Salient event detection in video surveillance scenarios

Kenneth Ellingsen

Master's thesis
Master of Science in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2008

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Every day huge amounts of surveillance video data is accumulated, and it needs to be handled in a way to extract only the eventful segments. There can be one or multiple persons, whose job it is to sit and watch all the surveillance video screens, seeking to discover any abnormal events taking place. This is not an ideal solution. In this project we investigate the detection of remarkable events in video surveillance scenarios.

We look into how to distinguish events in surveillance scenarios, and further what is an remarkable event. Analyzing surveillance data, without the knowledge of when and where or even if an interesting event has occurred often takes place, and is very time consuming labor. In this kind of analysis the analyst is interested in extraordinary events, something that deviates from the normal. Not having the suitable tools it can be a discouraging task for the analyst, consisting of sequentially viewing all raw video data, and using his judgement to determine if an event is unusual.

Is it possible to create a system for modeling salient events in surveillance scenarios? How does one determine what stands out as a remarkable event? How to distinguish between less remarkable events and more remarkable event taking place? Who decides what is remarkable and what is not?

Keywords: Abnormal events, surveillance, object tracking, feature extraction, feature analysis.

Sammendrag

Hver dag genereres store mengder video fra overvåkningssytemer verden rundt, og all denne videoen må håndteres på en måte slik at kun de mest interessante sekvensene vises. Det er normalt en eller flere personer, som jobb er å sitte å følge med på monitorer, for å kunne fange opp spesielle handelser når det skjer. Denne løsningen er ikke ideell. I denne oppgaven vil vi forske på metoder for å kunne detektere unormale hendelser automatisk.

Vi vil se på hvordanman kan skille mellom hendelser i forbindelsemed videoovervåkning, og videre hva som kan klassifiseres som en unormal hendelse. Å analysere video, uten å vite når, hvor eller om det i hele tatt har foregått noe unormalt skjer hele tiden. Dette er veldig tidskrevende arbeid. I denne type arbeid er personen interessert i hendelser utenom det vanlige. Å ikke ha tilgjengelig de rette verktøy kan for en person være en ganske frustrende oppgave. Personen må også være klart subjektiv i hva han mener å være en unormal hendelse. Noe som helt klart er forskjellig fra person til person.

Er det mulig å lage et system som klarer å modellere unormale hendelser? Hva skiller en unormal hendelse fra en normal? Hvordan kan man skille mellom mindre eller mer unormale hendelser som tar sted? Hvem bestemmer hva som kan klassifiseres som en unormal hendelse?

Preface

This master thesis has been carried out at the Department of Computer Science and Media Technology at Gjøvik University College. The purpose has been to investigate if it is possible to create a system for modeling salient events in a surveillance scenarios. The thesis work show that extracting a set of features from objects, and analyze the relationship between the features, it is possible to model a specific event.

I would like to thank my head supervisor Dr.Tech. Faouzi Alaya Cheikh for very good technical help, advices, motivation, and for following me up during the thesis period.

Kenneth Ellingsen, 2008/07/01

List of Figures

List of Tables

Contents

1 Introduction

1.1 Purpose of this thesis

The purpose of this thesis is to look into the possibility of modeling abnormal events for automatic detection. By analyzing objects behaviors in video sequences over time it should be possible to define a set of criteria's for events, which deviate from normal behavior. Before doing this there must be a clear understanding what characterizes as an event in a surveillance scenario, and what distinguishes events in the sense of what is abnormal and what is not.

1.2 Problem description

The primary goal of this thesis propose an algorithm for automatic detection of abnormal events in video surveillance scenarios. We specifically focus our attention on the event of object dropping in public places such as airports and train stations etc.

Each day there is an immense amount of surveillance data accumulated, and it is usually monitored by very few human eyes, relative to the number of cameras, therefore it becomes almost impossible to detect and respond to a abnormal event as they are happening.

Analyzing surveillance data, without the knowledge of when and where or even if an interesting event has occurred often takes place. In this kind of analysis the analyst is interested in extraordinary events, something that deviates from the normal. Not having the suitable tools it can be a daughnting task for the analyst, consisting of sequentially viewing all raw video data, and using his judgement to determine if an event is unusual.

