segments of the image in terms of their degree of observational note.
2. Detection of motion and classification as relevant or irrelevant.
3. The introduction of a degree of tolerance to moderate amounts of kinetic noise.

2.3.1 Image Segmentation

Image segmentation is the process of classifying parts of a digital image into segments which represent significant components of an image. This may take the form of a foreground-background segmentation or the identification of continuous regions which have a common similarity. In essence, this process assigns each pixel component to a set which share certain visual characteristics in order to create a representation of an image which separates different classes of pixels.

Perhaps the most obvious method of subtraction is the use of a simple frame differencing method. By taking a static image of the surroundings as a frame of reference and calculating the pixel-wise difference between incoming frames, a crude form of background subtraction is established. (Cheung & Kamath 2004) Frame differencing assumes that the background of the frame remains constant over time. Obviously, frame differencing is not a robust method of background subtraction for an application in an environment with a dynamic background, and it was subsequently discounted as a suitable method. (Cheung & Kamath 2004)

Frame differencing does not take into account factors posed by natural environments - for example, the gradual changes in natural lighting which occur throughout the progression of the day. (Cheung & Kamath 2004) Additionally, frame differencing assumes that all changes within the frame are of interest. Whilst this may be appropriate for some applications, kinetic noise is not significant in the context of wildlife monitoring. This also applies to noise generated by the camera sensors, which in some circumstances which can be quite significant. Even if the static reference is updated for every new session, there is significant potential for incorrect assumptions about the segmentation of the image to be made. Thus, a more sophisticated method was required to cope with varying lighting conditions and kinetic noise. (Cheung & Kamath 2004)

Many background subtraction methods rely upon the use of a histogram shaping mechanism to perform the pixel classification. One such method is the method proposed by Otsu (Otsu 1979), which suggests that a planar grayscale image can be reduced to a binary image given a threshold of intensity to classify each pixel. The Otsu thresholding method suggests that this threshold be calculated in such a way that the intraclass

variance is minimised. (Otsu 1979) These parameters can be calculated iteratively and a minimisation expectation applied. (Otsu 1979)

To exhaustively minimise the intraclass variance, the weighted sum of variances of the two classes can be represented according to:

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{2.1}$$

ω_i represents the weight of each mode, separated according to the threshold t and the mode variances represented by σ_i^2. (Otsu 1979)

As such, the minimisation of the intraclass variance is the equivalent of maximising the interclass variance:

$$\sigma_b^2(t) = \sigma^2 + \sigma_w^2)(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2 \tag{2.2}$$

The mean of each class is notated as μ_i. The class probability can be calculated from examining the histogram according to:

$$\omega_1 = \Sigma_0^t p(i) \tag{2.3}$$

The class mean is derived from:

$$\mu_1(t) = \Sigma_0^t p(i)x(i) \tag{2.4}$$

Where $x(i)$ is the center of the i^{th} histogram bin. The other class probability and mean can be calculated in converse, by utilising the right side of the histogram starting with bins which are greater than t.

The threshold is determined by way of an exhaustive search minimisation. (Otsu 1979) The image is then separated into a bimodal histogram which allows for the output

image to be used as a binary mask. Most commonly, the mask is represented by a binary image - foreground as white, background as black.

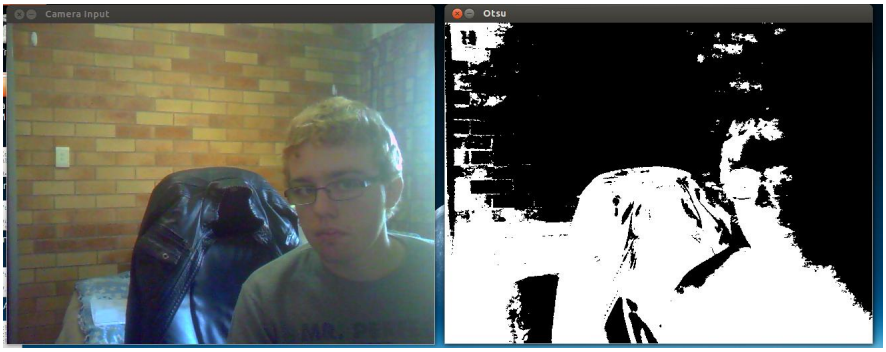

Figure 2.1: An example of a segmentation mask calculated using Otsu thresholding

Whilst the Otsu method yields reasonable results, the classification accuracy is not robust for applications which stipulate that the previous inputs to the system form the basis of ground truths for following frames. The use of a single frame of reference is a naive approach to image segmentation as the changes in the scene do not form a historical basis for segmentation of future frames. This creates difficulty in setting boundary conditions and thresholds in order to define rules which govern the detection of events. Additionally, regions of noise are not effectively eliminated as the use of only a small number of distributions for the entire image does not facilitate adequate segmentation depth. The utilisation of an iterative model, which places lifetime constraints of the input, allows previous inputs be used in determining the classification of the pixel components. Additionally, by segmenting the image according to the entire frame, localised image features are able to affect the segmentation as a whole. This was not considered to be desirable, as anomalies are required to be localised. In order to localise anomalies, the segmentation model must encompass many sub-models which segments the image according to local features, not the features of the neighbouring regions.

The Effect of Locality

With these limitations in mind, it was noted that methods which rely solely upon the current image did not present a suitable method of segmentation for use with this application. As events in the foreground are assumed to be sparse, a method which does

not rely upon absolute or "forced" segmentation was investigated. Clustering-based methods assign pixel components to distinct clusters for which the distribution can be modelled through the formulation of fitted distribution. Typical methods involving the clustering do not assume that the image will always contain foreground. Models which are mature and correctly configured will allow for greater accuracy.

The use of a mixtures model allows that the distribution of these clusters be modelled according to a weighted mixture of components which are updated upon each new sample to form an adaptive threshold for image segmentation. However, in the case where events are considered local,, the use of a mixture of only a few components for the entire image yields many of the same issues as the histogram based methods discussed previously.

A localised method of mixture modelling calls for the image to be divided into a number of regions. These regions can be as a small as a single pixel, or many encompass many pixels. Modelling each of the regions as a separate mixture of Gaussian components allows for local history to be preserved resulting in more accurate and robust segmentation. By defining the local segmentation in terms of the history of local features, this enables each part of the image to be considered independent from all others. This enables the model to cope with gradual changes, and changes will not have bearing upon any other locale.

The Gaussian Mixture Model

The Gaussian Mixture Model (GMM) provides a clustering means of estimating the background and foreground distributions through a sum of weighted Gaussian distributions. In essence, it provides a parametric probability density function describing the distribution of pixel values against background and foreground. (Reynolds 2008)

The GMM can be described by the equation:

$$p(x|\lambda) = \sum_{i=1}^{M} w_i g(x|\mu_i, \Sigma_i) \tag{2.5}$$

Where,

- x is a D-dimensional data vector (in this case, an image)
- The mixture weights are represented by w_i, where $i = 1, \ldots, M$
- $g(x|\mu_i, \Sigma_i)$ represent the Gaussian component densities. Again, $i = 1, \ldots, M$.

(Reynolds 2008)

The Gaussian component densities are represented by a D-variate Gaussian function. This function takes the form of:

$$g(x|\mu_i, \Sigma_i) = \frac{\exp\{-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)\}}{(2\pi)^{D/2}\sqrt{(|\Sigma_i|)}} \tag{2.6}$$

Hence,

- μ_i is the mean vector, representing the mean for each component.
- Σ_i is the covariance vector

The mixture weights must satisfy the condition that their sum must add to one (i.e. $\sum_{i=1}^{M} w_i = 1$). (Reynolds 2008)

The use of a GMM for background subtraction has been widely discussed. An implementation of a GMM with fixed parameters only has a limited scope of application (Stauffer & Grimson 1999). A common application of the GMM, is to model the base image (the background) and compare the error of incoming images with the established model according to a fixed threshold across a fixed number of components. (Stauffer & Grimson 1999). As changes in the background occur (such as a change in the intensity or direction of shadows), a fixed parameter model will accumulate an error and potentially allow for spurious event indicators (Stauffer & Grimson 1999).

In order for a system to be robust and not rely upon the user to follow a set of guidelines for its placement or make assumptions regarding the visual parameters of the

environment, the system should be able to adapt to it's surroundings. (Stauffer & Grimson 1999). Variations of the GMM make use of adaptive parameter settings (Stauffer & Grimson 1999) (Zivkovic 2004) (Dar-Shyang 2005) to adjust to gradual scene changes in order to successfully model the background under varying lighting conditions. Zivkovic (Zivkovic 2004) suggests a recursive method to adaptively select the appropriate number of Gaussian components for each pixel. This method differs from more common methods (Stauffer & Grimson 1999) as there is no fixed number of Gaussian components. This employs the locality principle discussed in the previous section, as all parts of the image are considered to be independent and uncorrelated.

To explain the implementation of the adaptive GMM in the algorithmic structure proposed by Zivkovic, we assume that the value of a pixel at time t is denoted by $x^{\vec{(t)}}$, the pixel classification as foreground or background can be described as (Zivkovic & van der Heijden 2006):

$$\frac{p(BG|x^{\vec{(t)}})}{p(FG|x^{\vec{(t)}})} = \frac{p(BG)p(BG|x^{\vec{(t)}})}{p(FG)p(FG|x^{\vec{(t)}})} \tag{2.7}$$

Assumptions surrounding the presentation of foreground objects cannot be made as there is no information regarding how they are be presented, or when they will appear. This concept is especially important in cases where relevant motion does not occur on a spatially or temporally consistent basis. As such, a uniform distribution is applied to the appearance of foreground objects. Thus, the classification decision can be described as (Zivkovic & van der Heijden 2006):

$$p(x^{\vec{(t)}}|BG) > c_{thr} \tag{2.8}$$

c_{thr} is the component threshold value and is described by:

$$c_{thr} = \frac{p(FG)p(FG|x^{\vec{(t)}})}{p(BG)} \tag{2.9}$$

(Zivkovic & van der Heijden 2006)

The background model, $p(BG|\vec{x^{(t)}})$, is estimated according to a training set, χ (Zivkovic & van der Heijden 2006). In the context of a system which processes data in or near real-time, the training set refers to a stream of incoming frames (such as from a camera or pre-recorded sequence). The training set is used to update the estimated model and in turn to classify pixels as foreground or background. In practice, assumptions cannot be made about the stability of the captured scene. It is assumed that the sample inputs are independent, and that adjacent pixels are uncorrelated. (Zivkovic & van der Heijden 2006). Realistically, consideration must given to the fact that objects could be removed from the background or placed in the scene to become part of the background. Thus, to ensure that the model eventually adapts to these changes, the training set samples must have a limited lifespan. This is performed by discarding samples which are older than a given interval (Zivkovic & van der Heijden 2006). This time interval is determined by the adaptation period, T. The learning rate is therefore, $\alpha = 1/T$. (Zivkovic & van der Heijden 2006)

If we denote the overall model estimation as, $\hat{p}(\vec{x}|\chi_\tau, BG + FG)$, where at time t, $\chi_\tau = \{x^{(t)}, \ldots, x^{(t-T)}\}$, referring to equation 1, an M component GMM can be represented by:

$$\hat{p}(\vec{x}|\chi_\tau, BG + FG) = \sum_{m=1}^{M} w_i g(\vec{x}, \vec{\mu_m}, \vec{\sigma_m}^2 I) \tag{2.10}$$

Where,

- $\vec{\mu_1}, \ldots, \vec{\mu_m}$ are the estimated pixel means
- $\vec{\sigma_1}, \ldots, \vec{\sigma_m}$ are the estimated variances of the Gaussian components
- I is the identity matrix

According to Titterington (Titterington 1984), the model can be updated recursively. Given a new set of sample data, $\vec{x}^{(t)}$, the components weights, pixel means and variances can be updated according to:

$$w_m = w_m + \alpha(o_m^{(t)} - w_m) \tag{2.11}$$

$$\hat{\mu_m} = \hat{\mu_m} + o_m^{(t)}(\alpha/w_m)\vec{\delta_m} \tag{2.12}$$

$$\vec{\sigma_m}^2 = \vec{\sigma_m}^2 + o_m^{(t)}(\alpha/w_m)(\vec{\delta_m}^\tau \vec{\delta_m} - \vec{\sigma_m}^2) \tag{2.13}$$

Where,

$$\vec{\delta_m} = \vec{x}^{(t)} - \vec{\mu_m} \tag{2.14}$$

In the context of Titterington's equations, α denotes an exponentially decaying window used as a weighting for the data samples. (Titterington 1984) As discussed previously, the learning rate is defined according to the adaptation period, and thus α in this context can be assumed to have the same effect.

The $\vec{o_m}$ terms represent the ownership of a sample by an existing Gaussian component. For new samples, this is initialised to 1 for the closest component with the most weight (i.e largest value of w_m). All other components are set to 0. (Zivkovic & van der Heijden 2006).

To determine the ownership, the squared distance is calculated in terms of the m^{th} component by (Zivkovic & van der Heijden 2006):

$$D_m^2(\vec{x}^{(t)}) = \frac{\vec{\delta_m}^\tau \vec{\delta_m}}{\vec{\sigma_m}^2} \tag{2.15}$$

The definition of close is determined by comparison against an arbitrary threshold. (Zivkovic & van der Heijden 2006) If a component is not found within the bounds of such a threshold, it is not considered close. In such cases a new component must be initialised. The parameters of such a component are initialised according to the model settings. A typical configuration is as follows:

- $w_{M+1} = \alpha$
- $\vec{\mu_{M+1}} = \vec{x}^{(t)}$
- $\vec{\sigma_{M+1}} = \sigma_0$, where σ_0 is an appropriate value for initial variance.

(Zivkovic & van der Heijden 2006)

If the maximum number of components has been reached, the component with the least weight (the smallest value of w_m) is discarded.

The number of components can be determined according to the adjusted update equation (Zivkovic & van der Heijden 2006)

$$w_m = w_m + \alpha(o_m^{(t)} - w_m) - \alpha c_\tau \tag{2.16}$$

If the GMM is initialised with a single component centred on the first sample, and new components are added in accordance with the criterion defined by equation 2.11, the component m is discarded when the weight, w_m becomes negative. In essence, components which are not supported by the incoming data are removed, and components which are supported are added. (Zivkovic & van der Heijden 2006). The value of c_τ is used to discriminate against components which are not supported. For a given adaptation period, a suitable interval to determine support levels is required. Assuming an adaptation period of T, the level of support required for a component will be determined against a given value of c_τ. This will be an arbitrarily defined threshold depending upon the level of sensitivity which is required. A smaller value of c_τ dictates that an entity must be consistently present for a longer interval in order to register as a valid component. Likewise, larger values of c_τ will depreciate the weighting of the component and cause it to be discarded faster. (Zivkovic & van der Heijden 2006).

The use of the method proposed by (Zivkovic & van der Heijden 2006) ensures that motion which adheres to some level of periodicity (i.e. breeze causing leaves to move) or motion which is not deemed to be in the foreground of the image is eliminated from the field of interest. This also allows for the background to change in situ, as

elements which were previously registered as foreground will eventually be transformed to a background component as observation time increases. This supports the need for a locality aware segmentation method.

In much the same manner as the Otsu method, the output matrix can be used to eliminate pixels which are not of observational interest.

Computational Performance Considerations

The method proposed in (Zivkovic 2004) offers the potential for a high level of accuracy at the cost of a comparatively high computational complexity. On embedded devices with low computational power, the use of this method could pose a significant performance threat.

The reliance upon floating point arithmetic must also be taken into consideration. A hardware floating point unit (FPU) would need to be present on the system, as emulated floating point calculations will further degrade performance.

Compressive Sensing for Gaussian Mixture Models

The application of compressive sensing to background subtraction (Shen et al. 2012) offers the potential for a significant performance increase whilst not limiting the accuracy of the model output. The method proposed by (Shen et al. 2012) draws upon similar concepts to that proposed in (Zivkovic 2004), again by modelling locales as individual mixtures model. The compressive sensing is applied by modelling each pixel in terms of compressible vectors. These compressible vectors allow the reconstruction of a dense vector from the sparse projection vector. By modelling the projection vectors, the number of Gaussian components is reduced significantly. (Shen et al. 2012)

For compression purposes, let ϕ equal a ± 1 Bernoulli matrix. (Shen et al. 2012) The probability of the values of the matrix is computed using a symmetric Bernoulli distribution, bounded between ± 1. For optimal performance, the matrix must be balanced in that the sum of each row in the matrix will be equal to zero.(Shen et al. 2012) The

dimensions of ϕ will be dependant upon the size of the compressible vectors, and the number of projections which are to be modelled.

The image need then be divided into $N \times N$ blocks. In order to model each of the components as M projections, ϕ needs to be sized accordingly. The recommendations put forward in (Shen et al. 2012) suggest a suitable value of M and N to be 8, as using a block size of 64 pixels and modelling to 8 projections allows for an even balance between performance and accuracy. To form compressible vectors, each of the blocks is subsequently vectorised to form a $N^2 \times 1$ matrix.

The compression step involves a reduction in the number of components contained within the modelled block. This is achieved by way of a simple matrix multiplication between the compressible vector and ϕ. Note that the same ϕ is used for all blocks in the image. Thus, if y is a projection vector, and x is a compressible vector:

$$y = \phi \times x \tag{2.17}$$

y will be an $M \times 1$ projection vector. The projections are then modelled using a mixture of Gaussian distributions which greatly mimics the way in which it was applied in (Zivkovic 2004). Given that x is compressible, given a constant value of ϕ and an appropriate value of M, it is apparent that the value in y contains almost all of the same information in x. (Shen et al. 2012) Thus, the projection vectors can be used to determine which parts of the image are foreground.

The speed increase is achieved by the fact that number of components which are represented by the mixture of Gaussians is much less. Considering a traditional model, consisting of three Gaussian components per pixel for a block of 64 pixels. This equates to 192 Gaussian components. Using compressive sensing, the 64 pixel block is compressed to a 8 projections. At 3 Gaussian components per projection, this equates to a mere 24 total Gaussian components. The factor of eight reduction offers a great speed advantage over traditional models. (Shen et al. 2012) The use of synthetic test sets suggests that the addition of the compressive sensing offers a speed increase of 5 - 7 times. (Shen et al. 2012) When comparing the performance of a traditional Gaussian Mixtures

model to the compressive sensing method, an image of 320 × 240 passes through the compressive sensing method in 56.6 ms, as opposed to the previous method in 280.88 ms. (Shen et al. 2012)

GMM Suitability

The Gaussian Mixtures Model presents a potentially suitable method of classifying locales within the image. Hence, the Gaussian Mixtures Model was recommended as the image segmentation method for incorporation into the design of the vision system.

The non-compressive and compressive sensing GMM were considered for suitability. It was necessary to evaluate and compare the performance both techniques. The methodology and outcomes surrounding these tests will be discussed in a later section.

2.3.2 Motion Detection Using Optical Flow

The use of a background subtraction technique allows the system to be ignorant of any parts of the image which are not deemed of observational importance. However those parts which are deemed to be important must be observed for changes which indicate an event has occurred.

Optical flow mathematically models the apparent motion of elements within an image, in terms of relative motion between frames in a sequence of images. In essence, it is a form of motion detection which aims to model the movements of an image in terms of motion and velocity vectors. In some cases, this can be used to model and track the motion of objects within the frame. For the purposes of this system, optical flow will only be used to detect the presence of objects which are moving, and any information collected regarding their trajectories will be disregarded. Thus, a method of optical flow which localises motion to defined features in not suitable. Hence, a method assumes that the features to be tracked are present in the frame a priori (sparse optical flow) is not suitable for this application. Dense optical flow estimates the motion of the entire frame, based upon the difference between sequential frames.

The Farneback Method of Optical Flow

The Farneback Method of Optical Flow provides an estimation of dense optical flow based on polynomial expansion. (Farneback 2003) For the purposes of explanation, the polynomial which models the pixel neighbourhoods are assumed to be a simple quadratic. (Farneback 2003) If A is a symmetric matrix, b is a vector and c is a scalar, the local signal model can be expressed as:

$$f(x) = x^T Ax + b^T x + c \tag{2.18}$$

(Farneback 2003)

By defining two frames, $f_1(x)$ and $f_2(x)$, $f_2(x)$ is able to be represented as a global displacement d of $f_1(x)$. (Farneback 2003)

As such:

$$f_2(x) = f_1(x + d) \tag{2.19}$$

Thus,

$$f_2(x) = x^T A_1 x + (b_1 + 2A_1 d)^T x + d^T A_1 d + b_1^T + c_1 \tag{2.20}$$

This can be simplified by equating the polynomial coefficients to incorporate the previous frame's coefficients:

$$A_2 = A_1 \tag{2.21}$$

$$b_2 = b_1 + 2A_1 d \tag{2.22}$$

$$c_2 = d^T A_1 d + b_1^T + c_1 \tag{2.23}$$

(Farneback 2003)

The global displacement, d is able to be calculated provided that at least the matrix A_1 is non-singular. This observation will hold true for any level of signal dimensionality. (Farneback 2003) As such:

$$2A_1 d = (b_1 + b_2) \tag{2.24}$$

$$d = \frac{1}{2} \times A_1^{-1}(b_1 + b_2) \tag{2.25}$$

Each of the pixel neighbourhoods is thus tied to a motion constraint between the two frames. (Farneback 2003) This would be especially useful should motion need to be predicted, however for this project the use of an optical flow method was investigated for the purposes of motion detection in relevant parts of the frame. Hence, motion is present within the frame if motion vector components with a magnitude other than zero is detected.

In terms of the vision system, the details of the optical flow method itself were not considered to be important, as there is no need for a high level of accuracy. The Farneback optical flow method was considered to be adequate and was recommended for use in the prototype.

2.4 Chapter Summary

In this section, the literature surrounding current wildlife monitoring methods has been identified. The shortcomings identified in current methods are a prime candidate for improvement by the development of a suitable vision system. The need for an accurate and reliable means of data collection can be greatly supplemented by the use of an embedded system. Such a system must provide a higher level of accuracy in species counting and tracking, and provide near-real time data return whilst still remaining cost effective and minimising the impact on habitat and the environment.

The proposed camera system will include the use of a background subtraction method. The Gaussian Mixtures Model provides an effective means of localised segmentation and masking the areas which are not of observational note. The methods proposed by

(Zivkovic 2004) and (Shen et al. 2012) are to be compared for suitability in terms of performance and accuracy when utilising an embedded system.

When combined with an optical flow method, motion in the relevant sections of the image can be detected. This project will investigate the method proposed in (Farneback 2003) for the purposes of event detection.

Chapter 3

Methodology

3.1 Chapter Overview

This chapter covers the research and development methodology utilised for the development of the vision system and the evaluation of it's performance. This chapter also details risks and hazards and suggests suitable mitigation strategies for their minimisation. Any resources which are required for the completion of each of the stages of development are outlined in this chapter.

3.2 Research and Development Methodology

The research and development methodology contained herein was developed with an understanding of how the deliverables would be achieved. This led to a logical breakdown of major tasks.

The development of the vision system called for the design and production of an unsupervised and autonomous system which was able to process frames in real time. The frame process was to be able to distinguish relevant motion from sample frames in order to minimise spurious captures. The recording state was to be determined on a per sample basis and be efficient in implementation so as to allow operation in real time

on an embedded device.

The images recorded by the system were to be stored in static storage and be accessible remotely. Thus, the project called for the development and implementation of a number of distinct *layers*. These layers formed the basis of the system and encompassed both hardware and software components.

At the hardware level, the system required a camera to capture the images and a suitable interface to retrieve the images for processing. The camera was to be connected to a suitable computer device with suitable support software for resource management, networking and device control.

The hardware layer was to be supplemented by a software platform which implemented a retrieval and buffering method. This layer was designed to feed frames from the camera to the processing software. Finally, the software system was be able to be "triggered" and store the recorded images in static storage. As such, several distinct layers were identified:

- Camera control and buffering layer.
- Frame processing and triggering layer.
- Recording and storage layer.

The interactions between the software and the hardware layers can be illustrated by the block diagram shown in Figure 3.1.

The evaluation of performance and validation of requirements required the definition of performance metrics and the design of suitable testing methodologies and analysis strategies. The outcomes of this step formed the recommendations for further enhancements and suggestions of future research.

Due to the modularity of the project, an incremental development methodology was adopted. This placed an emphasis on "test first" ideals, and encourages a more modular design with decreased dependencies. The modularity of the data structures and algorithm implementations encouraged code reuse throughout the software base.

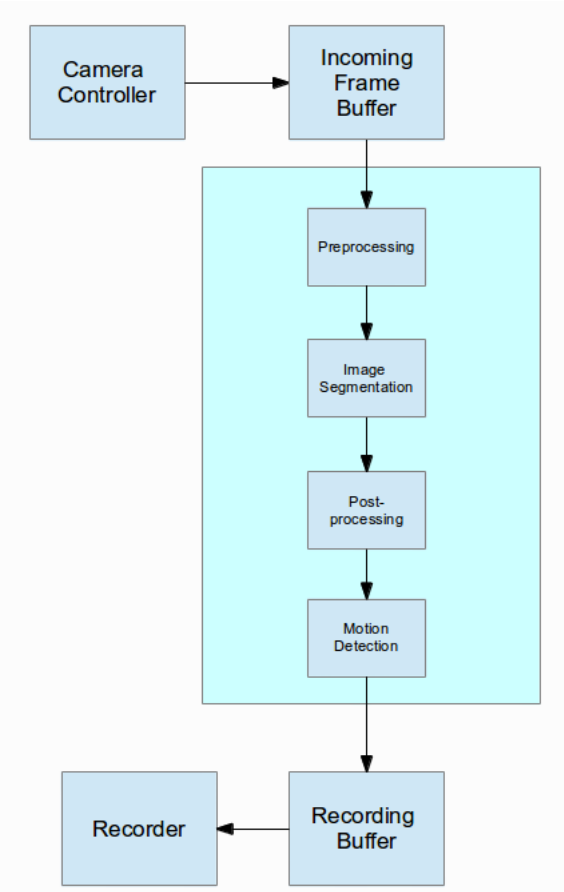

Figure 3.1: System Block Diagram

3.3 Task Analysis

The project methodology was refined to the following major steps:

- Investigation of suitable target devices (embedded computer) and support software and hardware.
- Investigation and implementation of possible suitable image processing techniques and algorithms.
- Design and implementation of a suitable camera control and frame buffering layer.
- Design and implementation of a suitable recording and storage layer.
- Full system integration and testing.

- Performance evaluation and recommendations of suitability, improvements and recommendations of parameters.
- Delivery of a suitable software prototype.

These tasks were identified as milestones for system development and as indicators of project progress. The following is an analysis of the details of each of the tasks and identifies any resources which are required for their completion.

3.3.1 Camera Control and Buffering

The vision system required a software layer to control the camera and buffer frames. The camera control layer must be able to interface with the camera and retrieve incoming frames from the internal buffers provided by Video4Linux. The camera control mechanism was to include a method to set the frame size and desired frame (sample) rate.

The buffering mechanism involved the development of a thread-safe (producer-consumer) frame buffer which is self contained and provides suitable methods for adding and retrieving frames.

The resources required for this task were:

- Suitable camera
- Working driver and API to control camera and retrieve incoming frames
- An API which provides suitable thread safety mechanisms (condition variables, mutexes)

3.3.2 Image Processing

The vision system required the implementation and development of suitable image processing techniques and algorithms. This layer was also required to interface with

the frame buffering mechanism described in the previous section. Incoming frames are to be processed to determine if relevant motion is occurring. The frame processing sequence was broken into four steps:

- Frame preprocessing
- Background subtraction
- Postprocessing
- Motion detection

To facilitate event detection, consideration was given to selection of a suitable sampling rate. If the processing sequence is not able to be performed at a suitable speed, successful segmentation of the background will not occur. An additional concern is the detection of fast moving objects. A faster sampling rate will increase the chances of detection of objects moving through the field. As such, the background modelling and processing stages were to be computed by the Pandaboard at a speed which can be considered greater than or equal to real time in order to ensure that buffer overflow does not occur.