The United Kingdom is very a good example of the huge deployment of surveillance cameras. The exact number of CCTV [1] cameras in the United Kingdom is not known. A 2002 working paper by Michael McCahill and Clive Norris [1] estimated the number of surveillance cameras in private premises in London to be around 500000, and the total number of cameras in the United Kingdom to be 4,2 million. This means that the UK has one camera for every 14 of its citizens.

The current most accurate method, for abnormal event detection, is by manual labor. Meaning that an analyst sits and pay close attention to multiple screens of live video feeds from surveillance cameras. Most of the video data being recorded is not in the analyst interest at all. To look through all the video data afterwords in search for something unusual that took place e.g. a week ago, is a highly laborious and time consuming. Therefore, there is a clean benefit in making this process automatic.

A paper by Mecocci et al. [2] introduce a way to, automatically and in real-time, detect abnormal behavioural events. Video surveillance systems today show two main drawbacks, namely, not adaptive to different operative scenarios (they only work in well known and structured world), and they generally need the assistance of a human operator in order to recognize and tag spesific visual events. The authors describe a system

[1] Closed-circuit television (CCTV) with which the picture is viewed or recorded, but not broadcasted. Initially developed as a means of security for banks. Today it has developed to the point where it is simple and inexpensive enough to be used in home security systems, and for everyday surveillance.

capable of automatically adapting to different scenarios without any human intervention, and use robust selflearning techniques to automatically learn the typically behaviour of the targets in each spesific operative environment. It uses an improved version of the Altruistic Vector Quantization algorithm (AVQ), which is a rework of the original idea by Johnson et. al. [3]. The system is capable of running in real-time, after a preliminary run-in period of about 40 minutes, in order to learn all the typical visual trajectories.

The overall goal of this thesis is to extract simple and reliable features which are descriptive, i.e. can be used by an unsupervised algorithm to discover the important image features in a video data set in order to detect unusual events.

One problem is training a system that can detect these abnormal events. There are several questions that needs to be answered to be able to get a system like this up and running. How does one determine what stands out as an remarkable event in a surveillance video, and also an important question would be who decides on what is unusual and what is not?

1.3 Research questions

The research questions for this thesis are the following:

1. What is an event in a surveillance scenario?
2. What is classified as a remarkable event in a surveillance scenario?
3. Is it possible to model (predefine behavior) remarkable events in surveillance video?
4. How to distinguish between remarkable events and less remarkable events?

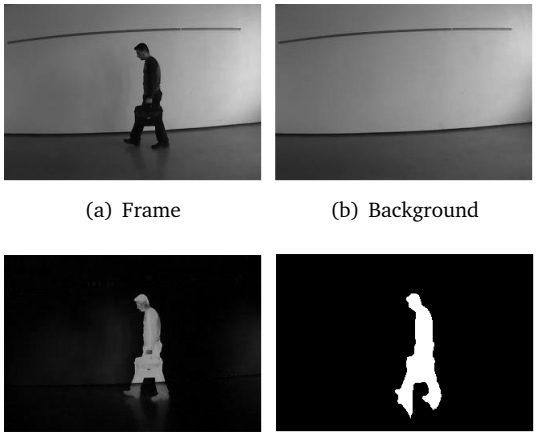
1.4 Thesis structure

In the next chapter we will discuss and outline some of the general related work in the area of event detection, object tracking and abnormal event. In the third chapter we discuss a little about the methods used in this project. Then the fourth, fifth and sixth chapters goes into depth on the implementation, experiements and testing correspondingly. Finally, we draw conclusions of the work accomplished and the retrieved results, and list the future work in the chapters eight and nine.

2 Related work

2.1 Background subtraction

Background subtraction is a commonly used class of techniques for segmenting out objects of interest in a scene for applications such as surveillance. It involves comparing an observed image with an estimate of the image if it contained no objects of interest. The areas of the image plane where there is a significant difference between the observed and estimated images indicate the location of the objects of interest. The name 'background subtraction' comes from the simple technique of subtracting the observed image from the estimated background image and thresholding the result to generate the objects of interest. An example of this is shown in figure 1.



(a) Frame (b) Background

(c) Absolute difference between a and b (d) Binary

Figure 1: An example of background subtraction to generate a result of the object of interest.

In [4] an adaptive background estimation algorithm for outdoor video surveillance system is proposed. In order to enhance the ability of adaptation to illumination changes and variant noise in long-term running, an improved Kalman filtering model based on local-region is discussed to dynamically estimate a background image, in which the parameters are predicted by a recursive-least-square adaptive filter. The experimental results on real-world videos show that the algorithm can perform robustly and effectively.