This layer was to also provide the means for user configurable parameters which affect the output of background subtraction and motion detection. This was to be configurable by the user and must not require recompilation, as different use scenarios will require that parameters be set accordingly to ensure optimal segmentation and detection.

The resources required for this task are:

- A source of input. This could take the form of the camera control and buffering layer or a synthetic source.
- A library which provides the requisite data types and rudimentary routines for preprocessing and postprocessing.

3.3.3 Recording and Storage

Upon processing of frames, if an event has been detected the frame sequence is to be recorded to a file and stored on a static medium for later use. If a suitable Internet connection is available, these files will be synchronized to a remote server for retrieval.

To avoid large file sizes and facilitate rapid return of data, each file will contain only the events of a single hour. Thus, this task called for the implementation of suitable output measures, configuration of storage mediums and the production and modification of support scripts to facilitate data return.

The resources required for this task include:

- A working buffering mechanism.
- Static storage medium of sufficient size.
- Support scripts to allow synchronisation of recordings to remote machine.

3.3.4 Testing, Evaluation and Benchmarking

To validate the performance of the system it was necessary to define ground truths which allow a benchmark of performance to be determined. There are several data sets which are openly available which provide such a benchmark. These test sets will be used to validate the accuracy of detection and define boundary conditions for detection. The aim of this task was to validate the algorithm and signal processing techniques.

The software system was also tested to ensure that it is able to operate as a complete, self supervised system.

The resources required for this task are:

- All working components.
- Synthetic data sets annotated with existing ground truths.

- Configured test hardware.

The selection of the test sets will be discussed in a later chapter.

3.4 Resource Stipulations

Due to cost factors and limitations of hardware and software availability, the project requirements stipulated that several key pieces of hardware and software were be used.

3.4.1 Development & Deployment Hardware

A development workstation which is capable of running a similar environment to the target device is required for ease of development. The GNU/Linux operating system provides a free (open source) and robust UNIX-like environment whilst targeting many devices. Such an environment is able to support the necessary tools suitable for use with this software system.

The Ubuntu Linux distribution (Canonical Ltd. 2012) provides a simple package management system which allows a wide variety of software packages to be added to the system easily. Ubuntu also provides precompiled binaries which target a wide variety of architectures and a large repository of device drivers. The most recent Ubuntu release (Ubuntu 12.04) is a long-term support release, meaning that software updates will be available for five years. Thus, Ubuntu was considered a suitable target operating system for this project due to the steady development regime and it's portability.

The use of Ubuntu Linux allowed the selection of standardised hardware for development. The development workstation chosen for this project was a Dell Latitude E6410.

For the target device, consideration was given to price, power consumption, size, software accessibility and computational performance. As image processing can be a computationally expensive operation, this discounted many embedded devices as they simply do not provide sufficient computational ability for use with such a system. The

target device selected for this project was to also feature an onboard Floating Point Unit (FPU) in order to ensure sufficient floating point performance capabilities. Furthermore, the target device was to support the standard software environment.

Due to cost limitations, and the availability of hardware, the Pandaboard (Pandaboard 2012) was selected as target device. The features provided by the Pandaboard include:

- Texas Instruments OMAP4430 system-on-chip. This includes a dual-core ARM Cortex A9 CPU (with FPU).
- 1 gigabyte of RAM.
- 10/100 Ethernet.
- Bluetooth and WiFi connectivity.
- 2 x USB 2.0 ports.
- SD Card slot for operating system and local storage.
- 5W power consumption.

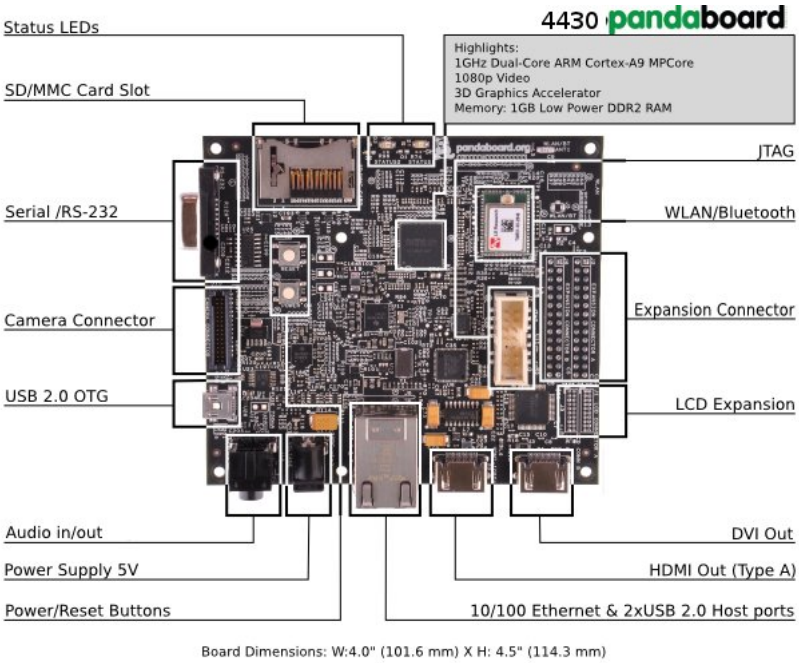

Figure 3.2: The Pandaboard

(Pandaboard 2012)

The Pandaboard is well supported by the Ubuntu operating system. It is of note that previous versions of Ubuntu do not support the Pandaboard's onboard FPU, merely providing emulated floating point computation. This significantly decreases the floating point performance of the device and would render the Pandaboard an unsuitable candidate. However, recent changes in the toolchain has led to the migration to the "armhf" port of Ubuntu (as opposed to the previous "armel" builds which only provide emulated floating point computation).

The final hardware component required was a suitable test camera. The camera was to be weather proof and night vision capable. Due to cost limitations, the camera selected is generic branded and little information is available. However, the following specifications are available:

- 640 x 480 maximum resolution.
- 25 frames/sec (maximum).
- Minimum illumination: 0.5 Lux.
- Lens: f = 6mm, F = 2.4 (infrared)
- USB interface providing Y2C video signal.

The Video4Linux API (Linux TV 2011) provides a suitable USB Video Class (UVC) driver interface to control the camera. This allowed the camera to be compatible with a wide variety of software packages and libraries.

3.4.2 Software Development Tools

Due to the use of the Linux operating system, an emphasis was placed on the use of open source software for development. The following software packages were added to each of the Ubuntu systems.

Figure 3.3: Test Camera

- `build-essential` - This package provides the GNU Compiler Collection, system libraries and other development tools such as `/usr/bin/make`.
- `vim` - The vim editor (`/usr/bin/vim`).
- `subversion` - The Subversion source code management system (specifically, `/usr/bin/svn`).

The OpenCV Library

The OpenCV (Open Source Computer Vision) library (Bradski 2000) provides data structures and algorithms for use in real time computer vision applications. Licensed under the BSD license, the source code is freely available and is able to used as a basis for building of vision-centred applications.

The OpenCV library provides a number of resources which were applicable to this application:

- A rich C++ API
- Basic camera control, frame capture and recording classes
- Matrix data types with automatic allocation and deallocation of memory and type conversion
- A wide selection of predefined image processing and vision algorithm

- Methods for the creation of rich user interfaces, useful for debugging and data collection

The OpenCV library is able to be linked to user code at compile time using the GNU Compiler Collection. The data types and data handling routines provided by OpenCV will be used as the core vision infrastructure for the system. The addition of specifically written algorithms and classes will be used to extend the functionality provided by OpenCV.

The utilisation of the OpenCV C++ API allowed the system to be portable to various architectures should future requirements change. Many routines provided by OpenCV include architecture specific performance enhancements, or make use of features provided by specific system. This was especially important for performance critical work, whilst retaining portability. The use of a common API allowed the vision system to be developed and tested on both the development workstation and the Pandaboard without code changes.

The Boost C++ Libraries

Boost (Boost 2012) provides libraries which are intended to supplement and extend the capabilities of the C++ programming language, and the standard template library. Of particular relevance for this project, the Boost libraries provided:

- A runtime arguments parsing and handling library
- A rich threading library, built on POSIX threading paradigms and the provision of thread synchronization types, such as mutexes and condition variables.

The Ubuntu Linux distribution provides packages which include the relevant development files and prebuilt objects for linking with user code. Boost version 1.4.6 was selected for this project.

GNU Screen

GNU Screen (The GNU Project 2010) is a terminal multiplexer, which allows a console session to contain many other console sessions. It can also be used to keep sessions active even if there are no users logged into the system. This was a useful function for the vision system to ensure that the vision software remains active on the system at all times, and was selected to be used as part of the self supervision systems.

rsync

rsync (Tridgell, A. and Mackerras, P. 1996) is a network protocol and application which can be used to synchronise directories and files on multiple machines over a local network or the Internet whilst minimising the data transferred. Due to bandwidth and cost considerations, the use of rsync is advantageous to the vision system, as under deployment conditions the system could be reliant upon Internet connections which are very costly when transferring large amounts of data.

OpenSSH

The OpenSSH (Secure Shell) protocol (The OpenBSD Project 2012) enables a secure remote terminal to be opened on the machine via the Internet. This is useful to monitor and administrate machines which are not local to the user but are able to be accessed remotely. This was considered to be a useful feature for the vision system, as the deployment site could be some distance from a convenient place of management.

Python

Python (Python Software Foundation 2012) is a high level, general purpose scripting language. It supports a wide variety of standard library functions, many of which are applicable to automation of tasks and machine administration. The Python language was selected to form the basis of the self supervision tasks required by the vision system.

3.4.3 Algorithm Verification Tools

The performance evaluation of the system was designed to demonstrate the suitability of the signal processing techniques for the application. This task entailed the use of synthetic test sets to be used as a ground truth for evaluation of the algorithms used in the vision system.

To provide an accurate account of the target environment, considerations regarding environmental and object variability were made. These considerations pertained to parameters such as object size, shape, orientation, velocity and trajectory. Additionally, environmental factors such as background topology, illumination and kinetic noise were considered.

The consideration of these factors led to the selection of the "PETS2001" (Computational Vision Group 2001) data set as the ground truth for evaluation and comparison. Datasets 1, 2 and 3 were selected, each providing an increasing level of challenge to the system. Dataset 1 was chosen to represent ideal conditions. Dataset 2 was selected to represent typical conditions. The selection of Dataset 3 was to measure system performance under worst case conditions, in order to place limitations and guarantees on system performance.

3.5 Consequential Effects

3.5.1 Sustainability

The Institute of Engineers Australia (Engineers Australia 2000) provides a set of guidelines to form the basis of evaluation processes relating to environmental, socioeconomic and cultural sustainability of projects over their lifetime.

The manufacturing of the equipment used in this project is outside the scope of control, however the development of the vision system considered these sustainability strategies and guidelines.

3.5.2 Safety

Equipment used in the development of the vision system was required to conform to relevant Australian and international safety standards. Where applicable, safety precautions were taken to minimise the risk of exceptional circumstances which could jeopardise safety.

3.5.3 Ethical Considerations

The Institute of Engineers Australia (Engineers Australia 2012) provides a code of ethics which presents a list of guidelines for engineering practices. These guidelines cover integrity, leadership, competence and sustainability. It is imperative that this code of ethics be upheld at all times throughout the undertaking of this project. This will ensure that the outcomes of this project are in line with the best interests of the community, environment and sustainability.

3.6 Risk Assessment

The risks and hazards associated with this project have been carefully considered, and strategies developed to minimise any impact to the ecosystem and any personnel (including the designer and future users). Risks which also pose a risk to the progress of the project have also been considered. Each risk has been categorised in terms of importance and likelihood. For details of these risks and their rating see Appendix B.

3.7 Research Timeline

The tasks laid out in the previous section were fitted against a timeline. See Appendix B.

3.8 Chapter Summary

In this chapter, the research and development methodologies to be used for the project were identified. The project was divided into a set of deliverables, and a sequence of tasks which were performed. The resources required for each of the tasks were identified and a schedule for their completion was proposed. The risks undertaken in the execution of this project have been outlined and strategies proposed to ensure that risks were minimised during and after the completion of this project.

Chapter 4

Design & Implementation

4.1 Chapter Overview

This chapter provides details of the software design and implementation based upon the requirements outlined in Chapter 3, and the existing methods and literature identified in Chapter 2. The chapter will outline the signal processing methodology employed by the vision system, and describe the software configuration and implementation.

4.2 Image Processing Methodology

In order to understand the implementation of the software system, it is necessary to explore the methodology behind the image processing techniques selected for investigation. The implementation details of this image processing methodology will be discussed in later sections.

As outlined in Section 3.3.2, the image processing routine was separated into four stages:

- Preprocessing
- Background subtraction

- Postprocessing
- Motion detection

4.2.1 Preprocessing

The frame preprocessing stage acquires incoming images from the frame buffer and performs necessary formatting and type conversions. In order to remain compatible with the background subtraction step, the incoming frames are converted to 8 bit grayscale. Prior to background subtraction, a Gaussian blur filter is applied. The blur filter element size is user configurable, so as to facilitate changes in sensitivity.

The blur filtering aims to accomplish two main tasks. The first is to remove noise & artifacts introduced by the camera sensor, as well as modest amounts of ambient noise. Additionally, by exaggerating the blur slightly, discernible detail within the image is greatly reduced. The reduction of detail contained within the image, aims to introduce a degree of tolerance to modest amounts of kinetic noise. It was hypothesised that the amount of blur would have an effect on accuracy and sensitivity - too little blur resulting in heightened sensitivity, and too greater blur resulting in missed events. The two images shown in Figure 4.1 show the original frame (left) and the result of a Gaussian blur with an element radius of 5 (right).

Figure 4.1: An example of Gaussian blur

It can be noted from the blurred frame that whilst the major details are still present, the blur has obfuscated many of the finer details. For example, the trees in the blurred frame have far less discernible features. By exploiting the fact that this application

does not require the vision algorithm to be aware of the fine details of it's surroundings, a degree of tolerance to the surrounding area can be introduced by deliberately obfuscating the details of elements within the frame. However, noise which drastically disrupts the pose of background objects will not be filtered. Thus, the use of a blur filter imposed a limitation upon the amount of noise which can be tolerated.

The effect of the filtering element size upon the accuracy of the system was used as a metric for evaluation in later sections.

4.2.2 Background Subtraction

As discussed in Section 2.3.1, the use of an image segmentation method to determine areas of observational interest can be used to further increase the tolerance of the system to it's surroundings. The introduction of the Gaussian Mixtures Model in Section 2.3.1 outlines two suitable methods for investigation and performance evaluation.

The Gaussian Mixtures Model accepts user-configurable initial parameters in order to facilitate optimal segmentation. These parameters will vary depending upon the setting, and as such they are provided to the user for modification in a manner which does not require modifying source code or recompilation.

As two separate GMM methods were investigated, the implementation of each method is modular and minimises modification to existing code in order to substitute for the opposing method.

Upon initialisation, the GMM requires a settling time in order to achieve model convergence. As such, a period of time which allows the model to settle was defined. Prior to the settling time elapsing, the system will not check for motion and thus no events can be recorded. The settling time selected was 100 samples.

Expectation Maximisation Algorithm

The use of expectation maximisation for model maintenance is common to the implementation of both the non-compressive and compressive sensing Gaussian Mixtures model. This section aims to outline the implementation of the algorithm in broad terms.

To begin the component update, the absolute deviation from the mean is calculated. If p represents the pixel value and k the component index, the difference can be calculated according to:

$$\Delta = |image[p] - component[k]| \tag{4.1}$$

A component is considered a match if the deviation from the mean is less than the deviation threshold, multiplied by the standard deviation of the component (or the minimum standard deviation, whichever is larger). Additionally, the weight of the component must be greater than the maximum weight, i.e. for N components the maximum weight is $\frac{1}{N}$. If a component does not match, a new component is created with initial parameters in place.

The update equations defined in equations 2.11, 2.12 & 2.13 are used to maintain the Gaussian components. Finally, the component ranking is calculated. The component ranking allows the components to be sorted according to their respective weights. The component is set to active if the sum of the component weights (not including the matching component) is greater than the background threshold.

User Parameters

Both GMM implementations require the following user modifiable parameters:

- α - model learning rate (input gain)
- Maximum Gaussian components (typically 3 - 5)

- Initial standard deviation
- Minimum standard deviation
- Deviation threshold
- Percentage of background components across the image
- Complexity prior

4.2.3 Postprocessing

The postprocessing step aims to further reduce internal and external noise. It is worth noting that the objects which the system will observe have a dense make up in that they can be considered to be solid objects, albeit with non-rigid poses. This distinct characteristic separates valid events from noise.

The behaviour of the GMM to kinetic noise was examined and the morphological structure of the output was studied. The details of this investigation is explained in Section 5.4. The frames in Figure 4.2 show the output of a GMM when introduced to a moderate amount of kinetic noise and the respective input frame.

Figure 4.2: GMM Behaviour - Kinetic Noise

The kinetic noise produces a sparse pattern of morphologically independent components, many of which consist of an area of only a few pixels. The introduction of a valid event into the frame, results in a mixed pattern of dense areas combined with the observed noise pattern (see Figure 4.3).

To extract valid regions within the mask, an assumption must be made regarding the density of components in areas which are to be considered noise. This is achieved by

Figure 4.3: GMM Behaviour - Mixed Event

a Morphological Erosion Filter. The MEF reduces the area of morphological regions which are not consistent with a preset density. (Gonzalez & Woods 2006)

The process of the erosion utilises a structuring element of a set size to be iteratively compared with parts of the image to observe how the shape fits within the morphology of the binary image. The size of the structuring element dictates the smallest allowable size for a region within the image. Pixels which do not adhere to this size threshold are set to zero, and as such the calculated mask is reduced to only consist of regions which were deemed to be sufficiently dense (see Figure 4.4). (Gonzalez & Woods 2006)

Figure 4.4: GMM Behaviour - Eroded Mask

To facilitate detection, regions which pass through the MEF can be exaggerated by an Morphological Dilation Filter (MDF). An MDF operates in converse to the MEF, employing a structuring element to dilate the area of regions which fall within the element. (Gonzalez & Woods 2006)

The utilisation of morphological filters placed a situational limitation on how kinetic noise generators will present post-segmentation. Thus, the successful operation of the morphological filter is largely dependant upon the quality of image segmentation pro-

vided by the GMM and the amount of kinetic noise present. This imposed limitation was considered as a compromise, as it was assumed that most spurious events only consist of mild to moderate changes in the calculated mask.

The level of noise tolerance provided by each of the image segmentation techniques was selected as a metric for evaluation. It was hypothesised that the CSGMM would not be as tolerant of noise, due to the reduction in the resolution of the locales modelled by the mixture of Gaussians.

4.2.4 Motion Detection

The detection of motion in the calculated mask relies upon the calculation of the optical flow, with reference to the current postprocessed mask and the previous mask. Thus, the system was required to be aware of the previous frame. The optical flow calculation, as described by the Farneback method in Section 2.3.2, produces a dense motion vector field. The detection of motion is reliant upon the presence of motion vectors with a magnitude other than zero.

4.3 System Configuration

This section describes the configuration of the development and runtime environments and the software configuration management strategies employed.

4.3.1 Software Installation

Ubuntu Linux 12.04 was installed on both the workstation and the Pandaboard. For the workstation, the standard `amd64` image was selected. For the Pandaboard, the `armhf` server image was selected as it features a kernel which supports preemptive scheduling. The development environment on both machines was intended to be identical, however cross compiling was not employed for this project. It was decided that all code which was produced must be able to be compiled on both the workstation and the Pandaboard

by merely changing compiler flags.

First, the GNU Compiler Collection, GNU Make and the relevant system libraries were installed via the `build-essential` package. The OpenCV libraries were installed on both machines by invoking the package manager with:

```
sudo apt-get install libcv* libopencv* libhighgui*
```

This installed OpenCV 2.3.1 and the necessary development header files to the system. The Boost C++ Libraries (version 1.4.6) were then installed with:

```
sudo apt-get install libboost.*1.4.6.1 libboost.*1.4.6-dev
```

The following command was then used to install the OpenSSH daemon, rsync and the Python interpreter:

```
sudo apt-get install openssh-server rsync python2.7
```

Upon installation, the linker was configured with the command, `sudo ldconfig` to ensure that the newly installed libraries were able to be linked at compile time.

4.3.2 Configuration Management

The Subversion (SVN) system was selected as the configuration management system for this project. Source code and documentation directories were created and added to the CSIRO SensorNet SVN tree. This was used to maintain synchronisation between the workstation and the Pandaboard and also to maintain revision history of source code.

4.3.3 Compiler and Linker Flags

GNU Make was selected as the build system for the project. The prototypical Makefile assumes that `g++` is available on the system, along with all requisite libraries.

For compiling on the Pandaboard, the following flags were defined:

```
-O3 -Wall -Wextra -pedantic -std=c++0x `pkg-config --cflags opencv`
-ffast-math -funroll-loops -mfloat-abi=hard -mfpu=neon -march=armv7-a
-mcpu=cortex-a9 -mtune=cortex-a9
```

These flags were necessary to ensure that floating point arithmetic was performed on the hardware FPU and not in emulation. The `-ffast-math` and `-funroll-loops` options were added for performance reasons. Flags specific to the ARM Cortex CPU were added to ensure that optimisation for the architecture was performed by the compiler. Of particular note is inclusion of the `-mfpu=neon` flag. The use of ARM NEON will be discussed in a later section.

The linking flags were defined as:

```
-lboost_thread -lboost_program_options `pkg-config --libs opencv`
```

All code presented in this document will compile without warnings when used with the included Makefile. The entire Makefile is included in Appendix C.

4.4 Implementation

The following sections describe the software implementation of each of the vision system components. Note that all code described here is listed in Appendix C unless otherwise stated.

The working title of the prototypical vision software was “FurryCap”.

4.4.1 Frame Buffering

The first component that the system required was a frame buffer to queue incoming frames. The frame buffer was implemented as a template class in order to interface with the OpenCV `cv::Mat` type.

The frame buffering mechanism is able to be accessed by multiple threads simultaneously. The class was designed with Mesa-style synchronization semantics in mind. The synchronization objects (scoped locks & mutexes) are provided by the Boost library, and are types `boost::mutex::scoped_lock` & `boost::mutex` respectively.

```
template<typename T>
class ts_buffer
{
  private:
    std::queue<T> payload;
    mutable boost::mutex m;
    boost::condition_variable cond;
  public:
    void push ( T const& data );
    bool is_empty ( void ) const;
    bool try_get ( T& val );
    void get( T& val );
};
```

Figure 4.5: Thread-safe Frame Buffer Class Prototype

Figure 4.5 shows the template class declaration with the details of the member functions omitted. For a complete listing, see Appendix C. The `push()` & `get()` member functions allow data to be pushed and retrieved from the buffer respectively. `is_empty()` indicates whether the buffer contains data.

4.4.2 Camera Control

The camera control layer was designed to interact with the Video4Linux API via the `cv::VideoCapture` object. The `VideoCapture` object allows different cameras connected to the system to be selected and settings regarding their capture stream to be

retrieved and set. The constructor for the `VideoCapture` object allows for the camera to be selected from the system's device file system. When connected to the system, UVC cameras will register as `/dev/videoX`, where `X` is the device number starting at 0. The `VideoWriter` constructor will accept this argument to differentiate between cameras in cases where multiple devices are connected. As such, the camera control layer was designed to accept this as an argument at run time.

Furthermore, the `VideoCapture` class enables parameters regarding the input stream to be set. This includes frame size and sample rate. The frame size and sample rate is to be configured by the user. A default frame size of 320×240 was selected. The frame size can be set using the `set()` member functions. The prototypical software included with this dissertation also offers the ability for the system to capture frames of sizes 160×120 and 640×480. This was added for the sake of complete customisability.

```
cv::VideoCapture cap;
cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
```

Figure 4.6: Setting frame size

A limitation with regards to frame rate was discovered in that Video4Linux does not support any other frame rate other than the full frame rate supported by the camera. The camera selected for the prototype system supports only 30 frames per second. As such, to emulate lower sampling rates, a frame counter was configured to only place a frame on the buffer if it conforms to the selected sampling rate. However, the assertion is made that the sampling rate must be a factor of the actual frame rate. The frame buffer declared for use with the capture routine is accessible by the frame processing routine. Thus, the frame buffer instance was declared as a global variable. This routine may require modification in cases where the camera is able to be correctly supported by Video4Linux.

The capture control routine was encapsulated within the function `cap_main()`. A full source listing is available in Appendix C.

4.4.3 Background Subtraction

In the interests of modularity, the background subtraction methods were each implemented as interchangeable sets of functions and classes. Each Gaussian Mixtures Modelling method included the use of expectation maximisation update algorithm to maintain the components.

To aid speed, all memory was allocated upon initialisation of the models. The use of expectation maximisation for model maintenance required the use of hardware supported floating point arithmetic for optimal performance. Emulated floating point arithmetic is supported, however the use of such will reduce performance.

Non-compressive Gaussian Mixtures Model

The implementation of the Non-compressive Gaussian Mixtures Model consists of three separate classes. The first class contains the user parameters. Upon instantiation, model initial parameters can be set by the user and passed to a model object. The architectural design for this model was inspired by the implementation presented by (Parks, D. 2011).

The user parameters class structure consists of:

```
typedef struct GMM_User_Params
{
  float low_thresh;
  float high_thresh;
  float bg_thresh;
  float alpha;
  float cp;
  float var;
  int max_components;
} GMM_User_Params;
```

Figure 4.7: GMM User Parameters Class

Each of the Gaussian components is represented by the following class:

```
class GaussianComponent
{
  private:
    float R_mu, G_mu, B_mu, sigma, w;
  public:
    GaussianComponent();
    void set_mu(float, float, float);
    void set_w(float);
    void set_sig(float);
    float r_mu(void);
    float g_mu(void);
    float b_mu(void);
    float weight(void);
    float sig(void);
}
```

Figure 4.8: GMM Component Class

It is of note that this model is capable of working with 3 channel images. In the interests of completeness, the implementation was designed with this capability. However, in the interests of comparability, only grayscale planar images with be modelled. The use of color need not be investigated, as in the case where the incoming frames have been sensed from the infrared capabilities of the camera, no color information is discernible.

The final class contains the Gaussian Mixtures Model itself, complete with EM maintenance algorithm.

```
class GaussianMixtures
{
  private:
    void setup(const cv::Mat);
    void subtract(const RGB_Img&,
                  Mask&,
                  Mask&);
    int pix_update(long,
                   const Pixel&,
                   int,
                   unsigned char&,
                   unsigned char&);
    int calc_comp_pos(int,int);
    bool fframe;
    long framen;

    float low_thresh, high_thresh, bg_thresh, alpha, cp, var;
    int max_components, f_width, f_height, f_sz;

    std::vector<GaussianComponent*> comp;
    std::vector<int> comp_per_pixel;
    IplImage *frame;
    RGB_Img _frame, mod_bg;
    Mask l_mask, h_mask;

    static const int BG = 0;
    static const int FG = 255;

  public:
    GaussianMixtures();
    GaussianMixtures(const GMM_User_Params&);
    void process(cv::Mat const&, cv::Mat&);
}
```

Figure 4.9: GMM Class

The data types used for this model are not standard to OpenCV, however are merely wrapper classes which provide easier pixel access than the legacy `IplImage` type. These wrapper classes were developed during an early implementation of the design and have been carried over to the final implementation. As a design improvement, the reliance upon these abstract types could be alleviated by taking advantage of the features for pixel-wise access provided by `cv::Mat`. These modifications would be merely cosmetic and for stricter compliance with the latest OpenCV API. The wrapper classes are included in Appendix C, under "type_wrappers.hpp" and "type_wrappers.cpp".

Whilst the architecture is similar to the implementation presented by (Parks, D. 2011), there are several key improvements which have been taken into account in the interests of compatibility and best practices. The implementation presented in this dissertation makes exclusive use of the standard library. Additionally, the model maintenance routines were updated to make use of new features provided by the OpenCV library, and some performance improvements.

The `GaussianMixtures` class provides a constructor which accepts a parameters structure. This constructor will set the internal parameters to the configuration defined in the parameter structure.

The `process()` member function is the only user-accessible interface into the model. This accepts the input frame (as a reference to a `cv::Mat`) and the reference to the calculated mask. Internally, this function calls upon `setup()` (in the case of the initial frame), and subsequently performs the necessary type conversions for the background subtraction itself. As the GMM models upon a per-pixel basis, the `subtract_pix()` routine performs the EM component maintenance, orchestrated by `subtract()`. The `calc_comp_pos()` function calculates the array indices of the `GaussianComponent()` objects related to the current pixel.

The output returned from `process()` is a binary mask image of the calculated segmentation. The model internally possesses a high and low threshold mask for use in the next iteration.

Compressive Sensing Gaussian Mixtures Model

The compressive sensing GMM operates in a similar manner to the standard GMM discussed previously. The CSGMM implementation discussed here is largely based upon the implementation used by (Shen et al. 2012). However, several modifications have been made to ensure compatibility with this project and to enhance performance on the ARM platform.

The initial implementation provided by (Shen et al. 2012) was written in pure C, and as such, heap memory allocation was managed by `calloc()` and `free()`. In the interests of compatibility and maintaining best practices, the implementation was modified to make use of `new` and `delete`.

Additionally, extensive modification was made to the structure of the model by implementing all associated functions and data structures inside a separate namespace. The model structure is shown in Figure 4.10. The model was also extensively modified to take advantage of the features offered by the ARM processor.

The namespace also includes the following functions for working with the CSGMM model:

- `init()` - Initialise the model by passing user specified parameters
- `destroy()` - Deallocates the CSGMM
- `process_frame()` - Performs the GMM update and compressive sensing projections

The `process_frame()` interface accepts pointers to the data sections of the input and output images, and byte steps describing their structure. For each frame, the `csProjection()` function is called in order to extract block and perform the matrix multiplication to reduce their dimensionality. The reference matrix is stored in static memory as a member of the namespace, specifically, “csc”. The ARM platform allows for matrix multiplications to be vectorized. This functionality is able to be utilised by way of ARM NEON compiler intrinsics. If compiled for ARM, `csProjection()`

will extract each block and perform each of the vector mutiply-accumulations as one operation. The NEON intrinsics make available data type primitives representing the slots of the NEON registers. (ARM 2012) The lookup matrix is defined as the `int16_t` type, and as such the NEON intrinsic of interest is the `int16x8_t` type (ARM 2012). This type represents a vector of eight signed 16 bit integers. In the interests of speed, the lookup matrix is preallocated upon model initialisation.

Additionally, the use of the NEON intrinsics limits the number of projections to eight. This is of no detriment, as the data presented by (Shen et al. 2012) suggest that a projection size of eight presents an optimal balance between performance and accuracy.

After the projection is performed, the EM component maintenance routine is carried out. This updates each component and sets up the mask for background. Background refinement is performed by `postProc()`, and will transfer the projection outcomes to pixels in the output mask image.

```
  typedef struct CSGMM_
  {
    uint32_t _width, _height;
    uint16_t _numGau;
    uint32_t * _rankIndex;
    uint32_t _initWithImage;
    uint32_t _numForegroundPixel;
    float _df;
    float _minStd;
    float _bgGauTh;
    float _alpha;
    float _sdInit;
    float * _weight;
    float * _sd;
    float * _rank;
    float * _mean;

    //Post processing variables
    uint8_t _postProcTh;
    uint8_t _postProcAlpha;

    //Temp storage variables
    uint8_t * _cIm;    // the current planar intensity image
    int16_t * _csr;    // for storing compressive sensing output, width*heigh/8
    uint8_t * _csmogr; // for storing mog output, width*heigh/8
    uint8_t * _bkgnd;  // background image;
    uint8_t * _csm;    // compressive sensing mask image
#ifdef __ARM_NEON__
    int16x8_t neon_lookup[64];
#endif
  } CSGMM;
```

Figure 4.10: CSGMM Model Structure

GMM Performance Considerations

To enhance performance, the `-funroll-loops` compiler flag was added to the build system to ensure that, where possible, repeated code compiles to sequential instructions and minimises the use of branching.

Additionally, the `-ffast-math` flag was included to facilitate best-case floating point performance. The decreased precision brought about by the use of this features was not considered of detriment. (ARM 2012)

4.4.4 Postprocessing (Density Filtering)

The morphological erosion and dilation was performed via the `cv::erode()` and `cv::dilate()` library functions. The structuring element size was defined as a local stack variable, and instantiated with `cv::getStructuringElement()`. (Bradski 2000)

4.4.5 Motion Detection

The Farneback Optical Flow method was provided by the OpenCV library. The optical flow between two frames can be calculated using the `cv::calcOpticalFlowFarneback()` function. This function accepts the current frame and the reference frame, and returns the vector field as a dual channel matrix. (Bradski 2000)

The `cv::calcOpticalFlowFarneback()` function also accepts the following parameters which control the accuracy of the motion estimation:

- Number of iterations for polynomial fit refinement
- Neighbourhood window size
- Polynomial order to fit to motion
- Polynomial variance
- Pyramidal expansion scale

(Bradski 2000)

The motion detection for this application does not rely on any information provided by the vector field, other than the inclusion of non-zero motion vectors. The function shown in Figure 4.11 calculates the optical flow between two frames, and places the motion vectors into a user specified matrix. Upon calculation of the optical flow, the flow field is selectively searched for the presence of non-zero components. The function returns a Boolean to indicate the current trigger state.

```
const float pyr_sc = 0.5;
const float levels = 3;
const float win_sz = 15;
const float iter = 3;
const float poly_order = 5;
const float poly_sigma = 1.2;

bool trig( cv::Mat& curr, cv::Mat& prev, cv::Mat& motion_vectors )
{
  int i, j;
  std::vector<cv::Mat> chans;
  cv::calcOpticalFlowFarneback( prev, curr, motion_vectors,
                                pyr_sc, levels, win_sz, iter,
                                poly_order, poly_sigma , 0 );
  cv::split(motion_vectors, chans);

  // Test to see if there are non-zero motion vectors
  for (i = 0; i < chans.at(0).rows; i++)
    for (j = 0; j < chans.at(0).cols; j++)
    {
      if (chans.at(0).at<float>(i,j) != 0 || chans.at(1).at<float>(i,j) != 0)
        return true;
    }
  return false;
}
```

Figure 4.11: Optical Flow Trigger Function

4.4.6 Recording

To facilitate ease of data return and a logical organisation of collected data, the data collected is separated into files based upon the hour in which they were collected. OpenCV provides the `VideoWriter` class as an external interface for saving sequences to files on disk within an AVI container. (Bradski 2000) The path where these files are stored is able to be configured at runtime.

The `EventRecorder` class was developed to automatically configure the output file based upon the hour and write frames to the file at the correct frame rate. The output file is named according to the date and hour.

Object Velocity & System Response Considerations

To provide better context to the recordings, the `PretrigBuffer` class was developed to hold a user configurable number of frames which are swapped out in a First-In-First-Out manner. The context is provided by placing frames which do not contain an event onto the pretrigger buffer, and subsequently placing the contents of this buffer into the output file prior to later samples.

The experiment which led to the design of the pretrigger buffer is outlined in Section 5.4. The pretrigger buffer also provides a level of protection against delayed triggers.

4.5 System Overview

The components of the system are illustrated according to the diagram in Figure 4.12.

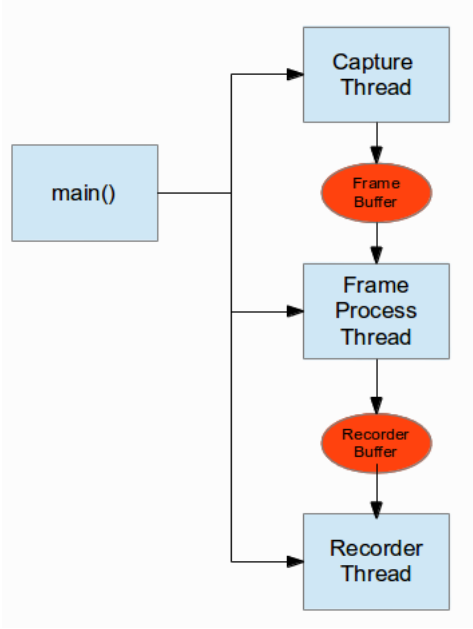

Figure 4.12: System Component Interactions

4.5.1 Threading

Due to the need for real-time processing, each component of the system was separated into threads. `main()` starts each component as a `boost::thread` object, and is added to a `boost::thread_group` object. (Boost 2012) All buffer objects are instantiated in static memory in order to be accessible by multiple threads.

All configurable parameters are set through the use of runtime arguments, and are either transferred to static variables or passed to each thread as an argument. The use of runtime arguments allows for simple configuration by the user or scripted operations, and eliminated the need for a configuration file and associated parsers.

The prototypical software presented in Appendix C accepts the following arguments:

- "help" - Displays the help message.

- “c” - Set the capture device, i.e. /dev/video0 would be selected with “–c 0”. The default device number was set to zero.
- “targ_pb” - Sets the target device as the Pandaboard (frame size as 320×240.
- “targ_pc” - Sets the target device as a PC (frame size as 640×280.
- “targ_demo” - Sets the frame size to 160×120.
- “pretrig_t” - Sets the pretrigger buffer length.
- “rec_t” - Sets the record time.
- “rec_path” - Specified the subdirectory where the recording output is stored.
- “blur” - The blur filter element size. Defaults to 3.
- “f” - Set the frame (sample) rate. Defaults to 10 Hz.
- Gaussian Mixtures Model specific settings - learning rate, standard deviation, background threshold, maximum components. As discussed in previous sections.

4.5.2 Frame Processing

The frame processing thread retrieves frames from the raw frame buffer (shared with the capture thread), performs the image segmentation and motion classification operations and subsequently operates the pretrigger buffer and the recording buffer.

A frame counter increases with each frame, and is used to calculate the end of the warm up period (allowing the GMM to converge) and recording frame offsets. The suggested time for the warm up period is 100 samples. The recording segment time is suggested to be 5 seconds with a 5 second pretrigger buffer length.

The trigger state is managed by way of a state machine. The state machine will set up the recording time offsets upon triggering, dump the pretrigger buffer and maintain the recording buffer. Once the recording time has elapsed, the state is changed and the cycle repeats.

The frame handling process is summarised in Figure 4.13.

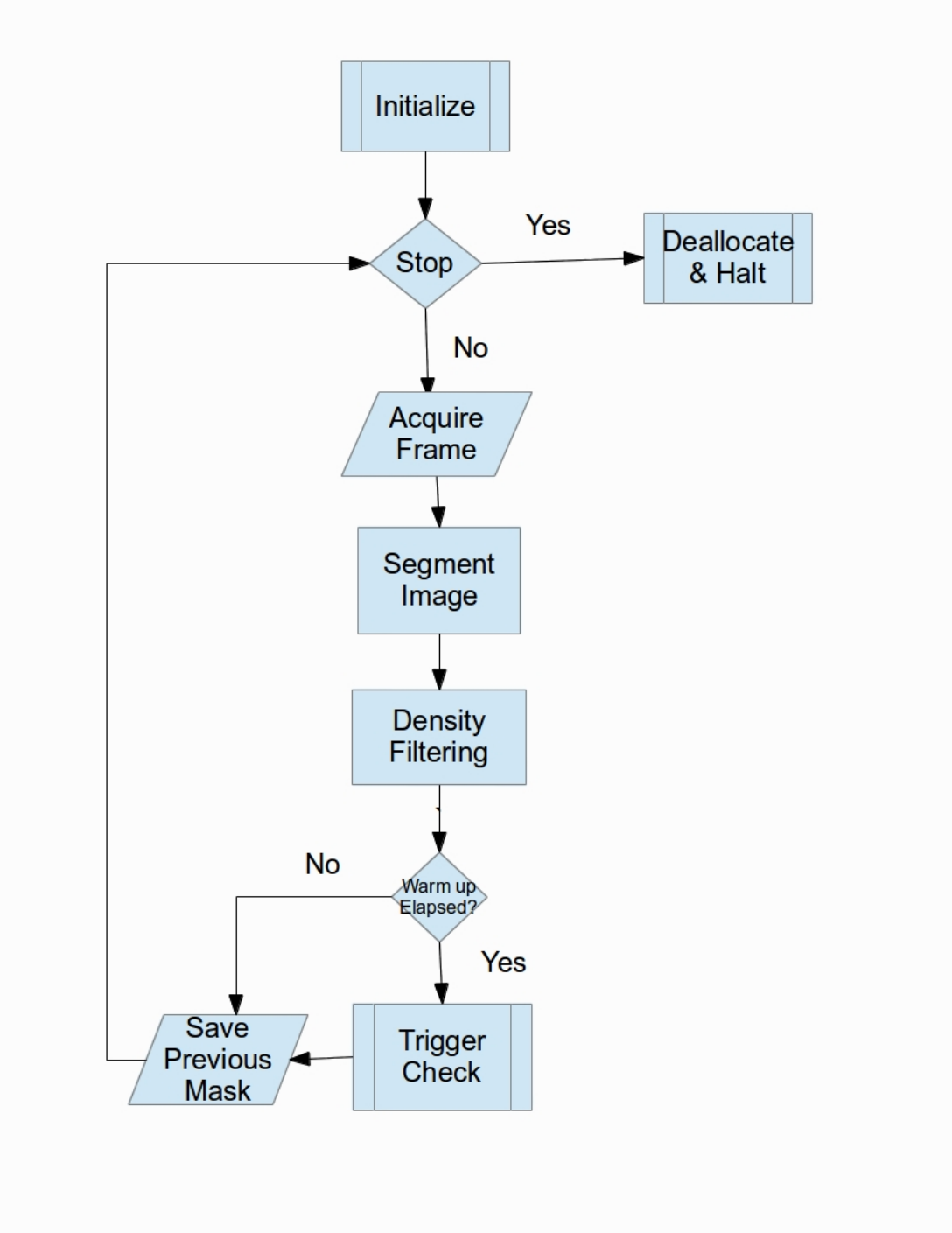

Figure 4.13: Frame Process Flow Chart

4.6 Support Software

4.6.1 Self Supervision

The system is to be self supervised using the supervisor script (written in Python) which is currently a part of CSIRO's Sensornet deployment software base. The details of the script will not be explored at length, however the script performs the following operations:

- Configures system variables, cron jobs and maintains the system clock.
- Maintains connectivity to the CSIRO VPN.
- Starts pre-defined software in a GNU Screen session. This will also check to see if a list of software is running and will be restarted in the event of a crash.
- Sets a crontab entry such that the supervisor script will be run automatically at specified intervals.

The supervisor script was modified to suit the addition of the vision system software and the rsync data synchronisation technique outlined in the following section. The supervisor script was configured to run once every five minutes. This was considered acceptable, such that in the event of a software crash, a maximum of five minutes observation time is lost.

The supervisor script is included in the Appendix E.

4.6.2 Data Synchronisation

The data synchronisation for this application is provided by rsync (Tridgell, A. and Mackerras, P. 1996) and is managed by the supervisor script. The supervisor script configuration allows the recording path and the destination server (and associated credentials) to be modified. The rsync command is placed into the cron table (`/etc/crontab`) and synchronises the recording directories on both machines. This allows the output

files to be collected from the remote server for later analysis. The rate at which the cron job is run will be required to be varied depending upon the Internet connection used in the field. The most likely candidate is a cellular modem, and as such this interval should be kept as long as desirable due to increase bandwidth costs associated with each synchronisation. The script configuration included with this document sets the data return time to twice per hour.

4.7 Chapter Summary

This chapter has detailed the signal processing methodology used by the vision system. The system configuration for the development and employment environment was discussed. The implementation of the vision system in software was detailed, and the support software base was designed and configured to allow autonomous operation of the vision system.

Chapter 5

Performance Evaluation

5.1 Chapter Overview

This chapter aims to explore the performance of the proposed vision system. The methodology and performance measures for each component have been defined. This chapter also includes the results yielded from the performance evaluation and recommendations drawn from this analysis.

5.2 Performance Metrics and Methodology

The performance evaluation of system was designed to demonstrate the suitability of the proposed technique. The evaluation consisted of two major components - the real time capabilities of the system and the effectiveness of the detection system. Additionally, the capabilities provided by the vision system for self supervision were also evaluated.

The real time capabilities of the system were demonstrated by the timing of the each stage within the frame handling process.

The evaluation of the detection systems aimed to place a performance guarantee on the system's ability to detect events. To achieve best performance, the initial parameters

provided for background subtraction were suitably selected.

Additionally, the tests performed aimed to provide a demonstration of "worst case" performance, in order to place a guarantee on the performance of the system. This consisted of a benchmark of processing performance and detection rate under varied conditions.

5.3 Processing Speed Evaluation

5.3.1 Methodology & Metrics

The prototypical software was modified to allow time measurements of the processing algorithm to be taken. The time was estimated by the use of the `cv::getTickFrequency()` and `cv::getTickCount()` functions provided by OpenCV. The former measures the number of CPU clock ticks which occur in a second. The latter returns the count on the free-running clock. Thus, in order to estimate the time taken for each step, assume that:

- Δt is the number of clock ticks per second
- x is the clock value a priori
- y is the clock value a posteriori

As such, the frame rate ($\frac{\Delta f}{\Delta t}$) can be calculated according to:

$$\frac{\Delta f}{\Delta t} = \frac{\Delta t}{y - x} \tag{5.1}$$

Thus, the time for each processing step (t), in milliseconds:

$$t = \frac{1000}{\frac{\Delta f}{\Delta t}} \tag{5.2}$$

The stages of processing which were evaluated for performance were:

- Background subtraction step - this was selected as the main metric for evaluation of processing speed.
- An estimate of total frame processing time: Frame retrieval from the incoming frame buffer to the setting of the trigger bit after motion detection. This was compared for both of the background subtraction methods.

Both the compressive and non-compressive background subtraction methods were compared for performance. The same parameters were utilised for both models to provide an accurate comparison of performance.

To simulate deployment conditions, the tests were performed on the Pandaboard with the software configuration that would be deployed on a production system.

As a starting point, a sample rate of 10 Hz was selected. This sample rate was selected as a usable target for real-time performance.The camera was connected to the system and the average time, best & worst time for each step was collected over 1000 frames.

The proposed background subtraction methods were each tested with the initial parameters defined in the previous section.

It is of note that the performance of the frame processing algorithm is dependant upon a number of factors. A Gaussian Mixture Model in it's infancy will result in slower performance, as the Gaussian components are yet to be matured and require constant re-computation. However, upon model maturity it can be expected that performance will increase. Additionally, the introduction of a large number of foreground components into the frame will result in a similar performance drop.

The performance goal of the frame processing algorithm is that it will operate at or faster than the sampling rate under nominal conditions. Ideally, the frame process time should provide for some overhead to allow the system to "catch up" in cases where throughput decreases. This deviation should be no more than two sampling periods.

5.3.2 Performance Measurement

The following tables display the results of the speed measurement testing as described above.

Time:	Average (ms)	Best Case (ms)	Worst Case (ms)
Background Subtraction	208.333	204.082	243.9
Total Process Time	231.24	216.43	278.97

Table 5.1: Frame Process Timing: Non-compressive

Time:	Average (ms)	Best Case (ms)	Worst Case (ms)
Background Subtraction	59.172	55.249	62.112
Total Process Time	71.428	66.667	100.705

Table 5.2: Frame Process Timing: Compressive

5.3.3 Results Discussion

The results shown in Table 5.1 indicate that the non-compressive Gaussian Mixtures Model is too slow for this application. Using such a method would result in an overall frame processing rate of less than 5 frame/sec. Hence, the performance goal was not met. The compressive sensing Gaussian Mixtures Model was therefore recommended for use whilst the application is still reliant upon the Pandaboard. The use of the non-compressive GMM may be considered in the case where the target hardware provides greater computational abilities.

The compressive sensing Gaussian Mixtures Model (as shown in Table 5.2) provides an average throughput of 16.89 frame/sec. The total processing time offers a worst case of 9.93 frame/sec which is still within acceptable limits and should enable the system to operate without detriment. It is of note that the worst case time was observed whilst the Gaussian Mixture Model was maturing in both cases. Upon model convergence, the performance was shown to be closer to that of the average measurements in each case.

5.4 Preliminary Processing Algorithm Verification

The methodology behind the segmentation and detection testing was developed with the outcomes of two precursor trials which were carried out during the incremental development of the system. To better understand this methodology, a description of such tests and respective outcomes has been provided.

The aim of these tests was to demonstrate the ability of the system to isolate events occurring within the frame and to gain an understanding of the segmentation behaviour when presented with noise. The preliminary test involves placing the camera in an outdoor setting and allowing the model to converge. Upon convergence, the detection system was tested by introducing a moving object into the frame at a variety of distances and trajectories.

The initial test revealed a 100% detection rate with no false positives. The kinetic noise did not seem to have an effect on the detection, and it can be noted in the resulting output that there are slight variations in natural lighting over the length of the test. Although the detection rate was acceptable, it was noted that some objects did not register immediately. Thus, the implementation of the pretrigger frame buffer was designed as per Section 4.4.6, in order to assist in the recording of events under the occurrence of detection latency.

The results from this test can be viewed via the Internet. See Appendix F, Section F.1.

Additionally, the scene was adjusted to allow for a test which exposes the system to a higher degree of kinetic noise. This was achieved by shifting the scene slightly to exaggerate the shadows cast by trees moving in the breeze. This would be considered an unideal operating condition. As with the above test, objects were introduced into the field of view at varying distances.

Whilst the system showed motion on 100% of the desired events, there were three events which could be deemed spurious. It is of note that the spurious triggers occurred during Gaussian Mixture Model infancy, which led to the implementation of a “warm up” period, as discussed in the previous chapter.

The results from this test can be viewed via the Internet. See Appendix F, Section F.2.

The outcomes of these tests identified a need to evaluate the system against a set of known testing conditions to place limitations upon the detection system. However, whilst both tests provided a preliminary validation of the signal processing strategies applied, further testing was applied to challenge the limits of the system operation.

The outcomes of these tests also shaped the development of the foreground density filtering methodology outlined in Section 4.2.3. The tests showed the valid objects registered as dense clusters of foreground components, whilst noise events were more sparse and were largely represented by areas of smaller dimensionality than valid events.

5.5 Extended Processing Algorithm Verification

The aim of this evaluation experiment was to validate the processing algorithm across a variety of conditions.

5.5.1 Methodology, Metrics & Ground Truths

The ground truth for these experiments was provided by the use annotated test sets which encompasses the target domain of the system. To provide an accurate account of the target environment, considerations regarding variability in significant events were made. These considerations pertained to object variability and environmental ambiance. Object variability encompasses parameters such as size, shape, orientation, velocity and trajectory. The environmental ambiance encompasses parameters such as background topology, illumination, kinetic noise and external interference (i.e. CCD sensor noise).

The consideration of these parameters led to the selection of the "PETS2001" dataset (Computational Vision Group 2001). Although the target application is the monitoring of wildlife, the PETS2001 test set provided comparable segmentation and detection challenges. The variability in object size, shape, trajectory, visibility and colour con-

trast mimics the variability potentially encountered in the target application.

Three datasets were selected for use; specifically, Dataset 1, Dataset 2 and Dataset 3. The single camera portion was selected for use from each dataset. Each dataset provides an increasing level of difficulty in processing. Dataset 1 and Dataset 2 were considered to be ideal and typical (respectively) operating conditions in that they provide reasonably consistent lighting conditions and limited amounts of environmental noise. Additionally, detection of events is aided by the introduction of comparatively large objects. Both test sets contain events featuring people and vehicles. The trajectory, distance and speed of all objects were highly variable across the test sets.

Figure 5.1: An example frame from Dataset 1

Dataset 3, however, is designed to challenge detection systems. In addition to the introduction of smaller objects (no vehicular events), there was a significant degree of lighting variability and kinetic noise. Dataset 3 was selected to demonstrate system performance under less than ideal circumstances in order to provide an account of worst case performance.

The specifications of each ground truth included an analysis of the events which present as positive events within the sample frames, and an exact notation of their motions throughout the sequence. As such, the test annotations were analysed and the events were classified as an acceptable or rejectable event. These classifications were made on merits of usefulness of their detection. Rejectable events could be considered noise as their detection would not provide useful data regarding their occurrence (i.e. hidden behind an object and subsequently would not be easily observed). The event annotations are provided in Tables 5.3, 5.3 & 5.5.

Figure 5.2: Example frames from Dataset 3 showing a significant degree of illumination change

Time	Event Description	Classification
0:02 - 0:12	Person hiding behind car	Reject (not useful)
0:03 - 0:26	Person with red shirt, from left	Accept
0:17 - 0:27	Blue car, from right	Accept
0:26 - 0:39	White van, from left	Accept
0:32 - 0:44	Man with backpack, from left	Accept
0:36 - 0:51	Man from blue car	Accept
0:52 - 1:14	Two men, near white van	Accept
1:03 - 1:18	White van reverse	Accept
1:13 - 1:31	Man, far back	Accept
1:20 - 1:42	Lady, bottom right	Accept
1:25 - 1:47	Station wagon entry	Accept
1:39 - 1:47	White van drives away	Accept

Table 5.3: Dataset 1 Event Annotation

Time	Event Description	Classification
0:02 - 0:17	Blue car enters	Accept
0:10 - 0:13	Black car drive by	Accept
0:14 - 0:17	Black car drive by	Accept
0:15 - 1:08	Person, blue shirt, from right	Accept
0:18 - 0:33	Person, black jumper, from right	Accept
0:20 - 1:24	Person with backpack, from right	Accept
0:26 - 0:42	Person, white clothes, far back	Accept
0:29 - 0:56	Person on bicycle	Accept
0:39 - 0:59	Person, black pants, far back	Accept
0:59 - 1:52	Person on bicycle	Accept
1:02 - 1:15	Person, white shirt, far back	Accept
1:04 - 1:52	Two people, from right	Accept
1:14 - 1:26	Person, dark clothes, far back	Accept
1:22 - 1:40	Person, from right	Accept
1:27 - 1:33	Person, from blue car	Accept
1:28 - 1:32	Dark car drive by	Accept
1:35 - 1:52	Person with plank, near buildings	Reject (obfuscated)
1:46 - 1:52	Two people, far back	Accept
1:48 - 1:52	Red car drive by	Accept

Table 5.4: Dataset 2 Event Annotation

Time	Event Description	Classification
0:15 - 0:42	Rapid lighting change	Reject (noise)
0:41 - 2:03	Two people, from bottom	Accept
0:51 - 1:12	Person, with briefcase from side	Accept
0:56 - 1:30	Two people, from building at rear	Accept
1:10 - 1:24	Person, on path behind building	Accept
1:13 - 2:21	Man, white shirt, from bottom	Accept
1:21 - 3:10	Cluster, 11 people	Accept
1:46 - 2:05	Rapid lighting change	Reject (noise)
1:54 - 2:23	Man on bicycle	Accept
1:58 - 3:07	Man with backpack	Accept
2:12 - 2:56	Man, black shirt	Accept
2:20 - 2:46	Man on bicycle	Accept
2:20 - 3:03	Rapid lighting change	Reject (noise)
2:25 - 3:33	Man, at rear	Accept
2:46 - 3:33	Two men with backpacks	Accept
3:24 - 3:33	Man, from bottom left	Accept

Table 5.5: Dataset 3 Event Annotation

In the case of Dataset 3, some events were recognised as multiple events. This was limited to the cluster of 11 people which eventually separates into multiple, distinct groups. Thus, the total events (acceptable and rejectable) for Dataset 3 was considered to be 21. In addition to the noise events listed in the annotation table, there is almost constant vegetation motion in various portions of the frame.