The central idea behind background modeling and subtraction is to create a probabilistic representation of the static scene that is compared with the current input to perform subtraction. Such approach, as discussed in [5] is efficient when the scene to be modeled refers to a static structure with limited perturbation. This paper addresses the problem of modeling dynamic scenes where the assumption of a static background is not valid, and is inspired by the work proposed in [6].

Background subtraction can be considered as a valuable low-level visual processing step toward performing high-level tasks like motion analysis, motion estimation, and ob-

ject tracking etc. To this end one has to obtain a representation of the background, update this representation over time and compare it with the actual input to determine areas of discrepancy. Such methods have to be adaptive and able to deal with changes of the illumination conditions. Image averaging over a certain window of time is a computationally efficient approach to provide a fair description of the static scene in the absence of moving objects. A step further involves the use of continuous functions to better describe the illumination behavior of such a scene. Under the assumption of limited and smooth variation, in [7] a Kalman-filter driven approach was proposed to capture the background properties while in [8] the use of a single Gaussian distribution was considered.

In [9] the authors propose a novel thresholding approach based on a new color space model and embed it into background subtraction. This model uses each pixel's color distortion and brightness distortion to detect the changes. The color distortion considers the vector's position in color space so that it can assemble the color features effectively. Moreover, this thresholding method also removes the moving shadow to some extent. By applying it to background subtraction, one obtain a comparative complete foreground object. For relatively complex background, the authors are motivated by the hysteresis threshold in Canny edge detection algorithm and introduce dual-threshold into background subtraction. It also achieves robust detection for videos containing complex background. Comparing their method with other multi-model techniques, and the test results show the feasibility of the proposed algorithm.

Being able to track moving targets is one of the important tasks in autonomous systems such as security and surveillance systems and unmanned aerial vehicles (UAV). When the camera system is stationery, background subtraction is a commonly used technique for segmenting out moving objects in a scene. In [10] the authors present two filters designed to remove the clutter in the resultant image of background subtraction technique. One is designed for the images with completely static background such as images taken inside a building. The other is capable of removing the clutter created by slight motions in the background such as motion of leaves of a tree due to wind. Background subtraction does not work when the camera system is moving. For this case the authors present a new technique that is developed using segmentation and coloring technique to isolate the targets of interest from the background. Upon the target of interest is extracted they calculate the position and the velocity of the target using the geometry of the camera system.

Automatic understanding of events happening in a scenario is the ultimate goal for many visual surveillance systems. Higher level understanding of events requires that certain lower level computer vision tasks be performed. These may include detection of unusual motion, tracking targets, labeling body parts, and understanding the interactions between people. To achieve many of these tasks, it is necessary to build representations of the appearance of objects in the scene. In [11] the authors focuses on two issues related to this problem. First, construct a statistical representation of the scene background that supports sensitive detection of moving objects in the scene, but is robust to clutter arising out of natural scene variations. Second, build statistical representations of the foreground regions (moving objects) that support their tracking and support occlusion reasoning. The probability density functions (pdfs) associated with the background and foreground are likely to vary from image to image and will not in general have a known parametric form. Accordingly utilize general nonparametric kernel density estimation techniques for

building these statistical representations of the background and the foreground. These techniques estimate the pdf directly from the data without any assumptions about the underlying distributions.

The idea presented in [12] is subtracting the current image from a time-averaged background image will leave only non stationary objects. It is, however, a crude approximation to the task of classifying each pixel of the current image; it fails with slow-moving objects and does not distinguish shadows from moving objects. The basic idea of this paper is that classifying each pixel using a model of how that pixel looks when it is part of different classes. By learning a mixture-of-Gaussians classification model for each pixel using an unsupervised technique, an efficient incremental version of EM. Unlike the standard image-averaging approach, this automatically updates the mixture component for each class according to likelihood of membership; hence slow-moving objects are handled perfectly. This approach also identifies and eliminates shadows much more effectively than other techniques such as thresholding.

2.2 Object tracking

2.2.1 General

Object tracking is the process of locating a moving object, or several ones, in time using a camera in e.g. surveillance scenario. An algorithm analysis the video frames and outputs the location of moving targets within the video frame.