In order to place measures upon the effectiveness of the system, the event annotations were used to qualify detected events as a True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). An event was considered to have been detected if at any time during it's presence within the frame, the motion vector neighbourhood surrounding it was shown to be active. Conversely, if an event was not detected, the motion vector neighbourhood should remain inactive. This was considered realistic, as the design goals did not dictate that events must be captured in their entirety.

From these measures, the following metrics were defined:

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{5.3}$$

$$Precision = \frac{t_p}{t_p + f_p} \tag{5.4}$$

$$Sensitivity = \frac{t_p}{t_p + f_n} \tag{5.5}$$

$$Specificity = \frac{t_n}{t_n + f_p} \tag{5.6}$$

The accuracy and precision for each test was graphed to show the relation with the blur filter settings. The sensitivity and specificity can be represented according to a Receiver Operating Characteristic (ROC) analysis. ROC models the probability of event detection against the probability of a false alarm (Shen et al. 2012) (Hand 2009). This shows the system specificity in terms of the sensitivity, and subsequently provides a measure of system performance as a guaranteed level of detection with the chance of false triggers. (Hand 2009)

Sensitivity, Specificity & Detector Intelligence

The ROC plot compares system sensitivity with inverse specificity. The horizontal axis consists of a plot of $1 - Specificity$, with the vertical axis being sensitivity. (Hand 2009)

The detector can be considered "intelligent" if the detection outcomes are better than that provided by a random, uniformly distributed classification. In essence, random classification can be represented by a 1:1 relationship between sensitivity and inverse specificity. (Hand 2009) Thus, detections which are superior to this random classification fall above this relationship on the ROC plot.

Detector Testing Methodology

As a comparison of performance, both the non-compressive and compressive sensing image segmentation methods were compared. The parameter which was used to tune the accuracy of the system was the element size used in the Gaussian blur filtering step. Varying this size enabled the sensitivity of the event detection algorithm to be increased (smaller values) or decreased (larger values). The filter size was varied from 1 through 19, in odd increments. A control run without blur filtering applied was also performed.

The morphological filtering parameters were set to a "common sense" value, and were kept consistent throughout the testing. In all cases, the erosion filter element size was set to 4, and complimented with a dilation element size of 4.

To keep image segmentation consistent, the parameters selected for the Gaussian Mixtures Model were applied to both image segmentation methods. These parameters were selected after consideration of the results put forward by (Shen et al. 2012), which suggest that the use of a Gaussian Mixtures model consisting of three Gaussian distributions per component allows for approximation of 99.9% of the pixel components. Their use of PETS2001 is also utilising three Gaussian components. Thus, the system was tested with a maximum Gaussian component limit of three.

The Gaussian Mixtures Model requires a sufficient learning rate such that objects which enter and remain in view are eventually regarded as background. However, making this allowance too great can result in false positives. As such, given a sufficient sampling rate a value of α as 0.1 for foreground refinement is typical to this type of application. (Shen et al. 2012)

As the Gaussian Mixtures Model relies upon initialisation with parameters which have been estimated a priori, in order to facilitate best case segmentation with parameters refined a posteriori the initial parameters must provide a reasonable estimate for the scenario.

The following parameter estimates were used for use with the test data sets:

- Initial standard deviation - 50
- Minimum standard deviation - 15
- Deviation threshold for block detection - 3 times the standard deviation
- Assumed proportional background - 70%
- Initial Gaussian component weighting - 0.01
- Postprocessing threshold - 10

(Shen et al. 2012)

The output of each test was recorded, showing the input frame, calculated mask, post-processed mask and the optical flow field. The optical flow field was rendered graphically using the function listed in Figure 5.4.

Each output was recorded to a video file and analysed manually (see Figure 5.3).

Figure 5.3: Test Output Example

```
void draw_flow_map(const cv::Mat& flow, cv::Mat& cflowmap, int step,
                   double, const cv::Scalar& color)
{
  cv::Mat log_flow, log_flow_neg;
  cv::log(cv::abs(flow)*3 + 1, log_flow);
  cv::log(cv::abs(flow*(-1.0))*3 + 1, log_flow_neg);
  const float scale = 64.0;
  const float offset = 128.0;

  float max_flow = 0.0;

  for(int y = 0; y < cflowmap.rows; y += step)
  {
    for(int x = 0; x < cflowmap.cols; x += step)
    {
      const cv::Point2f& fxyo = flow.at<cv::Point2f>(y, x);
      cv::Point2f& fxy = log_flow.at<cv::Point2f>(y, x);
      const cv::Point2f& fxyn = log_flow_neg.at<cv::Point2f>(y, x);

      if (fxyo.x < 0) {
        fxy.x = -fxyn.x;
      }
      if (fxyo.y < 0) {
        fxy.y = -fxyn.y;
      }
      cv::Scalar col = cv::Scalar(offset + fxy.x*scale,
          offset + fxy.y*scale, offset);
      cv::line(cflowmap, cv::Point(x,y), cv::Point(cvRound(x+fxy.x),
            cvRound(y+fxy.y)), color);
      cv::circle(cflowmap, cv::Point(x,y), 0, col, -1);

      if (fabs(fxy.x) > max_flow) max_flow = fabs(fxy.x);
      if (fabs(fxy.y) > max_flow) max_flow = fabs(fxy.y);
    }
  }
}
```

Figure 5.4: Optical Flow Render Function

5.5.2 Dataset 1 Results

The results of the detection tests against Dataset 1 are shown in Table 5.6. The results are shown graphically in Figure 5.5. The output of each test can be viewed individually - see Appendix F.3.1.

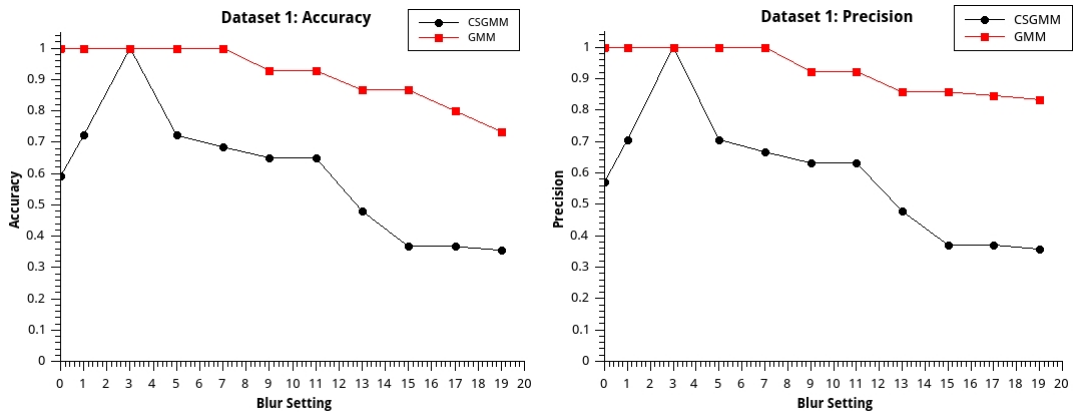

Figure 5.5: Dataset 1 Results

The ROC plot for Dataset 1 is shown in Figure 5.6.

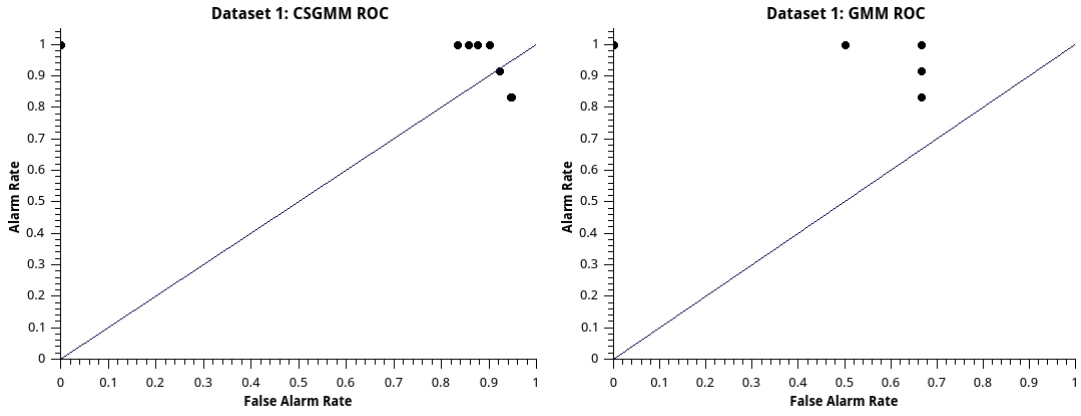

Figure 5.6: Dataset 1 ROC

	Compressive	Non-Compressive	Compressive	Non-Compressive
No Blur			**Blur Size: 11**	
TP	12	12	12	12
TN	1	1	1	1
FP	9	0	7	1
FN	0	0	0	0
Accuracy	0.59	1	0.65	0.93
Precision	0.57	1	0.63	0.92
Sensitivity	1	1	1	1
Specificity	0.1	1	0.125	0.5
Blur Size: 1			**Blur Size: 13**	
TP	12	12	11	12
TN	1	1	1	1
FP	5	0	12	2
FN	0	0	1	0
Accuracy	0.72	1	0.48	0.867
Precision	0.706	1	0.478	0.857
Sensitivity	1	1	0.917	1
Specificity	0.167	1	0.077	0.33
Blur Size: 3			**Blur Size: 15**	
TP	12	12	10	12
TN	1	1	1	1
FP	0	0	17	2
FN	0	0	2	0
Accuracy	1	1	0.367	0.867
Precision	1	1	0.37	0.857
Sensitivity	1	1	0.83	1
Specificity	1	1	0.056	0.33
Blur Size: 5			**Blur Size: 17**	
TP	12	12	10	11
TN	1	1	1	1
FP	5	0	17	2
FN	0	0	2	1
Accuracy	0.72	1	0.367	0.8
Precision	0.706	1	0.37	0.847
Sensitivity	1	1	0.83	0.917
Specificity	0.167	1	0.056	0.33
Blur Size: 7			**Blur Size: 19**	
TP	12	12	10	10
TN	1	1	1	1
FP	6	0	18	2
FN	0	0	2	2
Accuracy	0.68	1	0.355	0.73
Precision	0.67	1	0.357	0.833
Sensitivity	1	1	0.833	0.833
Specificity	0.143	1	0.0526	0.33
Blur Size: 9				
TP	12	12		
TN	1	1		
FP	7	1		
FN	0	0		
Accuracy	0.65	0.929		
Precision	0.632	0.923		
Sensitivity	1	1		
Specificity	0.125	0.5		

Table 5.6: Dataset 1 Detection Outcomes

5.5.3 Dataset 2 Results

The results of the detection tests against Dataset 2 are shown in Table 5.7. The results are shown graphically in Figure 5.7. The output of each test can be viewed individually - see Appendix F.3.2.

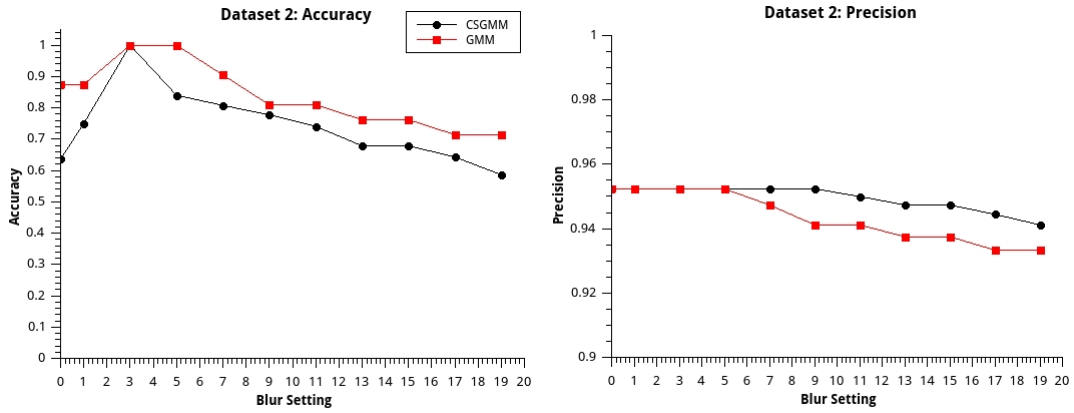

Figure 5.7: Dataset 2 Results

The ROC plot for Dataset 2 is shown in Figure 5.8.

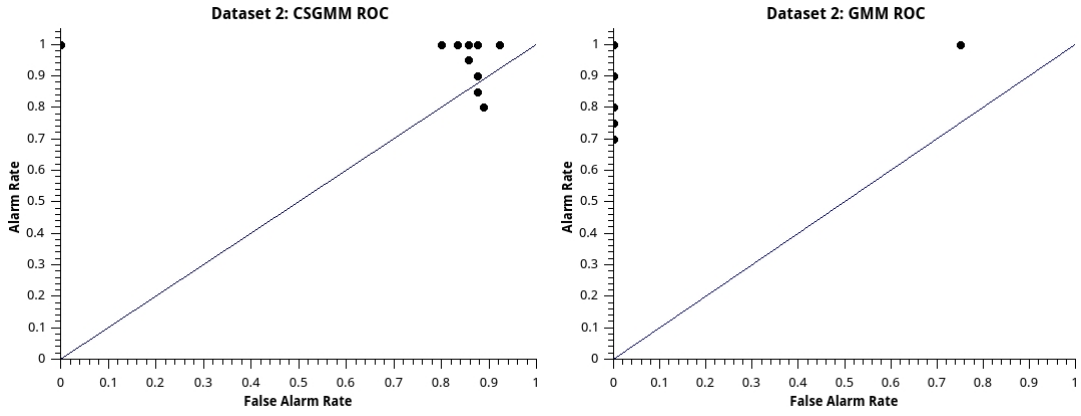

Figure 5.8: Dataset 2 ROC

	Compressive	Non-Compressive	Compressive	Non-Compressive
No Blur			**Blur Size: 11**	
TP	20	20	19	16
TN	1	1	1	1
FP	12	3	6	0
FN	0	0	0	4
Accuracy	0.636	0.875	0.778	0.81
Precision	0.952	0.952	0.952	0.941
Sensitivity	1	1	1	0.8
Specificity	0.077	0.25	0.143	1
Blur Size: 1			**Blur Size: 13**	
TP	20	20	18	15
TN	1	1	1	1
FP	7	3	7	0
FN	0	0	2	5
Accuracy	0.75	0.875	0.679	0.762
Precision	0.952	0.952	0.947	0.938
Sensitivity	1	1	0.9	0.75
Specificity	0.125	0.25	0.125	1
Blur Size: 3			**Blur Size: 15**	
TP	20	20	18	15
TN	1	1	1	1
FP	0	0	7	0
FN	0	0	2	5
Accuracy	1	1	0.679	0.762
Precision	0.952	0.952	0.947	0.938
Sensitivity	1	1	0.9	0.75
Specificity	1	1	0.125	1
Blur Size: 5			**Blur Size: 17**	
TP	20	20	17	14
TN	1	1	1	1
FP	4	0	7	0
FN	0	0	3	6
Accuracy	0.84	1	0.643	0.714
Precision	0.952	0.952	0.9444	0.9333
Sensitivity	1	1	0.85	0.7
Specificity	0.2	1	0.125	1
Blur Size: 7			**Blur Size: 19**	
TP	20	18	16	14
TN	1	1	1	1
FP	5	0	8	2
FN	0	2	4	6
Accuracy	0.808	0.905	0.586	0.714
Precision	0.95	0.947	0.94	0.93
Sensitivity	1	0.9	0.8	0.7
Specificity	0.167	1	0.1	1
Blur Size: 9				
TP	20	16		
TN	1	1		
FP	6	0		
FN	0	4		
Accuracy	0.78	0.81		
Precision	0.952	0.941		
Sensitivity	1	0.8		
Specificity	0.1439	1		

Table 5.7: Dataset 2 Detection Outcomes

5.5.4 Dataset 3 Results

The results of the detection tests against Dataset 3 are shown in Table 5.8. The results are shown graphically in Figure 5.9. The output of each test can be viewed individually - see Appendix F.3.3.

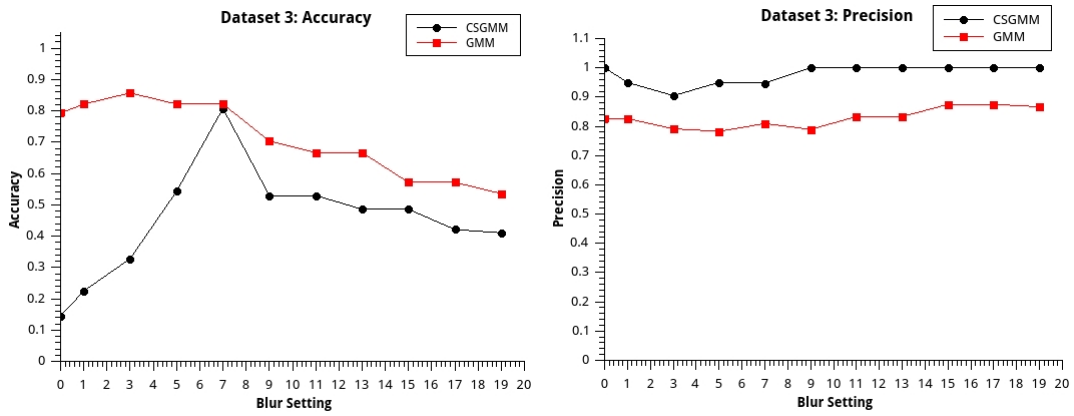

Figure 5.9: Dataset 3 Results

The ROC plot for Dataset 3 is shown in Figure 5.10.

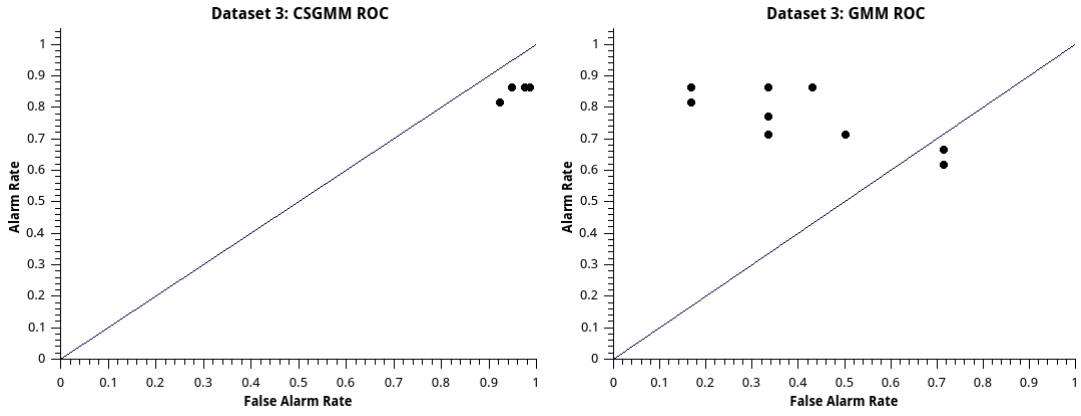

Figure 5.10: Dataset 3 ROC

	Compressive	Non-Compressive	Compressive	Non-Compressive
No Blur			**Blur Size: 11**	
TP	19	19	18	15
TN	0	4	0	3
FP	109	3	12	3
FN	3	0	4	6
Accuracy	0.145	0.793	0.529	0.667
Precision	1	0.826	1	0.83
Sensitivity	0.864	0.864	0.82	0.714
Specificity	0	0.57	0	0.5
Blur Size: 1			**Blur Size: 13**	
TP	19	19	17	15
TN	1	4	0	3
FP	66	2	13	3
FN	3	3	5	6
Accuracy	0.225	0.821	0.486	0.762
Precision	0.95	0.826	1	0.83
Sensitivity	0.864	0.864	0.773	0.714
Specificity	0.015	0.667	0	0.5
Blur Size: 3			**Blur Size: 15**	
TP	19	19	17	14
TN	2	5	0	2
FP	36	1	13	5
FN	3	3	5	7
Accuracy	0.35	0.857	0.486	0.57
Precision	0.905	0.792	1	0.875
Sensitivity	0.864	0.864	0.773	0.667
Specificity	0.053	0.833	0	0.286
Blur Size: 5			**Blur Size: 17**	
TP	19	18	16	14
TN	1	5	0	2
FP	38	1	16	5
FN	3	4	6	7
Accuracy	0.328	0.821	0.421	0.571
Precision	0.95	0.783	1	0.875
Sensitivity	0.864	0.818	0.727	0.167
Specificity	0.026	0.833	0	0.286
Blur Size: 7			**Blur Size: 19**	
TP	18	17	16	13
TN	1	4	0	2
FP	12	2	17	5
FN	4	5	6	8
Accuracy	0.543	0.75	0.41	0.536
Precision	0.947	0.8095	1	0.867
Sensitivity	0.82	0.77	0.73	0.62
Specificity	0.077	0.667	0	0.286
Blur Size: 9				
TP	18	15		
TN	0	4		
FP	12	2		
FN	4	6		
Accuracy	0.529	0.704		
Precision	1	0.789		
Sensitivity	0.82	0.71		
Specificity	0	0.667		

Table 5.8: Dataset 3 Detection Outcomes

5.5.5 Detection Test Discussion

The tests show a degree of consistency between the methods and datasets which was in agreement with initial theories. The experiments utilising Datasets 1 and 2 show similarity between both image segmentation methods in terms of noise tolerance and detection accuracy. The test results show that both methods are able to be configured to completely reject all spurious events and observe all desired events. For both methods, a blur setting of three yielded optimum results. Lower blur settings yielded false positives. In converse, higher blur settings yielded a mixture of false positives and false negatives. Such an outcome is intuitive, as applying a significant amount of blurring to an image can potentially reduce the impact of events. Conversely, applying too little blur can lead to noise events being detected. The results yielded in all of the tests support this hypothesis. As evidenced in the graphs for accuracy and precision, there is a definite relationship between the amount of blur applied and the outcomes of detection. The accuracy and precision graphs for each of the three datasets show similar relationships for each of the methods. For both models, the accuracy and precision peaked with a setting of 3. Under ideal conditions, the non-compressive GMM showed a higher, and more steady level of precision and accuracy over the CSGMM. However, under typical circumstances, the findings presented by these experiments support the statements regarding in the literature (Shen et al. 2012). In fact, under typical and ideal conditions, the CSGMM demonstrated a higher degree of precision. In terms of accuracy, the GMM presented a more stable and predictable relationship, whilst the CSGMM was more varied.

Whilst the results from Datasets 1 and 2 show a comparable level of tolerance, the results from Dataset 3 show that the compressive sensing GMM has a significantly decreased level of tolerance to environmental noise. The non-compressive GMM was able to successfully operate with minimal spurious triggers, however the changes in environmental conditions caused events which would be otherwise detected to be obfuscated. Similar to previous tests, the optimum performance was achieved utilising a blur filter element size of three. This yielded one false positive coupled with three false negatives. The best case put forward by the compressive sensing model was 12 false positives, 1 true negative and no false positives or negatives. Again, this was achieved with a blur

filter setting of 3.

Datasets 1 and 2 both offered accuracy and precision for both models in the order of 90%. However, for both models, Dataset 3 showed an expected reduction in these results.

The ROC plots show that the system is able to intelligently classify events. However, with incorrect tuning or challenging conditions, the system is no more accurate than a random classification. As such, the tuning of these parameters will be required to be an ongoing process upon commissioning at a site. Even in cases where large amounts of spurious events were detected (Dataset 3), the compressive sensing model demonstrated greater capacity for detecting valid events over the non-compressive model. However, under the conditions presented by Dataset 3, both systems lost the ability to intelligently discern events.

Both models operated with similar performance in the case of gradual lighting changes. However, large lighting changes which caused the Gaussian components to invert suddenly were responsible for many of the false events detected. This was particularly prevalent in the compressive sensing model. Throughout each of the datasets, the level of kinetic noise (represented by patches of vegetation moving in the breeze) varied and gradually increased. Dataset 3 was the most challenging in this regard, as there was a high level of wind with mobile vegetation centred directly in frame. By observing the test data, it was determined that whilst the compressive sensing GMM does offer a degree of tolerance, the non-compressive GMM provided a higher level of tolerance given the correct blur filtering parameters. However, under challenging circumstances, the GMM is more prone to missing valid events.

In all tests, some events did not trigger immediately or experienced periods where the detection suddenly ceased. This was largely due to distant objects which were not well defined, or objects which were surrounded by regions of similar intensity (essentially blending in to the background). An increase in sensitivity to small objects, by altering the morphological density filter settings, could result in an increase in undesired events. However, the use of the three datasets showed that a consistent density filter setting enabled the frame processor to conservatively discern valid events from the

incoming frames. Intuitively, the detection latency and consistency on objects which were comparatively closer to the camera surpassed that of objects which were not close. Intuitively, the results show that there was a degree of correlation in this regard with respect to small objects. The range for reliable detection of small objects was decreased when compared with larger objects. The outcomes of the tests performed on Dataset 2 illustrate that whilst people were detected, the detection process certainly favoured vehicles and people which were situated closer to the camera. Objects which were further away from the camera were not registered in the optical flow field as quickly, or were registered sporadically.

The use of a Gaussian Mixtures Model which relies upon only planar intensity rather than individual colour channels is advantageous with respect to speed, however, this performance increase is a trade off with sensitivity. As expected, some objects did not register well when juxtaposed with background regions of similar intensity. This was not detrimental to the detection of objects under these test conditions, however under some circumstances these phenomena may allow for events to be missed, partially or in their entirety. As a further enhancement, the extension of the GMM implementations to accommodate multichannel images may aid more effective segmentation.

From the results presented and analysed, it can be noted that both methods are prime candidates for use in the target application. The non-compressive GMM provided a high level of event rejection, which in some cases rejected acceptable events. The CS-GMM provided a superior detection rate, with acceptable rejection rates under nominal conditions.

5.6 System Integration Test

5.6.1 Methodology

As a final test, it was necessary to ensure the operation of the system under deployment conditions. To simulate a deployment, the vision software was amalgamated with the configured supervisor script. The aim of this test was not to determine the effectiveness

of the supervisor script, but to ensure that the entire system would operate effectively in a headless manner. The system was started and was successfully managed by the supervisor software. The recorded outcomes were also successfully transferred to a remote machine via rsync.

To recreate the scenario of a software crash, the vision software was manually stopped and the system observed to ensure that the supervisor script successfully restarted the system. The network connection monitoring capabilities of the script were also tested.

The remote administration capabilities provided by OpenSSH were also tested.

5.6.2 Outcomes

Overall, the outcomes of this test were successful. However, the tuning of the timing parameters used to schedule the automatic supervision script may require adjustment as deemed appropriate for deployment.

The rsync method for synchronising the output files worked well, and data was returned quickly and efficiently. The system was able to be remotely administrated via OpenSSH.

In addition, the supervisor script performed well. The vision software was automatically started upon system boot, and a VPN connection was successfully established. The vision software was also successfully restarted on the next supervisor check upon manually stopping the vision software.

Hence, the system operation was deemed satisfactory.

5.7 Chapter Summary

In this chapter, the performance of the two background subtraction methods were compared for speed and accuracy. Additionally, the system operation was tested under self-supervision and was deemed to be successful. The outcomes of the tests recommended that the compressive sensing GMM be included with the software prototype.

Chapter 6

Conclusion

6.1 Chapter Overview

This chapter outlines the recommendations for further research and the continuity of work outside the scope of this project.

6.2 Recommendations

The proposed design for a vision system was deemed a suitable candidate for use in wildlife monitoring applications. The evaluation results showed that the use of a compressive sensing Gaussian Mixtures Model provides detection rates and event rejection in line with reasonable expectations. The system was shown to operate well as an autonomous entity.

The outcomes of the performance evaluation indicated that superior performance can be achieved by utilising the compressive sensing GMM. Although the non-compressive GMM offers superior event rejection, it was deemed too performance intensive to be successfully operated on the Pandaboard. The performance offered by the compressive sensing GMM under ideal and nominal conditions was deemed a sufficient compromise for the performance offered under worst case operating conditions. Thus, it was rec-

ommended that the software prototype be delivered with the CSGMM as the selected image segmentation method. Alternatively, the GMM could be considered should the target device be substituted for a more powerful unit.

Thus, the prototypical software presented in this dissertation was recommended as the configuration for a field trial, pending necessary funding and appropriate permissions. Overall, the goals of this project were met and a suitable prototype was delivered. The source code of the final prototype is enclosed within Appendix C.

6.3 Future Research & Development

The completion of this project leaves the potential for further research and development. The use of a field trial and an eventual site deployment aim to further qualify the design of the system. Pending funding and success of trials further development into a commercial system remains a possibility.

Prior to commercialisation, several improvements which were considered outside the scope of this project may be investigated. The development of a database driven, web browser interface for remote management of the system would increase the accessibility of the system to users. This would require the development of a Remote Procedure Call stack and provide means to monitor and configure the system, as well as providing the incoming data. The system configuration could be further developed in such a way that allows the user to select a level of sensitivity, rather than tuning individual parameters. Furthermore, the parameter selection for parts of the model may be able to (at least in part) be performed automatically.

The ability of the selected hardware and software to be integrated into CSIRO's existing field deployment kit was not explored as part of this project. It is expected that no modification to the system will be required, as the field deployment kit offers standard rail voltages and Internet connectivity via Ethernet. Prior to field trials and eventual site commissioning, the system will be require integration into the field deployment kit and the operational performance tested. Although this was listed as a minor project goal, due to accessibility restrictions this was not explored in depth.

Additionally, recognition of animals within the frame may further eliminate spurious captures. This would entail the development of datasets and a learning mechanism which adds another layer of perception to the system. This would be a very specific targeted outcome and would not be suitable for general surveillance applications. The generalised approach presented in this dissertation could be applied to many other surveillance applications which require motion to be filtered.

Some cosmetic enhancements may also be considered to further improve the user experience offered by the system. The output files, whilst time stamped in their filenames, do not offer an accurate time for each event. Thus, the addition of the ability to time stamp individual frames would be an advantageous feature for extension of the system. Further research work may also entail the expansion of the system to support multiple cameras, configured as a master-slave node network, similar to the sensor networks discussed within the literature. This would entail the development of further software to manage the system, but would still utilise the base vision system to perform the observational tasks.

6.4 In Summary

This dissertation proposes a prototypical method for autonomous wildlife monitoring encompassing the use of computer vision and sensor network ideals. It is hoped that the outcomes presented in this thesis provide basis for further development and commissioning at sites of interest.

The process of development of the vision system involved the investigation of current literature surrounding wildlife monitoring. Upon identification of limitations of current methods, suitable signal processing techniques were investigated. The suitability of a Gaussian Mixtures Model, configured to perform segmentation at a locality level. The potential speed increase offered by exploiting the compressibility within these localities was explored and deemed to offer suitable performance in terms of speed and accuracy. The signal processing methodology outlined demonstrates the system is able to perform the task intelligently, and provides a significant degree of improvement in terms of

accessibility, user experience, performance and expandability over existing monitoring methods.

Overall, the goals and aims for the project were suitably satisfied.

References

ARM (2012), 'ARM Infocenter', http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337i/ch01s05s01.html#. [Online; accessed 14 September 2012].

Bond, A. R. & Jones, D. N. (2008), 'Temporal trends in use of fauna-friendly underpasses and overpasses', *Wildlife Research* **35**, pp. 103 – 112.

Boost (2012), 'Boost C++ Libraries', http://www.boost.org. [Online; accessed 23 Sept 2012].

Bradski, G. (2000), 'The OpenCV Library', *Dr. Dobb's Journal of Software Tools* .

Brown, J. & Gehrt, S. (2009), 'The basics of using remote cameras to monitor wildlife.', http://ohioline.osu.edu/w-fact/pdf/0021.pdf. [Online; accessed 10 March 2012].

Butcher, G. S., Fuller, M. R., McAllister, L. S. & Geissler, P. H. (1990), 'An Evaluation of the Christmas Bird count for Monitoring Population trends of Selected Species', *Wildlife Society Bulletin* **18**(2), pp. 129–134.

Canonical Ltd. (2012), 'Ubuntu', http://www.ubuntu.com. [Online; accessed 23 Sept 2012].

Cheung, S.-C. S. & Kamath, C. (2004), Robust techniques for background subtraction in urban traffic video, Vol. 5308, SPIE, pp. 881–892.

Computational Vision Group, R. U. (2001), 'PETS2001 Datasets', http://www.cvg.cs.rdg.ac.uk/PETS2001/pets2001-dataset.html. [Online; accessed 31 August 2012].

Dar-Shyang, L. (2005), 'Effective Gaussian mixture learning for video background subtraction', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(5), pp. 827 –832.

Engineers Australia (2000), 'Code of Ethics', `http://www.engineersaustralia.org.au/sites/default/files/shado/AboutUs/Overview/Governance/CodeOfEthics2000.pdf`. [Online; accessed 21 March 2012].

Engineers Australia (2012), 'Sustainability resources', `http://www.engineersaustralia.org.au/environmental-college/sustainability`. [Online; accessed 9 May 2012].

Farneback, G. (2003), Two-Frame Motion Estimation Based on Polynomial Expansion, *in* 'Image Analysis', Vol. 2749 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 363–370.

Ganick, A. & Little, T. (2009), 'Remote Wildlife Observation with IP Cameras: The Great Point Lighthouse Camera', *MCL Technical Report* (09-01-2009).

Gonzalez, R. C. & Woods, R. E. (2006), *Digital Image Processing (3rd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Hand, D. J. (2009), 'Measuring classifier performance: a coherent alternative to the area under the ROC curve', *Machine Learning* **77**(1), pp. 103–123.

Harris, R. L. & Nicol, S. C. (2010), 'The effectiveness of hair traps for surveying mammals: results of a study in sandstone caves in the tasmanian southern midlands', *Australian Mammology* **32**, pp. 62–66.

Jones, M. E. (2000), 'Road upgrade, road mortality and remedial measures: impacts on a population of eastern quolls and Tasmanian devils', *Wildlife Research* **27**, pp. 289–296.

Langbein, J., Scheibe, K. M., Eichhorn, K., Lindner, U. & Streich, W. (1996), 'An activity-data-logger for monitoring free-ranging animals', *Applied Animal Behaviour Science* **48**(1-2), pp. 115–124.

Linux TV (2011), 'Video4Linux utils', `http://linuxtv.org/projects.php`. [Online; accessed 23 Sept 2012].

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. & Anderson, J. (2002), Wireless sensor networks for habitat monitoring, *in* 'Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications', WSNA '02, ACM, New York, NY, USA, pp. 88–97.

Mata, C., Hervs, I., Herranz, J., Surez, F. & Malo, J. (2008), 'Are motorway wildlife passages worth building? Vertebrate use of road-crossing structures on a Spanish motorway', *Journal of Environmental Management* **88**(3), pp. 407 – 415.

Otsu, N. (1979), 'A Threshold Selection Method from Gray-level Histograms', *IEEE Transactions on Systems, Man and Cybernetics* **9**(1), 62–66.

Pandaboard (2012), 'Pandboard References', `http://pandaboard.org/content/resources/references`. [Online; accessed 23 Sept 2012].

Parks, D. (2011), 'Background Subtraction Library', `http://dparks.wikidot.com/source-code`. [Online; accessed 15 May 2012].

Python Software Foundation (2012), 'Python Programming Language', `http://www.python.org`. [Online; accessed 23 Sept 2012].

Reynolds, D. (2008), 'Gaussian Mixture Models', *Encyclopedia of Biometric Recognition* pp. 12 – 17.

Shen, Y., Hu, W., Yang, M., Wei, B. & Chou, C. T. (2012), 'Efficient Background Subtraction for Real-time Tracking in Embedded Camera Networks', *The 10th ACM Conference on Embedded Networked Sensor Systems (SenSyS)* . To appear.

Stauffer, C. & Grimson, W. (1999), Adaptive background mixture models for real-time tracking, *in* 'Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.', Vol. 2, pp. 637–663.

Taylor, B. D. & Goldingay, R. L. (2003), 'Cutting the carnage: wildlife usage of road culverts in north-eastern New South Wales', *Wildlife Research* **30**, pp. 529–537.

The GNU Project (2010), 'GNU Screen', `http://www.gnu.org/software/screen`. [Online; accessed 23 Sept 2012].

The OpenBSD Project (2012), 'OpenSSH', `http://www.openssh.com`. [Online; accessed 23 Sept 2012].

Titterington, D. M. (1984), 'Recursive Parameter Estimation Using Incomplete Data', *Journal of the Royal Statistical Society. Series B (Methodological)* **46**(2), pp. 257–267.

Tridgell, A. and Mackerras, P. (1996), The rsync algorithm, Technical report, Department of Computer Science, The Australian National University.

Zivkovic, Z. (2004), Improved adaptive Gaussian mixture model for background subtraction, *in* 'Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on', Vol. 2, pp. 28–31.

Zivkovic, Z. & van der Heijden, F. (2006), 'Efficient adaptive density estimation per image pixel for the task of background subtraction', *Pattern Recognition Letters* **27**(7), pp. 773 – 780.

Appendix A

Project Specification

ENG 4111/2 Research Project

Project Specification

For:	**Ashton Fagg**
Program:	BEng, Computer Systems
Topic:	A Linux Based Multimedia Node for Wildlife Monitoring
Supervisors:	Assoc Prof John Leis (USQ)
	Dr Navinda Kottege (CSIRO ICT Centre)
Sponsorship:	CSIRO Autonomous Systems Laboratory
Project Aim:	Design an autonomous sensor system to monitor fauna activity through fixed crossing structures.

1. Identify shortcomings with existing monitoring methods.
2. Investigate methods of reducing spurious captures whilst successfully capturing wildlife motion.
3. Design storage system, implement at least a rudimentary remote monitoring/retrieval method (node heartbeat).
4. Design and integrate hardware support systems (embedded PC, camera, power, modem etc) - pending availability.
5. Evaluate and compare performance of background modelling and motion detection algorithms and methods.

As time and resources permit:

1. Test field deployment.
2. Implement a full, database-driven retrieval method.

Agreed:

Student Name: Ashton Fagg

Date:

Supervisor Name: John Leis

Date:

Supervisor Name: Navinda Kottege

Date:

Examiner/Co-Examiner:

Date:

Appendix B

Project Management Errata

B.1 Project Timeline

Task Description	Projected Completion Date
Topic definition	7 March 2012
Formulation of specifications	21 March 2012
Investigation of possible solutions	30 April 2012
Literature review	30 April 2012
Project preliminary report	23 May 2012
Test implementation of processing software (C++)	1 August 2012
Implementation and testing support software, full system integration	11 August 2012
Partial draft dissertation	12 September 2012
Project presentation	Mid-semester break, September 2012
Final performance evaluations	1 October 2012
Final dissertation	25 October 2012

Table B.1: Research timeline

B.2 Risk Assessment

The following table defines levels for possibility of risk occurence:

Level	Code	Description
0	Highly Unlikely	May only occur under exceptional circumstances.
1	Unlikely	May occur at some point, but quite unlikely.
2	Moderate Likely	Statistically should occur at some point.
3	Likely	Will probably occur at some point.
4	Highly Likely	Is expected to occur under most circumstances.

Table B.2: Hazard occurrence likelihood

The following table defines levels for risk severity rating:

Level	Code	Description
0	Insignificant	Very low impact to personnel, progress or environment.
1	Minor	Minor impact to personnel, progress or environment.
2	Moderate	Moderate impact to personnel, progress or environment.
3	Major	Critical level of impact to personnel, progress or environment.
4	Catastrophic	Such an occurrence would result in catastrophic health effects, environmental effects or total breakdown of project progress.

Table B.3: Hazard consequence levels

B.2.1 Hazard Identification: Project Execution

The following risks have been identified as risks to personnel, environment or progress during the completion of the project.

1. Occupation Overuse Syndrome

    - Risks: Extended use of a personal computer can lead to conditions such as tendinitis, Carpal Tunnel Syndrome and other repetitive strain injuries.
    - Mitigation: A correctly set up computer workstation with ergonomically enhanced peripherals shall be used to ensure comfort and safety. During development, regular breaks shall be taken for several minutes and must include standing and walking.

2. Unsuitable Working Conditions

    - Risks: An incorrectly set up workstation can lead to eye strain, headaches and impede productivity.
    - Mitigation: Ensure that the development environment has an appropriate level and type of lighting, and correctly positioned furniture and computer peripherals.

3. Unrelated personal illness, injury or other unforeseen exceptional circumstances

    - Risks: Unexpected bouts of illness, injury or other unforeseen circumstances may lead to a decrease in productivity. A decrease in productivity can lead to stress, and in turn this can cause other conditions.
    - Mitigation: Create a reasonable schedule and adhere where possible to allow for some spare time if needed.

4. Stress

    - Risks: Long hours caused by an unreasonable schedule can cause stress to the operator, which can lead to illness or other conditions.
    - Mitigation: Formulation of a reasonable schedule, healthy diet and exercise, sufficient sleep and regular breaks.

5. Heavy Lifting

    - Risks: Computer equipment can be heavy, and pose a risk to the operator's health if not handled correctly.
    - Mitigation: Follow correct lifting procedures at all times. Always lift with a straight back and seek assistance where required.

6. Radiation Exposure

    - Risks: Electronic equipment can emit small levels of radiation. This is governed by a set of standards to minimise the dose that the user can be exposed to. However, equipment which is damaged or faulty has the potential to emit larger doses which would be considered a health risk to those in the immediate surroundings.
    - Mitigation: Ensure that damaged equipment is not used. Keep potential emitters away from users and ensure that they are not kept in an active state when not necessary.

7. Electric Shock

    - Risks: Electrical equipment can cause electric shock in a number of scenarios. This can pose a varying level of risk to the user if not dealt with correctly.
    - Mitigation: Ensure that damaged equipment is not used. Keep electrical equipment dry. Do not dismantle equipment unless qualified to do so safely. Ensure that all cables are in serviceable condition.

8. Equipment Failure or Loss

    - Risks: Equipment failure or loss can cause significant risk to project progress.
    - Mitigation: Ensure that all documentation is kept in case required by a warranty or insurance claim.

9. Data Loss

    - Risks: Data loss can cause significant impedance to the progress of the project should critical sections of source code or documentation be lost.
    - Mitigation: Use a source code management system (such as git or Subversion) to keep track of changes to the code base. Ensure that proper back ups of the workstation are kept in multiple locations. This should include daily differential backups and weekly snapshots of home directories.

B.2.2 Hazard Identification: Post-Completion

After the completion of the project, the hazards identified include those which have been identified previously. However, in the interests of sustainability it is important to consider the risks to the environment and community in the deployment and eventual disposal of the equipment.

1. Environmental Contamination

    - Risks: As the equipment will contain potentially hazardous materials (such as lead), contamination could occur if batteries are punctured or disposed of incorrectly.
    - Mitigation: Ensure that all equipment is disposed of correctly once no longer required. The field deployment kit should be designed to ensure that it is durable as possible to minimise potential damage should a leak occur.

B.2.3 Risk Summary

The hazards identified in the previous sections have been ranked according to likelihood and severity. See the risk matrix in Table 3.3.

Hazard	Likelihood	Severity
Occupational Overuse Syndrome	2	2
Unsuitable Working Conditions	2	2
Unrelated Circumstances	1	3
Stress	4	2
Heavy Lifting	1	4
Radiation Exposure	1	3
Electric Shock	0	4
Equipment Loss or Failure	1	3
Data Loss	2	4
Environmental Contamination	1	4

Table B.4: Risk matrix

Appendix C

Source Code Listing - Vision Software

The following is the source code listing of the vision software. This code will control the camera and perform the necessary operations to manipulate the images and establish if recording is to commence. It is written in C++, and assumes that OpenCV 2.3.1 and Boost C++ 1.4.6 is available on the system. This is the complete prototype, and includes the Compressive Sensing Mixture of Gaussians modelling method. The included Makefile will build for ARM only.

C.1 Makefile

```
# Makefile for furrycap - Pandaboard only
# Ashton Fagg (ash.fagg@csiro.au)

CPP = g++
CPPFLAGS = -O3 -Wall -Wextra -pedantic -std=c++0x `pkg-config --cflags opencv` -ffast-math -
    funroll-loops -mfloat-abi=hard -mfpu=neon -march=armv7-a -mcpu=cortex-a9 -mtune=cortex-a9
TARG = furrycap
SRC = main.cpp \
  frame_process_thread.cpp \
  pretrig.cpp \
  record.cpp \
  record_thread.cpp \
       cs_mog.cpp
OBJS = $(SRC:.cpp=.o)
LDFLAGS = -lboost_thread -lboost_program_options `pkg-config --libs opencv`

$(TARG): $(OBJS)
```

```
    $(CPP) $(CPPFLAGS) -o $(TARG) $(OBJS) $(LDFLAGS)

main.o: ts_buffer.hpp
frame_process_thread.o: ts_buffer.hpp pretrig.hpp cs_mog.hpp
pretrig.o: pretrig.hpp
record.o: record.hpp
record_thread.o: record.hpp pretrig.hpp ts_buffer.hpp
cs_mog.o: cs_mog.hpp

PHONY clean:
  /bin/rm $(OBJS) $(TARG)
install:
  /bin/cp ./furrycap /usr/bin/furrycap
  /bin/cp ../scripts/supervisor.py /usr/bin/supervisor.py
  /bin/cp ../scripts/conf.py /usr/bin/conf.py
  /usr/bin/python /usr/bin/supervisor.py
  /usr/bin/make clean
```

C.2 main.cpp

```
/*****************************************************************
 * main.cpp - main and capture/buffer layer implementation
 * Relies on OpenCV and Boost libraries (program options and threads)
 *
 * Ashton Fagg (ash.fagg@csiro.au) - May 2012
 *
 *****************************************************************/
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <cstdio>
#include <cassert>
#include <boost/program_options.hpp>
#include <boost/thread.hpp>
#include <highgui.h>

#include <cv.h>
#include "ts_buffer.hpp"

using std::cout;
using std::endl;
using std::string;
using std::vector;
namespace po = boost::program_options;

extern int process_main_compressive( bool );
extern int rec_main( int, int );

/*********************************************
 * Capture thread parameters                 *
 *********************************************
 * These are set to give the best balance between *
 * frame rate and sensitivity. The smaller the    *
 * frame, the less sensitive it will be.          *
 *
 * These are adjusted according to the target     *
 * hardware. Select which one using the targets   *
 * defined in the Makefile.                       *
 *                                                *
```

```
 * The suggested parameters are:                    *
 * For Pandaboard, 320 x 240                        *
 * For PC, 640 x 480                                *
 * Turbo mode, 160 x 120. Not recommended for       *
 * production use. Good for demos on the PB but     *
 * won't be as sensitive.                           *
 ************************************************/

int FR_SIZE_W;
int FR_SIZE_H;
int FRAME_RATE = 30;

// Initialise the frame buffer
ts_buffer<cv::Mat> raw_frame_buffer;

/************************************************
 * cap_main() - This is the capture thread itself
 ************************************************/

int cap_main( int cap_dev, int fps )
{
  cout << "[CAP] Starting up...\n";

  assert ( FRAME_RATE % fps == 0 );
  int idx = FRAME_RATE / fps;
  long fr = 0;
  // Set up the capture device
  cv::VideoCapture cap ( cap_dev );
  if ( !cap.isOpened() )
  {
    fprintf ( stderr, "[CAP] Error! Can't open capture device!\n" );
    return 1;
  }
  cap.set( CV_CAP_PROP_FRAME_WIDTH, FR_SIZE_W );
  cap.set( CV_CAP_PROP_FRAME_HEIGHT, FR_SIZE_H );
  cout << "[CAP] Startup OK! Now capturing!\n";
  cout << "[CAP] Capture Frame Size: " << cap.get( CV_CAP_PROP_FRAME_WIDTH )
       << " x " << cap.get( CV_CAP_PROP_FRAME_HEIGHT ) << "\n";
  cv::Mat frame( FR_SIZE_W, FR_SIZE_H, CV_8U );

  // Spin until we get told to stop
  while ( true )
  {
    cap >> frame;
    if ( fr++ % idx == 0 )
      raw_frame_buffer.push( frame.clone() );
  }
  frame.release();
  return 0;
}


#ifdef __ARM_NEON__
// This puts the Cortex chip into runfast mode. Linux should bring this up
// automatically but best to double check.
void enable_runfast()
{
  static const unsigned int x = 0x04086060;
  static const unsigned int y = 0x03000000;
  int r;
  asm volatile
  (
    "fmrx %0, fpscr\n\t"
```

```
    "and  %0, %0, %1\n\t"
    "orr  %0, %0, %2\n\t"
    "fmxr fpscr, %0\n\t"
    : "=r"(r)
    : "r"(x), "r"(y)
  );
}
#endif

/*********************************************************
 *  main()                                               *
 *********************************************************
 * These are the various parameters for the model.       *
 * Rather than using a config file, we can use the       *
 * supervisor script to configure everything automatically *
 * and verify that everything is OK with Python, so we can *
 * trust that these are valid parameters.                *
 *                                                       *
 * rec_path - path to storage for recordings.            *
 * rec_t - Trigger observation time                      *
 * pretrig_t - Pretrigger buffer length                  *
 * fps - frame rate                                      *
 * cam - capture device, i.e. /dev/videoX                *
 *********************************************************/
std::string rec_path;
int rec_t;
int pretrig_t;
int fps;
int cam;
bool targ_pb;
bool targ_pc;
bool targ_demo;
int blur_sz;

// GMM params
int csmog_n;
float csmog_bgauth;
float csmog_df;
float csmog_alpha;
float csmog_sdinit;
float csmog_minstd;


int main ( int argc, char* argv[] )
{

#ifdef __ARM_NEON__
  enable_runfast();
#endif
  try
  {
    // Parse the command line options
    po::options_description desc("Runtime Options");
    desc.add_options()
      ("help", "Show help message")
      ("c", po::value<int>(&cam)->default_value(0),
        "Set the capture device.")
      ("targ_pb",
        po::value<bool>(&targ_pb)->implicit_value(1)->default_value(0),
        "Enable Pandaboard mode.")
      ("targ_pc",
        po::value<bool>(&targ_pc)->implicit_value(1)->default_value(0),
        "Enable PC mode.")
```

```
    ( "targ_demo",
      po::value<bool>( &targ_demo )->implicit_value( 1 )->default_value( 0 ),
      "Enable demo mode." )
    ( "gmm_alpha", po::value<float>( &csmog_alpha )->default_value( 0.05f ),
      "Gaussian Mixture Model, alpha (learning rate)" )
    ( "gmm_bg", po::value<float>( &csmog_bgauth )->default_value( 0.7f ),
      "Gaussian Mixture Model, background threshold" )
    ( "gmm_var", po::value<float>( &csmog_sdinit )->default_value( 18.0f ),
      "Gaussian Mixture Model, initial variance" )
    ( "gmm_minvar", po::value<float>( &csmog_minstd )->default_value( 3.0f ),
      "Gaussian Mixture Model, minimum variance" )
    ( "gmm_comps", po::value<int>( &csmog_n )->default_value( 3 ),
      "Gaussian Mixture Model, number of components" )
    ( "rec_path", po::value<std::string>( &rec_path )->default_value( "/tmp/" ),
      "Recording storage path" )
    ( "pretrig_t", po::value<int>( &pretrig_t )->default_value( 5 ),
      "Pretrigger buffer length." )
    ( "blur", po::value<int>( &blur_sz )->default_value( 3 ),
      "Blur filter element size (must be odd)" )
    ( "rec_t", po::value<int>( &rec_t )->default_value( 5 ),
      "Event observation time." )
    ( "f", po::value<int>( &fps )->default_value( 10 ),
      "Set the sample rate (frame/sec)" );

  po::positional_options_description p;
  po::variables_map vm;
  po::store( po::command_line_parser( argc , argv ).
      options( desc ).positional( p ).run(), vm );
  po::notify( vm );

  if ( vm.count( "help" ) )
  {
    cout << "Usage: furrycap [options]\n";
    cout << desc;
    return 0;
  }
}
catch ( std::exception& e )
{
  cout << e.what() << "\n";
  return 1;
}

// If we have no mode set, assume this is for the Pandaboard
if ( targ_pb == false && targ_pc == false && targ_demo == false )
{
  targ_pb = true;
}
else if ( targ_pb == true && targ_pc == true )
{
  std::cout << "Invalid target configuration!\n";
  return 1;
}
else if ( (targ_pb == true && targ_demo == true) ||
          (targ_pc == true && targ_demo == true)
        )
{
  std::cout << "Well, that's odd. The target setup is conflicting. "
            << "So I'm going to go ahead and override to demo mode. "
            << "To avoid this, check which targets you have set. "
            << "Run ./furrycap --help for info.\n\n";
  targ_pb = false;
  targ_pc = false;
```

```
    targ_demo = true;
  }

  // Configure the frame size
  if ( targ_pb ) FR_SIZE_W = 320, FR_SIZE_H = 240;
  if ( targ_pc ) FR_SIZE_W = 640, FR_SIZE_H = 480;
  if ( targ_demo ) FR_SIZE_W = 160, FR_SIZE_H = 120;

  cout << "\nFurryCap v1.0 - By Ashton Fagg (ash.fagg@csiro.au)\n\n";
  cout << "---------------Context Information-----------------\n\n";
  cout << "Target Device: ";
  if ( targ_pb ) cout << "Pandaboard\n";
  if ( targ_pc ) cout << "PC\n";
  if ( targ_demo ) cout << "Demo Mode\n";
  cout << "Sample Rate: " << fps << " Hz\n";
  cout << "Recorder storage: " << rec_path <<"\n\n";
  cout << "\n---------------------------------------------------\n\n";

  // Thread control
  boost::thread_group threads;
  boost::thread *cap = new boost::thread( &cap_main, cam, fps );
  threads.add_thread( cap );
  boost::thread *proc_main = new boost::thread(&process_main_compressive, false);
  threads.add_thread( proc_main );
  boost::thread *r_main = new boost::thread( &rec_main, fps, FRAME_RATE );
  threads.add_thread( r_main );
  threads.join_all();

  return 0;
}
```

C.3 ts_buffer.hpp

```
/*****************************************************************************
 * ts_buffer.hpp - Thread-safe buffer implementation
 * This is used as a template class for the various buffers needed.
 *
 * Ashton Fagg (ashton@fagg.id.au) - May 2012
 *
 *****************************************************************************/
#ifndef TS_BUFFER_H
#define TS_BUFFER_H

#include <queue>
#include <boost/thread.hpp>

template<typename T>
class ts_buffer
{
  private:
    std::queue<T> payload;
    mutable boost::mutex m;
    boost::condition_variable cond;
  public:
    void push ( T const& data )
    {
      boost::mutex::scoped_lock lock ( m );
      payload.push ( data );
      lock.unlock ();
      cond.notify_one ();
```

```
    }
    bool is_empty ( void ) const
    {
      boost::mutex::scoped_lock lock ( m );
      return payload.empty ();
    }
    bool try_get ( T& val )
    {
      boost::mutex::scoped_lock lock ( m );
      if ( payload.empty () )
        return false;
      val = payload.front ();
      payload.pop ();
      return true;
    }
    void get( T& val )
    {
      boost::mutex::scoped_lock lock ( m );
      while ( payload.empty () )
      {
        cond.wait ( lock );
      }
      val = payload.front ();
      payload.pop ();
    }
};
#endif
```