The main difficulty in video tracking is to associate target locations in consecutive video frames, especially when the objects are moving fast relative to the frame rate. Here, video tracking systems usually employ a motion model which describes how the image of the target might change for different possible motions of the object to track.

Examples of simple motion models are;

- To track planar objects, the motion model is a 2D transformation (affine transformation or homography) of an image of the object (e.g. the initial frame).
- When the target is a rigid 3D object, the motion model defines its aspect depending on its 3D position and orientation.
- For video compression, key frames are divided into macro blocks. The motion model is a disruption of a key frame, where each macro block is translated by a motion vector given by the motion parameters.
- When the image of deformable objects can be covered with a mesh, the motion of the object is defined by the position of the nodes of the mesh.

The role of the tracking algorithm is to analyse the video frames in order to estimate the motion parameters. These parameters characterize the location of the target.

2.2.2 Common algorithms

There are two major components of a visual tracking system; Target Representation and Localization, and Filtering and Data Association.

The first component is mostly a bottom-up process. Typically the computational complexity for these algorithms is low. The following are some common Target Representa-

tion and Localization algorithms;

- Blob tracking, segmentation of object interior (for example blob detection, block-based correlation or optical flow).
- Kernel-based tracking as used in [13], which is an iterative localization procedure based on the maximization of a similarity measure.
- Contour tracking, detection of object boundary (e.g. active contours or Condensation algorithm [1]).
- Visual feature matching.

The second component is on the other hand a top-down process, which involves incorporating prior information about the scene or object, dealing with object dynamics, and evaluation of different hypotheses. The computational complexity for these algorithms is usually much higher. The following are some common Filtering and Data Association algorithms;

- Kalman filter, an optimal recursive Bayesian filter for linear functions and Gaussian noise.
- Particle filter as described in [14] written by Arulampalam et al. from 2002, which is useful for sampling the underlying state-space distribution of non-linear and non-Gaussian processes.

2.2.3 Proposed tracking methods

In [15] they propose the exploitation of a dynamic pogramming technique for efficiently comparing people trajectories adopting an encoding scheme that jointly takes into account both the direction and the velocity of movements.

The novelty of this paper is the adoption of a dynamic programming technique to efficiently compare trajectories. Its a technique borrowed from bioinformatics, allows for coding of the trajectory as a sequence of directions and velocities and to compare them pairwise in an efficient way. Then, dynamic programming is exploited to compute the global alignment score between each pair of trajectories in the training phase. Resulting scores are then clustered by means of k-medoids clustering algorithm [2].

After the clustering, each class memberships likelihood is modeled with a 1-D Gaussian distribution over the distance from the medoid of the class. Finally, in the testing phase, a MAP (Maximum A Posteriori) approach assigns a new trajectory to the trajectory class with the highest posterior probability.

Another tracking approach is comes from Mecocci et al. [16] which describes an automatic real-time video surveillance system which is capable of autonomously learning and

[1] The condensation algorithm (**Con**ditional **Dens**ity Propag**ation**) is a computer vision algorithm. The principal application is to detect and track the contour of objects moving in a cluttered environment

[2] The K-medoids algorithm is a clustering algorithm related to the K-means algorithm. Both algorithms are partitional (breaking the dataset up into groups) and both attempt to minimize squared error, the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the K-means algorithm K-medoids chooses datapoints as centers (medoids or exemplars).

signaling anomalous activities of moving objects. For a system to obtain the capabilities of learning, the authors propose in this paper an improved version of the Altruistic Vector Quantization algorithm (AVQ). This modified version is automatically evaluating the number of trajectory prototypes, and improves the representativeness of the prototypes themselves. So the visual events can be easily and accurately classified.

W4 [17] is a real time visual surveillance system for detecting and tracking multiple people and monitoring their activities in an outdoor environment. It operates on monocular gray-scale video imagery, or on video imagery from an infrared camera. W4 employs a combination of shape analysis and tracking to locate people and their parts (head, hands, feet, torso) and to create models of people's appearance so that they can be tracked through interactions such as occlusions. It can determine whether a foreground region contains multiple people and can segment the region into its constituent people and track them. W4 can also determine whether people are carrying objects, and can segment objects from their silhouettes, and construct appearance models for them so they can be identified in subsequent frames. W4 can recognize events between people and objects, such as depositing an object, exchanging bags, or removing an object. It runs at 25 Hz for 320x240 resolution images on a 400 MHz dual-Pentium II PC.