C.4 frame_process_thread.cpp

```
/*****************************************************************************
 * frame_process_thread.cpp - frame processing thread and support functions.
 * Performs the processing and motion detection in conjunction with maintaining
 * the Gaussian Mixtures Model.
 *
 * Ashton Fagg (ash.fagg@csiro.au) - May 2012
 *****************************************************************************/

#include <iostream>
#include <ctime>
#include "cv.h"
#include "cxcore.h"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include "ts_buffer.hpp"
#include "motion.hpp"
#include "pretrig.hpp"
#include "record.hpp"
#include "cs_mog.hpp"

using std::cout;
using std::endl;

extern ts_buffer<cv::Mat> raw_frame_buffer;
extern ts_buffer<cv::Mat> rec_buffer;
extern int FR_SIZE_W;
extern int FR_SIZE_H;
extern int FRAME_RATE;
extern int rec_main( int, int );
```

```
#define WARM_UP_FRAMES 100

/*************************************************************
 * Recording parameters                                      *
 *************************************************************
 * rec_t - total recording time, inc pretrigger (seconds)    *
 * pretrig_t - pretrigger buffering time (seconds)           *
 *************************************************************/

extern int rec_t;
extern int pretrig_t;
extern int FR_SIZE_W;
extern int FR_SIZE_H;

// This dumps the pretrigger buffer.
void pretrigger_buffer_dump( PretrigBuffer& buff, ts_buffer<cv::Mat>& rec )
{
  cv::Mat fr;
  while ( buff.slots_used() > 0 )
  {
    fr = buff.get();
    rec.push( fr );
  }
  fr.release();
}

/*************************************************************
 * Farneback optical flow parameters                         *
 *************************************************************/

const float pyr_sc = 0.5;
const float levels = 3;
const float win_sz = 15;
const float iter = 3;
const float poly_order = 5;
const float poly_sigma = 1.2;

// This checks to see if there is motion within the current frame with respect
// to the previous frame. We don't care about any other information this may
// tell us, we only worry about non-zero pixel velocities.
bool trig( cv::Mat& curr, cv::Mat& prev, cv::Mat& motion_vectors )
{
  int i, j;
  std::vector<cv::Mat> chans;
  cv::calcOpticalFlowFarneback( prev, curr, motion_vectors,
                                pyr_sc, levels, win_sz, iter,
                                poly_order, poly_sigma , 0 );
  cv::split(motion_vectors, chans);

  // Test to see if there are non-zero motion vectors
  for (i = 0; i < chans.at(0).rows; i++)
    for (j = 0; j < chans.at(0).cols; j++)
    {
      if (chans.at(0).at<float>(i,j) != 0 || chans.at(1).at<float>(i,j) != 0)
        return true;
    }
  return false;
}
////////////////////////////////////////////////////////////////////////
// Frame processor parameters
extern int csmog_n;
extern float csmog_bgauth;
```

```
extern float csmog_alpha;
extern float csmog_sdinit;
extern float csmog_minstd;
extern float csmog_df;

extern int blur_sz;



////////////////////////////////////////////////////////////////////////
// This frame processor utillises the compressive Gaussian Mixtures model

int process_main_compressive( bool verbose )
{
  assert(blur_sz % 2 != 0); // blur size must be odd

  cv::Mat in, in_cs, cs_out, eroded, dilated, prev, mvecs;
  PretrigBuffer pt_buff( pretrig_t * 10 );
  double dt = cv::getTickFrequency();
  long start_ticks, jiffies;
  long test_ticks;
  double total_jiffies = 0;
  long fr = 0;
  long stop_fr = 0;
  bool trigger = false;

  // Set up the density filters
  cv::Mat erode_elem = cv::getStructuringElement(cv::MORPH_RECT,
                                                 cv::Size(4, 4),
                                                 cv::Point(3, 3));

  // Instantiate the Compressive Sensing MoG.
  CSMOG::CSGMM *mog = new CSMOG::CSGMM;
  if ( !CSMOG::init( FR_SIZE_W, FR_SIZE_H, csmog_n,
                     csmog_bgauth, csmog_df, csmog_alpha,
                     csmog_sdinit, csmog_minstd, mog ))
  {
    std::cout << "[PROC] CSMOG Init Error!\n";
    return 1;
  }

  // Set up the matrices
  in_cs.create( cv::Size(FR_SIZE_W, FR_SIZE_H), CV_8U );
  cs_out.create( cv::Size(FR_SIZE_W, FR_SIZE_H), CV_8U );
  in.create( cv::Size(FR_SIZE_W, FR_SIZE_H), CV_8U );

  while (true)
  {
    raw_frame_buffer.get( in );
    start_ticks = cv::getTickCount();
    cv::cvtColor( in, in_cs, CV_BGR2GRAY );
    cv::GaussianBlur(in_cs, in_cs, cv::Size(blur_sz, blur_sz), 5);
    if ( CSMOG::process_frame( (uint8_t*) in_cs.data, 1, FR_SIZE_W,
          (uint8_t*) cs_out.data,1, FR_SIZE_W, mog ) )
    {
      cv::erode(cs_out, eroded, erode_elem);
      cv::dilate(eroded, dilated, erode_elem);

      if (++fr < WARM_UP_FRAMES)
      {
        trigger = false;
      }
      else if (fr >= WARM_UP_FRAMES)
```

```
      {
        if (fr == WARM_UP_FRAMES) std::cout << "[PROC] Now motion filtering.\n";
        test_ticks = cv::getTickCount();
        if (!trigger) trigger = trig(dilated, prev, mvecs);
        test_ticks = cv::getTickCount() - test_ticks;
        //std::cout << dt / (double) test_ticks << std::endl;
      }
      // This is the frame control state machine. If there is no motion we
      // simply place the frame on the pretrigger buffer. If not, control the
      // recording time and place frames on the recording buffer.
      switch (trigger)
      {
        case false: pt_buff.put(in);
                    break;
        case true: if ( stop_fr == 0 )
                   {
                     stop_fr = rec_t * 10 + fr;
                     pretrigger_buffer_dump(pt_buff, rec_buffer);
                   }
                   rec_buffer.push(in);
                   if ( fr == stop_fr )
                   {
                     trigger = false;
                     stop_fr = 0;
                   }
                   break;
      }

      jiffies = cv::getTickCount() - start_ticks;
      total_jiffies += (dt / (double) jiffies);
      if (fr % 100 == 0 )
      {
        double tmp = (double) total_jiffies / 100;
        std::cout << "[PROC] 100 Frame Average FPS: " << tmp << std::endl;
        total_jiffies = 0;
      }
    }
    else
    {
      std::cout << "[PROC] Frame process error!\n";
      return 1;
    }
    prev = dilated.clone();
  }

  in.release();
  in_cs.release();
  cs_out.release();

  return 0;
}
```

C.5 record_thread.cpp

```
/*************************************************************
 * record_thread.cpp - Recorder thread.
 *
 * Ashton Fagg (ash.fagg@csiro.au) - July 2012
 *************************************************************/
```

```
#include "cv.h"
#include "highgui.h"
#include "record.hpp"
#include "pretrig.hpp"
#include "ts_buffer.hpp"
#include <boost/thread.hpp>

/*************************************************************
 * Recording parameters                                      *
 *************************************************************
 * rec_t - total recording time, inc pretrigger (seconds)    *
 * pretrig_t - pretrigger buffering time (seconds)           *
 * rec_path - path to the recording storage                  *
 * rec_buffer - recording buffer                             *
 *************************************************************/
extern int rec_t;
extern int pretrig_t;
extern int FR_SIZE_W;
extern int FR_SIZE_H;

extern std::string rec_path;
ts_buffer<cv::Mat> rec_buffer;
extern boost::barrier rec_bar;

/*****************************************************************************/
int rec_main( int fps, int actual )
{
  int i;
  int idx;
  assert( actual % fps == 0 );
  idx = actual / fps;
  // Initialise the event recorder
  EventRecorder recorder( actual, cv::Size( FR_SIZE_W, FR_SIZE_H ),
                          rec_path, true );
  cv::Mat frame;
  std::cout << "[REC] Started and ready!\n";
  while ( true )
  {
    rec_buffer.get( frame );
    for ( i = 0; i < idx; i++ )
      if ( !recorder.rec_frame( frame ) )
      {
        std::cout << "[REC] Recorder Error! Exiting!\n";
        return 1;
      }
    frame.release();
  }
  return 0;
}
```

C.6 record.hpp

```
/*****************************************************************************
 * record.hpp - record helper structures
 *
 * Ashton Fagg (ash.fagg@csiro.au) - July 2012
 *****************************************************************************/
#include <cv.h>
#include <highgui.h>
```

```
#ifndef RECORDER_H
#define RECORDER_H

#define CODEC CV_FOURCC('D','I','V','X')

class EventRecorder
{
  public:
    EventRecorder( int, cv::Size, std::string, bool );
    bool rec_frame( cv::Mat& );
  private:
    bool first_frame;
    std::string path, filename;
    std::string curr_date, curr_hour;
    int fps;
    bool is_colour;
    cv::Size frame_sz;
    cv::VideoWriter writer;

    void configure( void );
    void verify_path( void );
    const std::string get_curr_date( void );
    const std::string get_curr_hour( void );
};
#endif
```

C.7 record.cpp

```
/*****************************************************************************
 * recorder.cpp - Recorder support functions
 *
 * Ashton Fagg (ash.fagg@csiro.au) - July 2012
 *****************************************************************************/

#include "record.hpp"
#include "pretrig.hpp"
#include <cv.h>
#include <highgui.h>
#include <time.h>

EventRecorder::EventRecorder( int _fps, cv::Size _frame_sz,
                              std::string _path, bool _is_colour )
{
  fps = _fps;
  is_colour = _is_colour;
  path = _path;
  frame_sz = _frame_sz;
  first_frame = true;
}

const std::string EventRecorder::get_curr_date( void )
{
  time_t now = time( 0 );
  struct tm tstruct;
  char buf[80];
  tstruct = *( localtime( &now ) );
  strftime( buf, sizeof(buf), "%Y-%m-%d", &tstruct );
  return buf;
}
```

```
// Return the current hour
const std::string EventRecorder::get_curr_hour( void )
{
  time_t now = time( 0 );
  struct tm tstruct;
  char buf[80];
  tstruct = *( localtime( &now ) );
  strftime( buf, sizeof(buf), "%H", &tstruct );
  return buf;
}

// Strips any trailing / from the path
void EventRecorder::verify_path( void )
{
  std::string::iterator it;
  it = path.end() - 1;
  if ( *it == '/' ) path.erase( it );
}

// This configures the VideoWriter class automatically to
// put hourly segments into corresponding seperate files.
void EventRecorder::configure( void )
{
  if ( first_frame )
  {
    curr_date = EventRecorder::get_curr_date();
    curr_hour = EventRecorder::get_curr_hour();
    EventRecorder::verify_path();
    filename.append( path );
    filename.append( "/FURRYCAP_" );
    filename.append( curr_date );
    filename.append( "_" );
    filename.append( curr_hour );
    filename.append( ".avi" );
    writer.open( filename, CODEC, fps, frame_sz, is_colour );
  }
  else
  {
    // Check to make sure we want to write to the same file.
    std::string tmp_date, tmp_hour;
    tmp_date = EventRecorder::get_curr_date();
    tmp_hour = EventRecorder::get_curr_hour();

    if ( tmp_date != curr_date || tmp_hour != curr_hour )
    {
      // Reconfigure the VideoWriter class
      curr_date = tmp_date;
      curr_hour = tmp_hour;
      filename.clear();
      filename.append( path );
      filename.append( "/FURRYCAP_" );
      filename.append( curr_date );
      filename.append( "_" );
      filename.append( curr_hour );
      filename.append( ".avi" );
      writer.open( filename, CODEC, fps, frame_sz, is_colour );
    }
  }
  first_frame = false;
}

bool EventRecorder::rec_frame( cv::Mat& f )
{
```

```
  EventRecorder::configure();
  if ( !writer.isOpened() ) return false;
  writer.write( f );
  return true;
}
```

C.8 pretrig.hpp

```
/*****************************************************************************
 * pretrig.hpp - Pre-trigger buffer defintion.
 *
 * This buffer is used to store the last N raw frames, such that we can append
 * them to the front of a file in the event of a trigger. Additionally, this
 * also stores the corresponding frame reference numbers.
 *
 * Ashton Fagg (ash.fagg@csiro.au) - July 2012
 *****************************************************************************/

#ifndef PRETRIG_H
#define PRETRIG_H

#include "cv.h"
#include <utility>
#include <queue>

class PretrigBuffer
{
  public:
    explicit PretrigBuffer( int );          // Buffer with N slots
    ~PretrigBuffer();
    void put( const cv::Mat& );      // Buffer frame and frame number
    cv::Mat get( void );  // Retrieve the frame
    void discard( void );                   // Discard
    int slots_used( void );
    int size( void );
    void reset( void );                     // Reset
    bool is_full( void );                   // Check if there's more room
  private:
    int slots;
    int used_slots;
    int get_idx;
    int put_idx;
    std::queue<cv::Mat> frames;
};

#endif
```

C.9 pretrig.cpp

```
/*****************************************************************************
 * pretrig.c - Pretrigger buffer class implementation
 *
 * Ashton Fagg (ash.fagg@csiro.au) - July 2012
 *****************************************************************************/

#include "pretrig.hpp"
#include <queue>
```

```
PretrigBuffer::PretrigBuffer( int _slots )
{
  if ( _slots <= 0 ) slots = 5;
  else slots = _slots;
  used_slots = 0;
}

PretrigBuffer::~PretrigBuffer( void )
{
  PretrigBuffer::reset();
}

void PretrigBuffer::discard( void )
{
  frames.pop();
  used_slots--;
}

void PretrigBuffer::put( const cv::Mat& _fr )
{
  if ( used_slots < slots )
  {
    frames.push( _fr.clone() );
  }
  else
  {
    PretrigBuffer::discard();
    frames.push( _fr.clone() );
  }
  used_slots++;
}

cv::Mat PretrigBuffer::get( void )
{
  cv::Mat ret = frames.front().clone() ;
  PretrigBuffer::discard();
  return ret;
}


bool PretrigBuffer::is_full( void )
{
  if ( used_slots == slots ) return true;
  else return false;
}

int PretrigBuffer::slots_used( void )
{
  return used_slots;
}

int PretrigBuffer::size( void )
{
  return slots;
}

void PretrigBuffer::reset( void )
{
  while ( PretrigBuffer::slots_used() > 0 )
    PretrigBuffer::discard();
}
```

C.10 cs_mog.hpp

```
/*****************************************************************************************
 * cs_mog.hpp - Compressive Sensing Mixture of Gaussians.
 * Modified for use with ARM and C++ specific features.
 * Ashton Fagg (ash.fagg@csiro.au) - August 2012
 *****************************************************************************************/

#ifndef __CS_MOG_H__
#define __CS_MOG_H__

#include <cstdint>
#ifdef __ARM_NEON__
#include <arm_neon.h>
#endif

#define MOG_NUMGAU      3
#define MOG_BGGAUTH     0.7f
#define MOG_DF          3.0f
#define MOG_ALPHA       0.05f
#define MOG_SDINIT      18.0f
#define MOG_MINSTD      3.0f
#define MOG_WEIGHTINIT  0.01f
#define MOG_CBCRTH      60
#define MOG_OUTPUT      255 //1 or 255

#define POSTPROCTH      15
#define POSTPROCALPHA   4

#ifndef MIN
#define MIN(x, y) ((x)<(y) ? (x):(y))
#define MAX(x, y) ((x)>(y) ? (x):(y))
#endif

#ifndef SUB
#define SUB(x,y) ((x)-(y))
#define ADD(x,y) ((x)+(y))
#define MULT(x,y) ((x)*(y))
#define DIV(x,y) ((x)/(y))
#endif

namespace CSMOG
{
  typedef struct CSGMM_
  {
    uint32_t _width, _height;
    uint16_t _numGau;
    uint32_t * _rankIndex;
    uint32_t _initWithImage;
    uint32_t _numForegroundPixel;
    float _df;
    float _minStd;
    float _bgGauTh;
    float _alpha;
    float _sdInit;
    float * _weight;
    float * _sd;
    float * _rank;
    float * _mean;

    //Post processing variables
    uint8_t _postProcTh;
    uint8_t _postProcAlpha;
```

```
    //Temp storage variables
    uint8_t * _cIm;     // the current planar intensity image
    int16_t * _csr;     // for storing compressive sensing output, width*heigh/8
    uint8_t * _csmogr; // for storing mog output, width*heigh/8
    uint8_t * _bkgnd;  // background image;
    uint8_t * _csm;     // compressive sensing mask image
#ifdef __ARM_NEON__
    int16x8_t neon_lookup[64];
#endif
  } CSGMM;

  int16_t init_default(uint32_t width, uint32_t height, CSGMM * mog);

  int16_t init(uint32_t width, uint32_t height, uint32_t numGau,
               float bgGauTh,  float df, float alpha,
               float sdInit, float minStd, CSGMM * mog);

  int16_t process_frame(uint8_t * iPtr, int32_t iIStep, int32_t iJStep,
                        uint8_t * aPtr, int32_t aIStep, int32_t aJStep,
                       CSGMM * mog);

  void destroy(CSGMM * mog);

  void csProjection(uint8_t * iPtr, uint32_t width, uint32_t height, int16_t * aPtr, CSMOG::CSGMM *mog );

  void postProc(uint8_t * csmogr, uint8_t * cIm, uint8_t* bkgnd, uint32_t width,
                uint32_t height, uint8_t th, uint32_t alpha,
                uint8_t * aPtr, uint32_t aIStep, uint32_t aJStep);

  void convertToIntensity(uint8_t * iPtr, uint32_t iIStep, uint32_t iJStep,
                          uint32_t width, uint32_t height, uint8_t * oPtr);

  // This is the Bernoulli lookup matrix
  const int16_t csc[512] = {
      1,1,1,1,1,-1,1,1,-1,-1,1,-1,1,-1,-1,-1,1,-1,1,-1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,
      1,-1,-1,-1,-1,1,-1,-1,-1,1,1,-1,-1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,-1,1,-1,1,-1,
      1,-1,-1,1,-1,1,1,1,-1,-1,1,-1,-1,1,1,1,1,-1,1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,-1,
      1,-1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,1,1,-1,-1,1,-1,-1,-1,1,-1,1,1,1,1,1,-1,1,1,
      -1,-1,-1,1,1,-1,-1,-1,-1,1,-1,1,-1,1,1,1,-1,1,1,1,1,-1,-1,-1,-1,1,1,-1,1,-1,-1,
      1,1,1,1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,1,-1,1,1,1,-1,1,
      -1,-1,-1,-1,-1,1,-1,1,-1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,
      -1,1,-1,1,1,1,-1,1,1,1,-1,-1,1,-1,-1,-1,1,1,1,1,-1,1,-1,-1,-1,-1,1,-1,1,1,1,1,1,1,
      -1,1,1,-1,1,1,-1,1,-1,-1,1,-1,-1,-1,1,-1,1,1,-1,1,1,-1,-1,1,1,1,-1,-1,-1,-1,
      -1,1,1,1,1,1,-1,1,1,1,-1,-1,1,1,-1,1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,1,1,
      -1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-1,-1,1,1,1,1,1,1,-1,1,-1,-1,-1,1,-1,-1,1,-1,-1,1,-1,
      1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,1,1,-1,1,1,-1,1,1,1,-1,-1,1,1,1,1,-1,-1,-1,
      -1,-1,1,-1,1,-1,1,1,1,-1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,
      1,1,1,-1,1,1,-1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1,-1,1,1,1,1,-1,1,1,-1,1,-1,1,1,
      1,1,-1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,-1,-1,1,1,1,1,1,-1,1,1,1,-1,1,-1,1,1,-1,1,-1,1,-1,
      -1,1,1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,-1,1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,1,-1
  };
#ifdef __ARM_NEON__
  void setup_neon_lookup(CSGMM *mog);
#endif
}
#endif
```