Abnormal behaviors are detected if visual trajectories deviate from the self-learned representations of 'typical' behaviors. As in [2] from 2003, if the surveillance cameras field-of-view is changed, the system automatically re-learns new 'typical' behaviours, and accurately detects anomalous events.

The trajectories of moving objects can provide crucial clues for video event analysis especially in surveillance systems. A study of the problem of detecting anomalous events by analyzing the motion trajectories are studied in [18] by Zhou et al. Different trajectories of the same category may have varying relative velocities, in addition to the variations and noises in location samples; hence the core of the problem is to provide a robust and accurate function for measuring the similarities of trajectory pairs.

Zhou et al. proposed a novel learning based algorithm for estimating the similarities of the multi-dimensional sequence pairs, and then an anomaly detection framework is presented to detect anomalous motion trajectories in surveillance videos. Their proposed algorithm offers several advantages over the traditional algorithms for dealing with the trajectories of moving objects. First, the similarity measurement is robust against data imperfections such as noise, algorithmic error and etc. Second, they introduce a learning algorithm which allows the similarity function to be adapted to the particular problems being solved. Third, the proposed anomaly detection framework is fully automatic and without parametric distribution assumption on the data.

Tracking multiple moving objects in image sequences involves a combination of motion detection and segmentation. This task can become complicated as image motion may change significantly between frames, like with camera vibrations. Such vibrations make tracking in longer sequences harder, as temporal motion constancy can not be assumed. A method is presented for detecting and tracking objects in [19], which uses temporal integration without assuming motion constancy. Each new frame in the sequence is compared to a dynamic internal representation image of the tracked object. This image is constructed by temporally integrating frames after registration based on the motion computation. The temporal integration serves to enhance the region whose motion is being tracked, while blurring regions having other motions. These effects help motion

analysis in subsequent frames to continue tracking the same motion, and to segment the tracked region.

In [20] the authors describe a vision system that monitors activity in a site over extended periods of time. The system uses a distributed set of sensors to cover the site, and an adaptive tracker detects multiple moving objects in the sensors. Their hypothesis is that motion tracking is sufficient to support a range of computations about site activities. It is demonstrated using the tracked motion data: to calibrate the distributed sensors, to construct rough site models, to classify detected objects, to learn common patterns of activity for different object classes, and to detect unusual activities.

2.3 Feature extraction

In computer vision and image processing the concept of feature extraction refers to methods that aim at computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions.

It involves simplifying the amount of resources required to describe a large set of data accurately. When performing analysis of complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power or a classification algorithm which overfits the training sample and generalizes poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy.

Feature extraction has been an important research topic in pattern recognition or pattern classification, data mining, machine learning and has been studied extensively by many researchers. Most of the conventional feature extraction methods are performed using a criterion function defined between two classes or a global function. Although these methods work relatively well in most cases, it is generally not optimal in any sense for multiclass problems. In order to address this problem, the authors of [21] propose a method to optimize feature extraction for multiclass problems. The authors first investigate the distribution of classification accuracies of multiclass problems in the feature space and find that there exist much better feature sets that the conventional feature extraction algorithms fail to find. Then the authors propose an algorithm that finds such features. Experiments with remotely sensed data show that the proposed algorithm consistently provides better performances compared with the conventional feature extraction algorithms.

In a paper by Peng et al. [22] a study on how to select good features according to the maximal statistical dependency criterion based on mutual information is researched. Because of the difficulty in directly implementing the maximal dependency condition, the authors first derive an equivalent form, called minimal-redundancy-maximal-relevance criterion (mRMR), for first-order incremental feature selection. Then, present a two-stage feature selection algorithm by combining mRMR and other more sophisticated feature selectors (e.g., wrappers). This allows to select a compact set of superior features at very low cost. The authors perform extensive experimental comparison of the algorithm and other methods using three different classifiers (naive Bayes, support vector machine, and

linear discriminate analysis) and four different data sets. The results confirm that mRMR leads to promising improvement on feature selection and classification accuracy.