C.11 cs_mog.cpp

```
/*************************************************
 * cs_mog.cpp - Compressive Sensing Mixture of Gaussians
 *
 * The compressive sensing MOG will take advantage of the
 * features provided by an ARM CPU if available.
 *
 * Compile with -march=armv7-a -mfloat-abi = hard -mfpu=neon
 * -mcpu=cortex-a9 -mtune=cortex-a9
 *
 * Ashton Fagg (ash.fagg@csiro.au) - August 2012
 *************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstdint>
#include <cmath>
#include <ctime>
#include <cstring>
#include <cassert>
#include "cs_mog.hpp"

#ifdef __ARM_NEON__
#include <arm_neon.h>
#endif

////////////////////////////////////////////////////////////////////////////////
// init_default() - Initialises a CSGMM object with the default parameters
// outlined in cs_mog.hpp.
////////////////////////////////////////////////////////////////////////////////

int16_t CSMOG::init_default(uint32_t width, uint32_t height, CSMOG::CSGMM * mog)
{
  uint32_t i=0;
  assert( height % 8 == 0 && width % 8 == 0 );
  uint32_t eims = height * width*8/64;

  mog->_width = width;
  mog->_height = height;
  mog->_numGau = MOG_NUMGAU;

  mog->_bgGauTh = MOG_BGGAUTH;
  mog->_df =  MOG_DF;
  mog->_alpha =  MOG_ALPHA;
  mog->_sdInit =  MOG_SDINIT;
  mog->_minStd =  MOG_MINSTD;
  mog->_postProcTh = POSTPROCTH;
  mog->_postProcAlpha = POSTPROCALPHA;
  mog->_initWithImage = 0;
  mog->_numForegroundPixel=0;

  float initW =  (1.0f/ (float) mog->_numGau);

  mog->_weight = new float[ eims * mog->_numGau ]();
  if(!mog->_weight) return 0;

  mog->_sd = new float[ eims * mog->_numGau ]();
  if(!mog->_sd) return 0;

  mog->_mean = new float[ eims * mog->_numGau ]();
  if(!mog->_mean) return 0;

  mog->_rank = new float[ mog->_numGau ]();
```

```
  if(!mog->_rank) return 0;

  mog->_rankIndex = new uint32_t[ mog->_numGau ]();
  if(!mog->_rankIndex) return 0;

  mog->_cIm = new uint8_t[ width * height ]();
  if(!mog->_cIm) return 0;

  mog->_csr = new int16_t[ eims ]();
  if(!mog->_csr) return 0;

  mog->_csmogr = new uint8_t[ eims ]();
  if(!mog->_csmogr) return 0;

  mog->_bkgnd = new uint8_t[ width * height ]();
  if(!mog->_bkgnd) return 0;

  mog->_csm = new uint8_t[ width * height / 64 ]();
  if(!mog->_csm) return 0;

  for(i=0;i< eims*(mog->_numGau);i++)
  {
    mog->_mean[i] = 0.0;
    mog->_weight[i] = initW;
    mog->_sd[i] = mog->_sdInit;
  }
#ifdef __ARM_NEON__
  setup_neon_lookup(mog);
#endif
  return 1; // OK
}

////////////////////////////////////////////////////////////////////////////////
// init() - Initialises a CSGMM with custom parameters.
////////////////////////////////////////////////////////////////////////////////
int16_t CSMOG::init(uint32_t width, uint32_t height, uint32_t numGau,
            float bgGauTh, float df, float alpha,
                    float sdInit, float minStd, CSMOG::CSGMM * mog)
{
  uint32_t i=0;
  mog->_width = width;
  mog->_height = height;
  assert( width % 8 == 0 && height % 8 == 0 );
  mog->_numGau = numGau;
  mog->_bgGauTh = bgGauTh;
  mog->_df = df;
  mog->_alpha = alpha;
  mog->_sdInit =  sdInit;
  mog->_minStd =  minStd;
  mog->_initWithImage = 0;
  mog->_numForegroundPixel = 0;

  mog->_postProcTh = POSTPROCTH;
  mog->_postProcAlpha = POSTPROCALPHA;

  uint32_t eims = height * width*8/64;
  float initW =  (1.0f/ (float) mog->_numGau);

  mog->_weight = new float[ eims * mog->_numGau ]();
  if(!mog->_weight) return 0;

  mog->_sd = new float[ eims * mog->_numGau ]();
  if(!mog->_sd) return 0;
```

```
  mog->_mean = new float[ eims * mog->_numGau ]();
  if(!mog->_mean) return 0;

  mog->_rank = new float[ mog->_numGau ]();
  if(!mog->_rank) return 0;

  mog->_rankIndex = new uint32_t[ mog->_numGau ]();
  if(!mog->_rankIndex) return 0;

  mog->_cIm = new uint8_t[ width * height ]();
  if(!mog->_cIm) return 0;

  mog->_csr = new int16_t[ eims ]();
  if(!mog->_csr) return 0;

  mog->_csmogr = new uint8_t[ eims ]();
  if(!mog->_csmogr) return 0;

  mog->_bkgnd = new uint8_t[ width * height ]();
  if(!mog->_bkgnd) return 0;

  mog->_csm = new uint8_t[ width * height / 64 ]();
  if(!mog->_csm) return 0;

  for(i=0;i<eims*mog->_numGau;i++)
  {
    mog->_mean[i] = 0.0f;
    mog->_weight[i] = initW;
    mog->_sd[i] = mog->_sdInit;
  }
#ifdef __ARM_NEON__
  setup_neon_lookup(mog);
#endif
  return 1; // OK
}

////////////////////////////////////////////////////////////////////////
// This processes the incoming frames and outputs the calculated foreground
// mask.
//
// Parameters: iPtr -> Pointer to input image data (from cv::Mat)
//             iIStep -> Input horizontal byte step
//             iJStep -> Input vertical byte step
//             aPtr ->  Pointer to output image data (from cv::Mat)
//             aIStep -> Output horizontal byte step
//             aJStep -> Output vertical byte step
//
// This calls some other functions which are outlined below.
////////////////////////////////////////////////////////////////////////

int16_t CSMOG::process_frame( uint8_t * iPtr, int32_t iIStep, int32_t iJStep,
                              uint8_t * aPtr, int32_t aIStep, int32_t aJStep,
                             CSMOG::CSGMM * mog)
{

  float *_wPtr = mog->_weight;
  float *_sPtr= mog->_sd;
  float * _mean = mog->_mean;
  float *_rank = mog->_rank;

  float initWeight =  (double)(MOG_WEIGHTINIT);
```

```
uint32_t _numGau = mog->_numGau, _height= mog->_height,
         _width= mog->_width, wh = mog->_height * mog->_width;
uint32_t mog_length = _width*_height/8, wh_64 = wh/64;
float _df = mog->_df, _bgGauTh = mog->_bgGauTh, _alpha = mog->_alpha, _sdInit = mog->_sdInit;
float p = 0.0, _minStd = mog->_minStd;
uint32_t * _rankIndex = mog->_rankIndex;

uint32_t m = 0, i = 0, j = 0,k = 0;

uint32_t * nfp = & (mog->_numForegroundPixel);
int16_t *_csr = mog->_csr, *csrPtr = mog->_csr;
uint8_t *_csmogr = mog->_csmogr, *csmogrPtr = mog->_csmogr;
uint8_t *_cIm = mog->_cIm;
uint8_t *_bkgnd = mog->_bkgnd;
uint8_t *_csm = mog->_csm;
unsigned int matchIndex;
int minWIndex, tempRankIndex, match;
float dist, maxW, sumW, minW, tempRank, dif, lumVal;


convertToIntensity(iPtr, iIStep, iJStep, _width, _height, _cIm);
csProjection(_cIm, _width, _height, _csr, mog);

if(!mog->_initWithImage)
{
  for(i=0;i<mog_length;i++)
  {

    _mean[0] =  (float)(*csrPtr);
    *csmogrPtr = 0;
    _mean += _numGau;
    csrPtr ++;
    csmogrPtr ++;
  }
  memcpy(_bkgnd, _cIm, sizeof(uint8_t)*wh);
  mog->_initWithImage = 1;
  return 1;
}

// update gaussian components for each pixel
for(i=0;i<mog_length;i++)
{
  matchIndex = -1;
  minWIndex = 0, tempRankIndex = 0, match = 0;
  dist = -1.0f, maxW = -1.0f, sumW = 0.0f, minW = 0.0f;
  tempRank = 0.0f, dif = 0.0f, lumVal = 0.0f;

  lumVal= (float)(*csrPtr);

  //first match that has the highest weight
  for(k=0;k<_numGau;k++)
  {
    dif = abs(lumVal - _mean[k]);
    if(dif<_df*MAX(_sPtr[k], _minStd) && _wPtr[k]>maxW)
    {
      match = 1;
      matchIndex = k;
      maxW = _wPtr[k];
      dist = dif;
    }
  }
  if(match==1)
  {
```

```
    for(k=0;k<_numGau;k++)
    {
        _wPtr[k] = (matchIndex == k ) ? (1-_alpha)*_wPtr[k] + _alpha : _wPtr[k] * (1-_alpha);
        if(matchIndex == k && dist < _df*_sPtr[k])
        {
          p = _alpha/_wPtr[k];
          _mean[k] = (1-p)*_mean[k] + p*lumVal;
          _sPtr[k] = sqrt((1-p)*(_sPtr[k])*(_sPtr[k]) +
              p*(lumVal - _mean[k])*(lumVal - _mean[k]));
        }
      sumW +=_wPtr[k];
    }
    // Calculate the component ranking
    for(k=0;k<_numGau;k++)
    {
      _wPtr[k] /=sumW;
      _rank[k] = _wPtr[k]/_sPtr[k];
      _rankIndex[k] = k;
    }
    // Sort the component ranks
    for(k=1;k<_numGau;k++)
    {
      for(m=0;m<k;m++)
      {
        if (_rank[k] > _rank[m])
        {
          tempRank = _rank[m];
          _rank[m] = _rank[k];
          _rank[k] = tempRank;
          tempRankIndex = _rankIndex[m];
          _rankIndex[m] = _rankIndex[k];
          _rankIndex[k] = tempRankIndex;
        }
      }
    }
    sumW = 0.0;
    for(k=0;k<_numGau;k++)
    {
      if (_rankIndex[k] == matchIndex) break;
      sumW += _wPtr[_rankIndex[k]];
    }
    *csmogrPtr = MOG_OUTPUT *(sumW>=_bgGauTh);
  }
  else
  {
    minW = 1000;
    for(k=0;k<_numGau;k++)
    {
      if(_wPtr[k]<minW)
      {
        minW = _wPtr[k];
        minWIndex = k;
      }
    }

    _mean[minWIndex] = lumVal;
    _sPtr[minWIndex] = _sdInit;
    _wPtr[minWIndex] = initWeight;
    *csmogrPtr = MOG_OUTPUT;
  }
  _wPtr += _numGau;
  _sPtr += _numGau;
  _mean += _numGau;
```

```
    csrPtr ++;
    csmogrPtr ++;
  }


  //now we have _csmogr, the outpt of mog, determine which blocks have changed.
  uint8_t * csmPtr = _csm;
  csmogrPtr = _csmogr;
  memset(_csm, 0, sizeof(uint8_t)* wh_64);
  for (i=0; i<wh_64; i++)
  {
    for (j =0; j<8; j++)
    {
      csmPtr[i] = (csmogrPtr[j]>0);
    }
    csmogrPtr+=8;
  }
  // Convert this to an output image.
  CSMOG::postProc(_csm, _cIm, _bkgnd, _width, _height,
                  mog->_postProcTh, mog->_postProcAlpha,
                  aPtr, aIStep, aJStep);

  return 1;
}

////////////////////////////////////////////////
// destroy() - Deallocates a CSGMM object
////////////////////////////////////////////////

void CSMOG::destroy(CSMOG::CSGMM * mog)
{
  delete mog->_weight;
  delete mog->_mean;
  delete mog->_sd;
  delete mog->_rank;
  delete mog->_rankIndex;
  delete mog->_cIm;
  delete mog->_csr;
  delete mog->_csmogr;
  delete mog->_bkgnd;
  delete mog->_csm;
  mog->_initWithImage = 0;
}



////////////////////////////////////////////////////////////////////////////////
// The following are internal routines which are used to perform the compressive
// sensing projection and update the Mixture of Gaussians. Don't call these
// explicitly.
////////////////////////////////////////////////////////////////////////////////
// csProjection() - This performs the Compressive Sensing projection itself. This takes
// advantage of ARM's NEON if we're running on the Pandaboard.
////////////////////////////////////////////////////////////////////////////////

void CSMOG::csProjection(uint8_t * iPtr, uint32_t width, uint32_t height, int16_t * aPtr, CSMOG::CSGMM *mog )
{
  int16_t *cs_result = new int16_t[width*height/64*8];
  int16_t *cs_result_ptr= cs_result;
  uint32_t iIndex = 0, csIndex = 0;

#ifdef __ARM_NEON__
  // Set up the NEON arrays
```

```
int16x8_t proj0, proj1, proj2, proj3, proj4, proj5, proj6, proj7;
int16x8_t res0, res1, res2, res3, res4, res5, res6, res7;
int16_t p0[8], p1[8], p2[8], p3[8], p4[8], p5[8], p6[8], p7[8];
int16_t r0[8], r1[8], r2[8], r3[8], r4[8], r5[8], r6[8], r7[8];
unsigned int csy, csx, csk, i;
int16_t cs_temp;

for (csy = 0; csy < height; csy += 8)
{
  for (csx = 0; csx < width; csx += 8 )
  {
    cs_temp = 0;
    for (csk = 0; csk < 8; csk++)
    {
      //Set the pointer to the block to the first pixel of the first row
      iIndex = csy * width + csx;

      // For each of the 8 projections, we extract the current block and
      // multiply it by the lookup

      // Copy the data to the NEON arrays
      p0[0] = (int16_t) iPtr[iIndex + 0];
      p0[1] = (int16_t) iPtr[iIndex + 1];
      p0[2] = (int16_t) iPtr[iIndex + 2];
      p0[3] = (int16_t) iPtr[iIndex + 3];
      p0[4] = (int16_t) iPtr[iIndex + 4];
      p0[5] = (int16_t) iPtr[iIndex + 5];
      p0[6] = (int16_t) iPtr[iIndex + 6];
      p0[7] = (int16_t) iPtr[iIndex + 7];
      proj0 = vld1q_s16(p0);

      p1[0] = (int16_t) iPtr[width + iIndex + 0];
      p1[1] = (int16_t) iPtr[width + iIndex + 1];
      p1[2] = (int16_t) iPtr[width + iIndex + 2];
      p1[3] = (int16_t) iPtr[width + iIndex + 3];
      p1[4] = (int16_t) iPtr[width + iIndex + 4];
      p1[5] = (int16_t) iPtr[width + iIndex + 5];
      p1[6] = (int16_t) iPtr[width + iIndex + 6];
      p1[7] = (int16_t) iPtr[width + iIndex + 7];
      proj1 = vld1q_s16(p1);

      res0 = vmulq_s16( proj0, mog->neon_lookup[csk*8] );
      vst1q_s16(  r0, res0 );

      p2[0] = (int16_t) iPtr[2*width + iIndex + 0];
      p2[1] = (int16_t) iPtr[2*width + iIndex + 1];
      p2[2] = (int16_t) iPtr[2*width + iIndex + 2];
      p2[3] = (int16_t) iPtr[2*width + iIndex + 3];
      p2[4] = (int16_t) iPtr[2*width + iIndex + 4];
      p2[5] = (int16_t) iPtr[2*width + iIndex + 5];
      p2[6] = (int16_t) iPtr[2*width + iIndex + 6];
      p2[7] = (int16_t) iPtr[2*width + iIndex + 7];
      proj2 = vld1q_s16(p2);

      res1 = vmulq_s16( proj1, mog->neon_lookup[csk*8+1] );
      vst1q_s16(  r1, res1 );

      p3[0] = (int16_t) iPtr[3*width + iIndex + 0];
      p3[1] = (int16_t) iPtr[3*width + iIndex + 1];
      p3[2] = (int16_t) iPtr[3*width + iIndex + 2];
      p3[3] = (int16_t) iPtr[3*width + iIndex + 3];
      p3[4] = (int16_t) iPtr[3*width + iIndex + 4];
      p3[5] = (int16_t) iPtr[3*width + iIndex + 5];
```

```
p3[6] = (int16_t) iPtr[3*width + iIndex + 6];
p3[7] = (int16_t) iPtr[3*width + iIndex + 7];
proj3 = vld1q_s16(p3);

res2 = vmulq_s16( proj2, mog->neon_lookup[csk*8+2] );
vst1q_s16(  r2, res2 );

p4[0] = (int16_t) iPtr[4*width + iIndex + 0];
p4[1] = (int16_t) iPtr[4*width + iIndex + 1];
p4[2] = (int16_t) iPtr[4*width + iIndex + 2];
p4[3] = (int16_t) iPtr[4*width + iIndex + 3];
p4[4] = (int16_t) iPtr[4*width + iIndex + 4];
p4[5] = (int16_t) iPtr[4*width + iIndex + 5];
p4[6] = (int16_t) iPtr[4*width + iIndex + 6];
p4[7] = (int16_t) iPtr[4*width + iIndex + 7];
proj4 = vld1q_s16(p4);

res3 = vmulq_s16( proj3, mog->neon_lookup[csk*8+3] );
vst1q_s16(  r3, res3 );

p5[0] = (int16_t) iPtr[5*width + iIndex + 0];
p5[1] = (int16_t) iPtr[5*width + iIndex + 1];
p5[2] = (int16_t) iPtr[5*width + iIndex + 2];
p5[3] = (int16_t) iPtr[5*width + iIndex + 3];
p5[4] = (int16_t) iPtr[5*width + iIndex + 4];
p5[5] = (int16_t) iPtr[5*width + iIndex + 5];
p5[6] = (int16_t) iPtr[5*width + iIndex + 6];
p5[7] = (int16_t) iPtr[5*width + iIndex + 7];
proj5 = vld1q_s16(p5);

res4 = vmulq_s16( proj4, mog->neon_lookup[csk*8+4] );
vst1q_s16(  r4, res4 );

p6[0] = (int16_t) iPtr[6*width + iIndex + 0];
p6[1] = (int16_t) iPtr[6*width + iIndex + 1];
p6[2] = (int16_t) iPtr[6*width + iIndex + 2];
p6[3] = (int16_t) iPtr[6*width + iIndex + 3];
p6[4] = (int16_t) iPtr[6*width + iIndex + 4];
p6[5] = (int16_t) iPtr[6*width + iIndex + 5];
p6[6] = (int16_t) iPtr[6*width + iIndex + 6];
p6[7] = (int16_t) iPtr[6*width + iIndex + 7];
proj6 = vld1q_s16(p6);

res5 = vmulq_s16( proj5, mog->neon_lookup[csk*8+5] );
vst1q_s16( r5, res5 );


p7[0] = (int16_t) iPtr[7*width + iIndex + 0];
p7[1] = (int16_t) iPtr[7*width + iIndex + 1];
p7[2] = (int16_t) iPtr[7*width + iIndex + 2];
p7[3] = (int16_t) iPtr[7*width + iIndex + 3];
p7[4] = (int16_t) iPtr[7*width + iIndex + 4];
p7[5] = (int16_t) iPtr[7*width + iIndex + 5];
p7[6] = (int16_t) iPtr[7*width + iIndex + 6];
p7[7] = (int16_t) iPtr[7*width + iIndex + 7];
proj7 = vld1q_s16(p7);

res6 = vmulq_s16( proj6, mog->neon_lookup[csk*8+6] );
vst1q_s16( r6, res6 );
res7 = vmulq_s16( proj7, mog->neon_lookup[csk*8+7] );
vst1q_s16( r7, res7 );

cs_temp += (r0[0] + r0[1] + r0[2] + r0[3] + r0[4] + r0[5] + r0[6] + r0[7]);
```

```
        cs_temp += (r1[0] + r1[1] + r1[2] + r1[3] + r1[4] + r1[5] + r1[6] + r1[7]);
        cs_temp += (r2[0] + r2[1] + r2[2] + r2[3] + r2[4] + r2[5] + r2[6] + r2[7]);
        cs_temp += (r3[0] + r3[1] + r3[2] + r3[3] + r3[4] + r3[5] + r3[6] + r3[7]);
        cs_temp += (r4[0] + r4[1] + r4[2] + r4[3] + r4[4] + r4[5] + r4[6] + r4[7]);
        cs_temp += (r5[0] + r5[1] + r5[2] + r5[3] + r5[4] + r5[5] + r5[6] + r5[7]);
        cs_temp += (r6[0] + r6[1] + r6[2] + r6[3] + r6[4] + r6[5] + r6[6] + r6[7]);
        cs_temp += (r7[0] + r7[1] + r7[2] + r7[3] + r7[4] + r7[5] + r7[6] + r7[7]);

        *cs_result_ptr = cs_temp;
        cs_result_ptr++;
      }
    }
  }
#else
  unsigned int csy, csx, csm, csk, csj;
  int16_t cs_temp, cs_temp2;
  for (csy=0;csy<height;csy+=8)
  {
    for(csx=0; csx<width; csx+=8)
    {
      csIndex = 0;
      for(csj=0;csj<8;csj++)// each of the 8 projections
      {
        cs_temp= 0;
        for(csk=0;csk<8;csk++)// block vertical
        {
          for(csm=0;csm<8;csm++)//block horizontal
          {
            cs_temp2 = (int16_t)iPtr[iIndex];
            cs_temp+=(int16_t)(cs_temp2 * CSMOG::csc[csIndex]);
            iIndex++, csIndex++;
          }
          iIndex+=(width-8);
        }
        *cs_result_ptr = cs_temp;
        cs_result_ptr ++;
        iIndex+=(-width*8);
      }
      iIndex +=8;
    }
    iIndex+=(width*7);
  }
#endif

  for (csx=0; csx<width*height/64*8; csx++)
  {
    aPtr[csx] = (int16_t) cs_result[csx];
  }

  delete cs_result;
}

////////////////////////////////////////////////////////////////////////////////
// postProc() - This converts the output from the CS projection and GMM to a
// usable calculated foreground mask.
////////////////////////////////////////////////////////////////////////////////

void CSMOG::postProc(uint8_t * csmogr, uint8_t * cIm, uint8_t* bkgnd, uint32_t width,
                     uint32_t height, uint8_t th, uint32_t alpha,
                     uint8_t * aPtr, uint32_t aIStep, uint32_t aJStep)
{
  //note alpha is the logical shift, so if alpha = 3, alpha is effective 1/8
```

```
  int csk = 0, csm = 0;
  unsigned int csx= 0, csy =0;
  uint8_t * csmogrPtr = csmogr;
  uint8_t * cImPtr = cIm;
  uint8_t * bkgndPtr = bkgnd;
  uint8_t dif;
  uint32_t coef = (1<<alpha)-1;
  uint32_t height_8 = height/8, width_8 = width/8;
  uint8_t *aXPtr = aPtr, *aYPtr = aPtr;

  for (csy=0;csy<height_8;csy++)
  {
    for (csm=0; csm<8; csm++)
    {
      aXPtr = aYPtr;
      for(csx=0; csx<width_8; csx++)
      {

        for(csk=0;csk<8;csk++)// block vertical
        {
          if (csmogrPtr[csx]>0)// if the pixel is in a block that is flagged
          {//compare it to the background
            dif = abs((int32_t)(*bkgndPtr) - (int32_t)*(cImPtr));
            if (dif>=th)
            {
              //yes this pixel has changed
              *aXPtr =MOG_OUTPUT;
            }
            else
            {
              //update the background of this pixel
              *aXPtr = 0;
              *bkgndPtr = (uint8_t)((((uint32_t)(*bkgndPtr)*coef)+((uint32_t)(*cImPtr)))>>alpha);
            }
          }
          else
          {
            *aXPtr = 0;
            *bkgndPtr = (uint8_t)((((uint32_t)(*bkgndPtr)*coef)+((uint32_t)(*cImPtr)))>>alpha);
          }
          cImPtr ++;
          bkgndPtr ++;
          aXPtr += aIStep;
        }
      }
      aYPtr += aJStep;
    }
    csmogrPtr += width_8;
  }
}

////////////////////////////////////////////////////////////////////////////////
// convertToIntensity() - This vectorises the incoming frame.
////////////////////////////////////////////////////////////////////////////////

void CSMOG::convertToIntensity(uint8_t * iPtr, uint32_t iIStep, uint32_t iJStep,
                               uint32_t width, uint32_t height, uint8_t * oPtr)
{
  int length = width * height;
  if ( iIStep == 1 && iJStep == width )
  {
    memcpy( oPtr, iPtr, length * sizeof (uint8_t) );
    return;
```

```
  }
  int i = 0;
  uint8_t * _iPtr = iPtr;
  for( i=0; i < length; i++)
  {
    oPtr[i] = *_iPtr;
    _iPtr += iIStep;
  }
}

#ifdef __ARM_NEON__
////////////////////////////////////////////////////////////////////////////////
// setup_neon_lookup() - Sets up the lookup chunks for use with NEON intrinsics.
////////////////////////////////////////////////////////////////////////////////

void CSMOG::setup_neon_lookup(CSGMM *mog)
{
  int16_t c0[8], c1[8], c2[8], c3[8], c4[8], c5[8], c6[8], c7[8];
  int i, p = 0;
  for (i = 0; i < 8; i++)
  {
    c0[0] = CSMOG::csc[p];
    c0[1] = CSMOG::csc[p+1];
    c0[2] = CSMOG::csc[p+2];
    c0[3] = CSMOG::csc[p+3];
    c0[4] = CSMOG::csc[p+4];
    c0[5] = CSMOG::csc[p+5];
    c0[6] = CSMOG::csc[p+6];
    c0[7] = CSMOG::csc[p+7];
    mog->neon_lookup[8*i] = vld1q_s16(c0);

    c1[0] = CSMOG::csc[p+8];
    c1[1] = CSMOG::csc[p+9];
    c1[2] = CSMOG::csc[p+10];
    c1[3] = CSMOG::csc[p+11];
    c1[4] = CSMOG::csc[p+12];
    c1[5] = CSMOG::csc[p+13];
    c1[6] = CSMOG::csc[p+14];
    c1[7] = CSMOG::csc[p+15];
    mog->neon_lookup[8*i+1] = vld1q_s16(c1);

    c2[0] = CSMOG::csc[p+16];
    c2[1] = CSMOG::csc[p+17];
    c2[2] = CSMOG::csc[p+18];
    c2[3] = CSMOG::csc[p+19];
    c2[4] = CSMOG::csc[p+20];
    c2[5] = CSMOG::csc[p+21];
    c2[6] = CSMOG::csc[p+22];
    c2[7] = CSMOG::csc[p+23];
    mog->neon_lookup[8*i+2] = vld1q_s16(c2);

    c3[0] = CSMOG::csc[p+24];
    c3[1] = CSMOG::csc[p+25];
    c3[2] = CSMOG::csc[p+26];
    c3[3] = CSMOG::csc[p+27];
    c3[4] = CSMOG::csc[p+28];
    c3[5] = CSMOG::csc[p+29];
    c3[6] = CSMOG::csc[p+30];
    c3[7] = CSMOG::csc[p+31];
    mog->neon_lookup[8*i+3] = vld1q_s16(c3);

    c4[0] = CSMOG::csc[p+32];
    c4[1] = CSMOG::csc[p+33];
```

```
    c4[2] = CSMOG::csc[p+34];
    c4[3] = CSMOG::csc[p+35];
    c4[4] = CSMOG::csc[p+36];
    c4[5] = CSMOG::csc[p+37];
    c4[6] = CSMOG::csc[p+38];
    c4[7] = CSMOG::csc[p+39];
    mog->neon_lookup[8*i+4] = vld1q_s16(c4);

    c5[0] = CSMOG::csc[p+40];
    c5[1] = CSMOG::csc[p+41];
    c5[2] = CSMOG::csc[p+42];
    c5[3] = CSMOG::csc[p+43];
    c5[4] = CSMOG::csc[p+44];
    c5[5] = CSMOG::csc[p+45];
    c5[6] = CSMOG::csc[p+46];
    c5[7] = CSMOG::csc[p+47];
    mog->neon_lookup[8*i+5] = vld1q_s16(c5);

    c6[0] = CSMOG::csc[p+48];
    c6[1] = CSMOG::csc[p+49];
    c6[2] = CSMOG::csc[p+50];
    c6[3] = CSMOG::csc[p+51];
    c6[4] = CSMOG::csc[p+52];
    c6[5] = CSMOG::csc[p+53];
    c6[6] = CSMOG::csc[p+54];
    c6[7] = CSMOG::csc[p+55];
    mog->neon_lookup[8*i+6] = vld1q_s16(c6);

    c7[0] = CSMOG::csc[p+56];
    c7[1] = CSMOG::csc[p+57];
    c7[2] = CSMOG::csc[p+58];
    c7[3] = CSMOG::csc[p+59];
    c7[4] = CSMOG::csc[p+60];
    c7[5] = CSMOG::csc[p+61];
    c7[6] = CSMOG::csc[p+62];
    c7[7] = CSMOG::csc[p+63];
    mog->neon_lookup[8*i+7]= vld1q_s16(c7);
  }
}
#endif
```

Appendix D

Source Code Listing - Non-compressive Mixture of Gaussians

This appendix contains the non-compressive Mixture of Gaussians class and modified frame processing function. This can be used as a substitute with the corresponding files in Appendix C.

D.1 type_wrappers.hpp

```
/*****************************************************************************
 * type_wrappers.hpp - Wrapper classes for OpenCV IplImage for easier access
 * to image data. Based on code found here:
 * http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html
 *
 * Ashton Fagg (ash.fagg@csiro.au) - May 2012
 *****************************************************************************/

#ifndef _TYPE_WRAPPERS_H
#define _TYPE_WRAPPERS_H

#include <cv.h>
#include <cxcore.h>
/*****************************************************************************
 * pixel class definition
 * This class models each pixel in terms of red, green and blue.
 *****************************************************************************/

class Pixel
```

```
{
  public:
    Pixel(){ ; } // Don't need this one
    Pixel( unsigned char, unsigned char, unsigned char );
    Pixel& operator=(const Pixel&);
    inline unsigned char& operator()( const int _ch )
    {
      return ch[ _ch ];
    }
    inline unsigned char operator() ( const int _ch ) const
    {
      return ch[ _ch ];
    }
    unsigned char ch[ 3 ];
};

/*****************************************************************************
 * RGB_Img class definition
 *
 * This class holds an RGB image. This is wrapped around the IplImage defintion
 * in order to provide some nice, easy access to the image data and perform all
 * of the necessary pointer arithmetic automatically.
 *****************************************************************************/

class RGB_Img
{
  public:
    RGB_Img( IplImage* );
    RGB_Img( void );
    ~RGB_Img( void ){ if ( payload != NULL ) cvReleaseImage( &payload ); }

    // Lets us yank the raw IplImage out if we need to
    IplImage* extract( void );
    const IplImage* extract( void ) const;

    void reset( void ); // Reset to all zeros if needed
    void operator=( IplImage* ); //Treat these objects the same as IplImage

    const unsigned char& operator()(  const int row,
        const int col,
        const int chan ) const
    {
      int i = row * payload->widthStep + col * payload->nChannels + chan;
      return (unsigned char &) payload->imageData[ i ];
    }

    unsigned char& operator()( const int row,
        const int col,
        const int chan )
    {
      int i = row * payload->widthStep + col * payload->nChannels + chan;
      return (unsigned char &) payload->imageData[ i ];
    }

    Pixel& operator()( const int row, const int col )
    {
      int i = row * payload->widthStep + col * payload->nChannels;
      return (Pixel &) payload->imageData[ i ];
    }

    const Pixel& operator()( const int row, const int col ) const
    {
      int i = row * payload->widthStep + col * payload->nChannels;
```

```
      return (Pixel &) payload->imageData[ i ];
    }

  private:
    IplImage *payload;
};

/*****************************************************************************
 * Mask class definition
 * This holds a mask showing the pixels which are members of the foreground
 * or background. This will be a binary image so there's no need to have
 * channel specific operations.
 *****************************************************************************/

class Mask
{
  public:
    Mask( IplImage * );
    Mask() { ; }
    ~Mask( void ) { ; }
    void dealloc( bool );
    void reset( void );
    IplImage* extract( void );
    const IplImage* extract( void ) const;
    void operator=( IplImage* );
    inline unsigned char& operator()( const int row, const int col )
    {
      int index = row * payload->widthStep + col;
      return (unsigned char &) payload->imageData[ index ];
    }
    inline const unsigned char& operator()( const int row,
        const int col ) const
    {
      int index = row * payload->widthStep + col;
      return (unsigned char &) payload->imageData[ index ];
    }
  protected:
    IplImage *payload;
    bool release_mem;
};

#endif
```

D.2 type_wrappers.cpp

```
/*****************************************************************************
 * type_wrappers.cpp - Wrapper class implementations as per type_wrappers.hpp
 *
 * Ashton Fagg (ash.fagg@csiro.au) - May 2012
 *****************************************************************************/

#include "type_wrappers.hpp"
#include <cv.h>
#include <cxcore.h>

/*****************************************************************************
 *                    Pixel class implementation
 *****************************************************************************/

// This should be obvious, but this just sets up a pixel with the values
```

```
// specified as R, G and B.

Pixel::Pixel( unsigned char r, unsigned char g, unsigned char b )
{
  ch[ 0 ] = r;
  ch[ 1 ] = g;
  ch[ 2 ] = b;
}

// Make this equal to another pixel
Pixel& Pixel::operator=( const Pixel& a )
{
  ch[ 0 ] = a.ch[ 0 ];
  ch[ 1 ] = a.ch[ 1 ];
  ch[ 2 ] = a.ch[ 2 ];
  return *this;
}


/*****************************************************************************
 *                    RGB_Img class implementation
 *****************************************************************************/

RGB_Img::RGB_Img( )
{
  payload = NULL;
}
RGB_Img::RGB_Img( IplImage *a )
{
  payload = a;
}


void RGB_Img::operator=( IplImage *a )
{
  //if ( payload != NULL ) cvReleaseImage( &payload );
  payload = a;
}

void RGB_Img::reset( void )
{
  cvZero( payload );
}

IplImage* RGB_Img::extract( void )
{
  return payload;
}

const IplImage* RGB_Img::extract( void ) const
{
  return payload;
}

/*****************************************************************************
 *                          Mask class implementation
 *****************************************************************************/

Mask::Mask( IplImage *a = NULL )
{
  payload = a;
  release_mem = true;
}
```

```
void Mask::dealloc( bool a )
{
  release_mem = a;
}

void Mask::reset( void )
{
  cvZero( payload );
}

void Mask::operator=( IplImage *a )
{
  payload = a;
}

IplImage* Mask::extract( void )
{
  return payload;
}

const IplImage* Mask::extract( void ) const
{
  return payload;
}
```

D.3 gmm.hpp

```
/*****************************************************************************
 * gmm.hpp - Gaussian Mixture Model class definitions
 *
 * See gmm.cpp for implementation details.
 *
 * Ashton Fagg (ashton@fagg.id.au) - May 2012
 *****************************************************************************/

#ifndef GMM_H
#define GMM_H

#include "type_wrappers.hpp"

#define RED 0
#define GREEN 1
#define BLUE 2

/*****************************************************************************/
// Struct to hold the user specified GMM parameters.
typedef struct GMM_User_Params
{
  float low_thresh;
  float high_thresh;
  float bg_thresh;
  float alpha;
  float cp;
  float var;
  int max_components;
} GMM_User_Params;
/*****************************************************************************/
// Gaussian components for the GMM. There will be M of these per pixel.
class GaussianComponent
```

```
{
  public:
    GaussianComponent()
    {
      R_mu = 0; G_mu = 0; B_mu = 0;
      w = 0;
      sigma = 0;
    }
    void set_mu( float _r, float _g, float _b )
    {
      R_mu = _r; G_mu = _g; B_mu = _b;
    }
    void set_w( float _w ) { w = _w; }
    void set_sig( float s ) { sigma = s; }
    float r_mu( void ) { return R_mu; }
    float g_mu( void ) { return G_mu; }
    float b_mu( void ) { return B_mu; }
    float weight( void ) { return w; }
    float sig( void ) { return sigma; }

  private:
    float R_mu, G_mu, B_mu, sigma, w;
};
/*****************************************************************************/
// This is the GMM itself.
class GaussianMixtures
{
  public:
    GaussianMixtures()
    {
      fframe = true;
      framen = 0;
    }
    GaussianMixtures( const GMM_User_Params& usr_par )
    {
      fframe = true;
      framen = 0;
      low_thresh = usr_par.low_thresh;
      high_thresh = usr_par.high_thresh;
      alpha = usr_par.alpha;
      max_components = usr_par.max_components;
      bg_thresh = usr_par.bg_thresh;
      cp = usr_par.cp;
      var = usr_par.var;
    }
    // Processes incoming frames
    void process( cv::Mat const& , cv::Mat& );
  private:
    void setup( const cv::Mat );
    void subtract( const RGB_Img&, Mask&, Mask& );
    int pix_update( long, const Pixel&, int,
        unsigned char&, unsigned char& );
    int calc_comp_pos( int, int );
    bool fframe;
    long framen;
    float low_thresh, high_thresh, bg_thresh, alpha, cp, var;
    int max_components;
    int f_width, f_height, f_sz;
    std::vector<GaussianComponent*> comp;
    std::vector<int> comp_per_pixel;
    IplImage *frame;
    RGB_Img _frame;
    RGB_Img mod_bg;
```

```
    Mask l_mask;
    Mask h_mask;
    static const int BG = 0;
    static const int FG = 255;
};

#endif
```

D.4 gmm.cpp

```
/*****************************************************************************
 * gmm.cpp -> Gaussian Mixtures Model implementation
 *
 * Model update code is based upon code by Donovan Parks:
 * http://dparks.wikidot.com/source-code
 *
 * The functions in this file are ordered in terms of abstraction, with higher
 * level interfaces towards the top. These functions include the user-facing
 * processing routine, set up routines and Mixture Model update and background
 * subtraction routines.
 *
 * Ashton Fagg (ash.fagg@csiro.au), May 2012
 *****************************************************************************/
#include "gmm.hpp"
#include "cv.h"
#include "type_wrappers.hpp"

using std::cout;
using std::endl;

/* This processes the incoming frames and sets up a few more things if
 * the frame is the first one.
 */

void GaussianMixtures::process( cv::Mat const &in, cv::Mat &out )
{
  if( in.empty() ) return;

  // Perform the type conversions
  frame = new IplImage( in );
  _frame = frame;
  // If this is the first frame, set up some stuff.
  if ( fframe )
  {
    GaussianMixtures::setup( in );
    l_mask = cvCreateImage( cvSize( f_width, f_height ), IPL_DEPTH_8U, 1 );
    l_mask.extract()->origin = IPL_ORIGIN_BL;
    h_mask = cvCreateImage( cvSize( f_width, f_height ), IPL_DEPTH_8U, 1 );
    h_mask.extract()->origin = IPL_ORIGIN_BL;
    mod_bg = cvCreateImage( cvSize( f_width, f_height ), IPL_DEPTH_8U, 3 );
    mod_bg.extract()->origin = IPL_ORIGIN_BL;
  }
  GaussianMixtures::subtract( _frame, l_mask, h_mask );
  cv::Mat fg( h_mask.extract() );
  fg.copyTo( out );
  fframe = false;
  framen++;
  fg.release();
  delete frame;
}
```

```
/* This sets up the mixture model according to the frame size parameters.
 * Allocates components for each pixel and sets the used components to zero.
 */

void GaussianMixtures::setup( const cv::Mat fr )
{
  int i;

  f_width = fr.size().width;
  f_height = fr.size().height;
  f_sz = f_width * f_height;
  for ( i = 0; i < f_sz; ++i )
  {
    comp_per_pixel.push_back( 0 ); // Set them all to zero
  }

  for ( i = 0; i < f_sz * max_components; ++i )
  {
    comp.push_back( new GaussianComponent );// Components are set to zero
    // by the constructor.
  }
}

int GaussianMixtures::calc_comp_pos( int r, int c )
{
  int ret = max_components * (r * f_width + c );
  return ret;
}


/* This performs the actual pixel-by-pixel subtraction. Eventually this
 * will become SMP-aware, so it'll set to work on multiple pixels at once.
 */
void GaussianMixtures::subtract( const RGB_Img& d,
    Mask& l_mask, Mask& h_mask )
{
  unsigned char l_tmask;
  unsigned char h_tmask;
  long pos;
  int comps_used;
  int r, c;

  // Update according to raster order
  for ( r = 0; r < f_height; r++ )
  {
    for ( c = 0; c < f_width; c++ )
    {
      pos = GaussianMixtures::calc_comp_pos( r, c );

      l_mask( r, c ) = l_tmask;
      h_mask( r, c ) = h_tmask;
      // Update the pixel.
      comps_used = comp_per_pixel.at( r * f_width + c );
      comp_per_pixel.at( r*f_width + c  ) = GaussianMixtures::pix_update( pos,
          d( r, c ), comps_used, l_tmask,
          h_tmask );
      // Set the background up.
      mod_bg( r, c, RED ) = (unsigned char) comp.at( pos )->r_mu();
      mod_bg( r, c, BLUE ) = (unsigned char) comp.at( pos )->b_mu();
      mod_bg( r, c, GREEN ) = (unsigned char) comp.at( pos )->g_mu();
      l_mask( r, c ) = l_tmask;
      h_mask( r, c ) = h_tmask;
```

```
    }
  }
}

/* This mess updates the model for one pixel and it's components. */
int GaussianMixtures::pix_update( long pos, const Pixel& pix,
    int comps_used,
    unsigned char& lt, unsigned char& ht )
{
  int i, locali;
  long p;
  bool fits_pdf = false;
  bool bg_low = false;
  bool bg_high = false;
  float min_alpha = 1 - alpha;
  float kill = (-1*alpha)*cp;
  float total_w = 0.0f;
  int bg_comps = 0;
  double sum = 0.0;

  // Work out how many background Gaussian components
  for ( i = 0; i < comps_used; ++i )
  {
    if ( sum < bg_thresh )
    {
      bg_comps++;
      sum += comp.at( i + pos )->weight();
    }
    else break;
  }

  // Update all of the components and check if there is an existing component
  for ( i = 0; i < comps_used; i++ )
  {
    p = pos + i;
    float w = comp.at( p )->weight();
    if( !fits_pdf )
    {
      float v = comp.at( p )->sig();
      float mur = comp.at( p )->r_mu();
      float mug = comp.at( p )->g_mu();
      float mub = comp.at( p )->b_mu();
      // Calculate the drift
      float dr = mur - pix( RED );
      float dg = mug - pix( GREEN );
      float db = mub - pix( BLUE );
      float sq_dist = (dr*dr) + (dg*dg) + (db*db);

      if ( sq_dist < high_thresh * var && i < bg_comps )
        bg_high = true;
      if ( sq_dist < low_thresh * var )
      {
        fits_pdf = true;
        // Check if this is part of background
        if ( i < bg_comps ) bg_low = true;
        // Update the components
        float k = alpha / w;
        w = min_alpha * w + kill;
        w += alpha;
        comp.at( p )->set_mu( mur - (k*dr),
            mug - (k*dg),
            mub - (k*db) );
        float new_sigma =  v + k * ( sq_dist - v );
```

```
        // Set the new variance according to the limit
        comp.at( p )->set_sig( new_sigma < 4 ? 4 : new_sigma > 5 * var
            ? 5 * var : new_sigma );
        // Sort the component into descending
        // order of weight.
        for ( int iloc = i; iloc > 0; iloc-- )
        {
          long pl = pos + iloc;
          if ( comp.at( pl )->sig() > comp.at( pl - 1 )->sig() )
          {
            GaussianComponent *tmp = comp.at( pl );
            comp.at( pl ) = comp.at( pl - 1 );
            comp.at( pl - 1 ) = tmp;
          }
          else
          {
            break;
          }
        }
      }
    }
    else
    {

      w = min_alpha * w + kill;
      // Check if we need to drop this component
      if ( w < -1*kill )
      {
        w = 0.0;
        comps_used--;
      }
      comp.at( p )->set_sig( w );
    }
    total_w += w;
  }

  for ( int locali = 0; locali < comps_used; locali++ )
  {
    int pl = pos + locali;
    comp.at( pl )->set_w( comp.at( pl )->weight() / total_w );
  }


  // Make a new component if required
  if ( !fits_pdf )
  {
    if( comps_used == max_components )
    {
    }
    else
    {
      comps_used++;
      long pl = pos + comps_used - 1;

      if ( comps_used  == 1 )
      {
        comp.at( pl )->set_w( 1 );
      }
      else
      {
        comp.at( pl )->set_w( alpha );
      }
      // Renormalise the weights so they sum to 1
```

```
      float sum = 0.0;
      for ( locali = 0; locali < comps_used; locali++ )
      {
        sum += comp.at( pos + locali )->weight();
      }
      // Check to make sure we actually need to do the renormalisation
      if ( sum != 0.0 )
      {
        for ( locali = 0; locali < comps_used; locali++ )
        {
          float curr_weight = comp.at( pos + locali )->weight();
          comp.at( pos + locali )->set_w( (1.0f/sum) * curr_weight );
        }
      }
      comp.at( pl )->set_mu( pix( RED ), pix( GREEN ), pix( BLUE ) );
      comp.at( pl )->set_sig( var );
    }
    // Sort the components in descending order by weight
    for ( locali = comps_used - 1; locali > 0; locali-- )
    {
      long x = pos + locali;
      if ( comp.at( x )->weight() > comp.at( x - 1 )->weight() )
      {
        GaussianComponent *tmp = comp.at( x );
        comp.at( x ) = comp.at( x - 1 );
        comp.at( x - 1 ) = tmp;
      }
      else
      {
        break;
      }
    }

  }

  if ( bg_low )
  {
    lt = (unsigned char) BG;
  }
  else
  {
    lt = (unsigned char) FG;
  }
  if ( bg_high )
  {
    ht = (unsigned char) BG;
  }
  else
  {
    ht =  (unsigned char) FG;
  }

  return comps_used;
}
```

Appendix E

Source Code Listing - Support Scripts

This Appendix contains the support scripts used to supervise the vision software and synchronise output data between the local deployment and a remote server and self-supervision of the vision system. Please note, that due to security reasons some parts have been redacted or generalised.

E.1 conf.py

```
# System Vars
hostName = 'name'
location = 'location'
conf_ver = '2.0'
logfile = '/var/log/supervisor.py'

# VPN Config
vpn_enable = False # Enable the VPN tunnel
vpn_connect_cmd = ['/usr/sbin/vpnc-connect', hostName]
vpn_disconnect_cmd = ['/usr/sbin/vpnc-disconnect']
vpn_ping_internal = ['ping', '-c5','-t5','REDACTED'] # ping 5 times with a ttl of 5 secs
vpn_timeout = 30 # 30 second timeout

# File return configuration
rsync_host = 'somehost.somewhere.com'
rsync_user = 'rsyncuser'
rsync_localdir = '/mnt/usb/'
rsync_remotedir = '/path/to/collection/'
rsync_cmd = ['/usr/bin/rsync -azh %s %s@%s:%s', %(rsync_localdir, rsync_user, rsync_host, rsync_remotedir,)]

## Config Rules
```

```
# The lines begin with a keyword (no leading spaces!).
# The values start exactly one space after the keywords, and run to the end of line.
# This lets you put any kind of weird character (except CR, LF and NUL) in your strings,
# but it does mean you can't add comments after a string, or spaces before them.
vpnConnect = [
'IPSec gateway REDACTED',
'IPSec ID REDACTED',
'IPSec secret REDACTED',
'Xauth username REDACTED',
'Xauth password REDACTED',
'NAT-Keepalive packet interval 60' # Ping the VPN Tunnel every 1min
]

# Note: supervisor will populate the ip and mac address
ip = ''
mac = ''

timeServer = '0.au.pool.ntp.org'
inetIface = 'eth0'
tunnIface = 'tun0'

# Local app list
appDir = '/usr/local/bin'

## How to use this section
#
# The program must be a list, with each parameter a seperate item.
# Note: Pipe (|) and stream redirection (>) will not work
#
# If the first item in the list matches one of the items in 'launchInScreen'
# then the app will be launched in a detatched screen.
#
# appList = [
#     ['foslisten', '--serial=%s' % fleckPort, '--errlogfile=/var/log/foslisten.log', '--safety', '--show=rtd
#     ['fosdbpush', '--location=%s' % (location,)],
#     ['ssh', '-NfR', '2025:127.0.0.1:22', 'fleck@www.sensornets.csiro.au'],
#     ['ssh', '-NfR', '9341:127.0.0.1:9001', 'fleck@www.sensornets.csiro.au'],
#           ]

# REDACTED



## Variables for gateway reset
modem_reset_enable = False
modem_reset_cmd = ['ssh', 'REDACTED', "'/sbin/reboot'"]
```

E.2 supervisor.py

```
#!/usr/bin/env python

import FosVirtual
import os
import signal
import subprocess
import re
import string
import sys
import time
# For IP Address discovery
```

```
import socket
import fcntl
import struct
# For option parsing
import optparse
# For logging
import LogMsgHandler

## What this script needs to do:
##  - get config information from the database/config file
##  - make vpn connection
##  - make sure all the apps are running
##  - check the status of dbpush and listener
##  - check for updates
##  - push log information into the database
##    - uptime, ip address

################################################################################
## Helper Functions

# credit: http://code.activestate.com/recipes/439094/
def getIpAddr(ifname):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        return socket.inet_ntoa(fcntl.ioctl(
                s.fileno(),
                0x8915,  # SIOCGIFADDR
                struct.pack('256s', ifname[:15])
                )[20:24])
    except IOError:
        return None

def getHwAddr(ifname):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        info = fcntl.ioctl(s.fileno(), 0x8927,  struct.pack('256s', ifname[:15]))
        return ''.join(['%02x:' % ord(char) for char in info[18:24]])[:-1]
    except IOError:
        return None

def readConfigFile(configFile):
  """
        readConfigFile reads the config file set in variable configFile or given as argv[1].
        The imported configuration parameters are accessed through the params variable
  """
  global config
  if configFile[-3:] == ".py":
    configFile = configFile[:-3]
  config = __import__(configFile, globals(), locals(), [])
  print "Parameters loaded from config file."
  return True

def getAppPID(appName):
  pid = {};
  # The list of running apps
  ret = os.popen("ps xa", "r");
  runningAppList = ret.readlines();
  for r in runningAppList:
    if r.find( appName ) != -1:
      pid[r.split()[0]] = r[27:-1]

  return pid;
```

```
def uptime2Time(uptime):
    try:
        uptimeStr = re.search("up[\s\w,]*(\d+):(\d+)", uptime).group().split()
        if len(uptimeStr) == 4:
            return 60*(int(uptimeStr[1])*24 + int(uptimeStr[3].split(':')[0])) + int(uptimeStr[3].split(':')[1
        elif len(uptimeStr) == 2:
            return int(uptimeStr[1].split(':')[0])*60 + int(uptimeStr[1].split(':')[1])
        else:
            return None
    except AttributeError:
        return None

def proccessWait(proccess, waitTime=10):
    startTime = time.time()

    # If waitTime is zero or below return only when finished
    if waitTime <= 0:
        os.waitpid(proccess.pid, 0)

    # If proccess running with timeout
    while proccess.poll() is None and time.time() - startTime < waitTime:
        time.sleep(5)

    if proccess.returncode is None:
        os.kill(proccess.pid,signal.SIGKILL)

def vpnConfigure():
    global config
    vpnconf = None
    if os.path.isfile("/etc/vpnc/%s.conf"%config.hostName):
        f = open("/etc/vpnc/%s.conf"%config.hostName, 'r')
        vpnconf = f.read()
        f.close()
    vpncmd = ""
    for cmd in config.vpnConnect:
        vpncmd += "%s\n"%cmd
    if vpnconf != vpncmd:
        try:
            #os.system( "/usr/local/sbin/remountrw" );
            f = open("/etc/vpnc/%s.conf"%config.hostName, 'w')
            f.write( vpncmd );
            f.close();
            os.system( "/usr/local/sbin/remountro" );
            log.write('VPN config updated /etc/vpnc/%s.conf'%config.hostName, 3)
        except IOError:
            log.write('Could not update /etc/vpnc/%s.conf'%config.hostName, 3)
            return False
    return True

def vpnConnect():
    global log, vpn
    if getIpAddr(config.tunnIface) == None:
        ret = subprocess.Popen(config.vpn_connect_cmd, shell=False)
        proccessWait(ret, config.vpn_timeout)

        if ret.returncode == 0:
            log.send('VPN connect command executed', 3)
            vpn['ip'] = getIpAddr(config.tunnIface)
            vpn['upd'] = True
            return True
        else:
            log.send('VPN connect command failed', 3)
            modemReset()
```

```
            return False
    else:
        log.send('VPN_disconnect_command_execute', 3)
        ret = subprocess.call(config.vpn_disconnect_cmd)

        if ret == 0 or ret ==1:
            time.sleep(1)
            ret = subprocess.Popen(config.vpn_connect_cmd, shell=False)
            proccessWait(ret, config.vpn_timeout)

            if ret.returncode == 0:
                log.send('VPN_connect_command_executed', 3)
                vpn['ip'] = getIpAddr(config.tunnIface)
                return True
            else:
                log.send('VPN_connect_command_failed', 3)
                modemReset()
                return False
        else:
            log.send('VPN_disconnect_command_failed', 3)
    return False

def vpnCheck():
    global vpn, log
    vpn['ip'] = getIpAddr(config.tunnIface)
    vpn['upd'] = False
    if vpn['ip'] == None:
        log.write('VPN_not_connected', 3)
        if vpnConfigure():
            return vpnConnect()
        else:
            return False
    else:
        log.write('VPN_connected_-_Testing_Connection', 3)
        ret = subprocess.call(config.vpn_ping_internal)

        if ret == 0:
            log.send('VPN_responding', 3)
            return True
        else:
            log.send('VPN_failed', 3)
            if vpnConfigure():
                return vpnConnect()
            else:
                return False
    return False

def modemReset():
    global db, log, config
    if config.modem_reset_enable:
        if db:
            db.disconnect()
            db = None
        config.modem_reset_enable = False
        ret = subprocess.call(config.modem_reset_cmd)
        if ret == 0:
            log.send('Modem_reset_command_executed', 1, 'supervisor', 'Connection', info)
        else:
            log.send('Modem_reset_command_failed', 1, 'supervisor', 'Connection', info)

################################################################################
## Main Program
```

```
print "Starting supervisor.py"

# Should have an option handler here - to specify config file, test mode, offline mode etc

p = optparse.OptionParser()
p.add_option('--debug', dest='debug', type='int', help='Debug level (0-3)')
p.add_option('--simulate', dest='simulate', action='store_true', help='Simulate launching of programs')
p.add_option('--config', dest='configLoc', type='str', help='Config file location')

p.set_defaults(debug=0, simulate=False, nodb=False, configLoc='conf.py')

(opts, args) = p.parse_args()
globals().update(opts.__dict__)

########################################
## Import config file
readConfigFile('conf.py')

########################################
## Start LogMsgHandler

db = None
log = LogMsgHandler.LogMsgHandler(db, config.logfile, sys.stderr)

########################################
##
statusNotes = 'Normal - VPN'
statusNotesDefault = statusNotes

########################################
## Check network interfaces
# Local interface
config.ip = getIpAddr(config.inetIface)
config.mac = getHwAddr(config.inetIface)
log.write('IP Address: %s' % config.ip, 3)
log.write('MAC Address: %s' % config.mac, 3)
info = {'mac_address': config.mac, 'location': config.location}

########################################
## Check if supervisor already running
supervisorRunningList = getAppPID('supervisor')
for pid in supervisorRunningList:
    if int(pid) not in [ os.getpid(), os.getppid() ]:
        os.kill(int(pid),signal.SIGKILL)
        # Log message here that supervisor (#pid) was already running and killed
        log.send('Supervisor already running. Process %s killed' % pid, 1, 'supervisor', 'Program', info)


########################################
## Check VPN connection
# VPC interface
vpn = {}
if config.vpn_enable and vpnCheck():
    vpn_active = True
    log.write('VPN IP Address: %s' % vpn['ip'], 3)
    log.write('VPN Status: %s' % vpn_active, 3)
elif not config.vpn_enable:
    vpn_active = True
    log.write('VPN Status: Disabled', 3)
    vpn['upd'] = False
else:
    vpn_active = False
    modemReset()
```

```
########################################
## Update Time

if vpn_active:
    p = subprocess.Popen(['/usr/sbin/ntpdate', config.timeServer], shell=False)
    ntpTime = time.time()

    # If proccess running with timeout of 10 seconds
    while p.poll() is None and time.time() - ntpTime < 10:
        time.sleep(5)

    if not p.returncode and p.returncode == 0:
        log.send("Time adjusted using ntpdate", 2, 'supervisor', 'Program', info)
    else:
        try:
            p = subprocess.Popen(['/etc/init.d/ntp', 'stop'], shell=False)
            ret = os.waitpid(p.pid, 0)
        except OSError:
            ret = 0
        if ret == 0:
            p = subprocess.Popen(['/usr/sbin/ntpdate', config.timeServer], shell=False)

            # If proccess running with timeout of 10 seconds
            proccessWait(p)
        if not p.returncode and p.returncode == 0:
            log.send("Time adjusted using ntpdate", 2, 'supervisor', 'Program', info)
        else:
            log.send("Problem occured when updating time using ntpdate", 1, 'supervisor', 'Program', info)
            if statusNotes != statusNotesDefault:
                statusNotes += ', time not synced'
            else:
                statusNotes = 'Time not synced'

########################################
## Update config file

# Need to save directly to the file and then reload the config file. Check the version numbers first
#if statusNotes != statusNotesDefault:
#    statusNotes += ', Local config updated'
#else:
#    statusNotes = 'Local config updated'


########################################
## Update hostname on system
f = open('/etc/hostname', 'r')
host = f.read()
f.close()
if host != config.hostName:
    try:
        f = open('/etc/hostname', 'w')
        f.write( config.hostName );
        f.close();
        log.write('Updated /etc/hostname', 3)
    except IOError:
        log.write('Could not update /etc/hostname', 3)

########################################
## Check for system updates

# If update, kill running program and then relaunch
```

```
#if statusNotes != statusNotesDefault:
#    statusNotes += ', Updated System'
#else:
#    statusNotes = 'Updated System'


########################################
## Check list of running programs

for app in config.appList:
    appRunning = False
    appPid = getAppPID(app[0])
    for pid in appPid:
        if appPid[pid].find(''.join(['%s ' % i for i in app])[:-1]) != -1:
            log.write('App %s already running' % app[0], 3)
            appRunning = True
    if not appRunning:
        log.write('App %s not running' % app[0], 3)
        if app[0] in config.launchInScreen:
            cmd = ['screen', '-d', '-m'] + app
        else:
            cmd = app
        if not simulate:
            try:
                subprocess.Popen(cmd)
            except OSError:
                log.write('Could not find %s'%app[0], 3)
        log.send('Starting %s with %s' % (app[0], ''.join(['%s ' % i for i in cmd])[:-1]), 1, 'supervisor', 'P

########################################
## Update log files

# Get uptime information
ret = os.popen('uptime', 'r')
uptime = ret.readlines()[0][:-1]
uptime = uptime[0:uptime.rfind(',  load')]
ret.close()
log.write("Uptime: %s" % uptime, 2)

########################################
## Check crontab entry
f = open( "/etc/crontab", "r" );
cronTab = ''.join( f.readlines() );
f.close();
if cronTab.find( "supervisor.py" ) == -1:
    try:
        f = open( "/etc/crontab", "a" );
        f.write( "PYTHONPATH=.:%s/pyclasses\n" % (config.appDir,) );
        f.write( "# Run the supervisor every 5 min\n" );
        f.write( "*/5 * * * * root /usr/bin/python %s/supervisor.py 2> /tmp/supervisor.log\n" % (config.appDir
        f.write( "# Start the supervisor and set user variables on startup\n" );
        # Export the python path back into bashrc as /root is a ram disk and all changes are lost on reboot
        f.write( "@reboot root echo 'export PYTHONPATH=%s/pyclasses:.' >> /root/.bashrc\n" % (config.appDir,)
        f.write( "@reboot root /usr/bin/python %s/supervisor.py 2> /tmp/supervisor.log\n" % (config.appDir,) )
        # Set up the rsync file push to run every 30 minutes
        f.write( "*/30 * * * * root %s" %(rsync_cmd,) );
        f.close();
    except IOError:
        log.send('Unable to open crontab', 2, 'supervisor', 'Program', info)
    os.system( "/usr/local/sbin/remountro" );
    log.send('Adding supervisor to crontab', 2, 'supervisor', 'Program', info)
else:
    log.write("Supervisor in crontab", 3)
```

```
########################################
## Trim supervisor.log file
os.system("sed -i -e ':a' -e '$q;N;101,$D;ba' %s" % config.logfile)
```

Appendix F

Errata

This Appendix contains external resources which may aid in the understanding & verification of results presented in this dissertation.

F.1 Preliminary Detection Trial - Results

The results of this test have been uploaded to the Internet and can be viewed via YouTube. The outcomes of this test show a 100% detection rate. Each stage of processing is visible to aid in understanding of the results.

See: `http://www.youtube.com/watch?v=xdPQEtSCEs4`

F.2 Preliminary Noise Tolerance Trial - Results

The results of this test have been uploaded to the Internet and can be viewed via YouTube. The outcomes of this test show a significant degree of noise tolerance, with only one spurious motion detected. Each stage of processing is visible to aid in understanding of the results.

See: `http://www.youtube.com/watch?v=E1a1bMTlVJk`

F.3 Extended Algorithm Testing - Results

The results of these tests have been uploaded to the Internet and can be viewed via YouTube. Each stage of processing is visible to aid in understanding of the results.

F.3.1 Dataset 1

See: http://www.youtube.com/playlist?list=PLS6E9hSzh0N4arm_xXo21MKD7SDgWGakq&feature=view_all

F.3.2 Dataset 2

See: http://www.youtube.com/playlist?list=PLS6E9hSzh0N71rMHLSfVsOjXSli3yTnBd&feature=view_all

F.3.3 Dataset 3

See: http://www.youtube.com/playlist?list=PLS6E9hSzh0N6WgyX2tHU8lv4gPm2lxERv&feature=view_all