The authors of [23] introduce the information theory, propose a new concept of probability information distance (PID) and prove that the PID satisfies four requests of axiomatization of the distance. So the PID is a kind of distance measure, which can be used to measure the degree of variation between two random variables, which could be two object in a scene. The authors make the PID be separability criterion of the classes for information feature extraction, and call it PID criterion (PIDC). Based on PIDC, a novel algorithm for information feature extraction is designed. Compared with principal components analysis (PCA), correlation analysis etc., the algorithm put forward in this paper had regarded for the class information, and so it is a kind of supervised algorithm of feature extraction. The experimental results demonstrate that the algorithm is valid and reliable, and it provides a new research approach for feature extraction, data mining and pattern recognition.

Low-level features, as mentioned in [24], are defined to be those basic features that can be extracted automatically from an image without any shape information (information about spatial relationship). Thresholding is a form of low-level feature extraction performed as a point operation. All of these approaches can be used in high-level feature extraction, where we find shapes in images.

2.4 Feature analysis

Instead of only learning the trajectory of an object's movement one can perform, in addition, a content-based analysis of blobs, and for this use shape-analysis. The shapes are frequently extracted and saved, and later used as prototypes as part of a learning system.

In [25] it is presented an event detection framework using multimodal feature analysis. In this framework, multimodal features are extracted from video data and then analyzed to generate various mid-level concepts, such as video shot, face appearance and so on. Two schemes, the logistic regression and Bayesian belief network, are then employed to fuse the information obtained from multimodal feature analysis and detect the video events of interest. The authors aim to use this framework as a general template for event detection in different video domains. Experimental results on various test videos in different video domains suggest that the proposed event detection framework is promising.

In a paper regarding human posture recognition [26] they believe that recognition of posture is one step in the global process of analysing human behaviour.

Usually detecting human behavior is recognized through the study of trajectories and positions of persons and using a priori knowledge about the scene. Boulay et al. describes two methods based on 2D appearances. The application recognize postures on the blobs (binary images). First a determination is made, on which postures to recognize.

The selected postures are of everyday work in offices and are easily recognized, and they are classified into three main categories: the standing posture (figure 2), and the seated posture (figure 3), and the bending posture (figure 4).

Two of the postures, figures 2 and 4 respectively, are highly applicable to the work in this thesis. When an object is about to make a drop, it has the tendency to bend it's upper part slightly.

The authors of [27] suggest an automatic surveillance system based on a robust and

Figure 2: The standing postures; standing with arms near body, standing with arm to left, standing with arm to right, and T-shape.

Figure 3: The seated postures; seated on a chair and seated on the floor.

Figure 4: The bending postures; a person who tries to pick up something on the floor.

simple algorithm for foreground and tracking of multiple objects in complex environments. It is explained that it is possible to obtain enough information from the characteristics of the detected objects. The system analyses the video stream and rises an alarm to call the operators attention if the features fulfill predefined requirements. It is based on very simple algorithms in order to allow a single computer analyze up to eight video streams in real-time. Blob and scene characteristics are used to create a low-level description of predefined events. It detects events by comparing the sequence's parameters with those associated with the events for the current scenario.

2.5 Abnormal event detection

In the field of video processing there are several recently proposed methods for the detection of abnormal events in video surveillance scenarios. Most systems today consist of the steps presented in figure 5.

Figure 5: General system architecture for abnormal event detection.

A paper by Remagnino et al. [28] from 2001 discuss an approach of classifying events unfolding in a surveillance scenario into *types* and *behaviors*. This approach employs a Bayesian classifier to determine type from event attribute such as height, width and velocity. The classifier, however, is extended to integrate all available evidence from the entire track. A typical Hidden Markov Model [3] approach, as seen in the example in figure 6, has been employed to model the common event behaviours typical in this case of a car-park environment. People and vehicles enjoy a distinct velocity width-to-height-ratio characteristics. In this paper object classification has been divided into two common classes, persons and vehicles. They also define two basic event behaviour classifications for each of the common classes. These are vehicles exiting and vehicles entering, and the same applies for persons.

Each type of activity for people or vehicle events may be characterized by a family of event trajectories passing through the image. Each family can be represented as a *hidden Markov model* in which states represent regions in the image, the prior probabilities measure the likelihood of an event starting in a particular region, and the transitional probabilities capture the likelihood of progression from one state to another across the image.

In recent years, real-time direct detection of events by surveillance systems has attracted a great deal of attention. In [29] the authors propose a new video-based surveillance system that can perform real-time event detection. In the background modeling phase, adoption of a mixture of Gaussian approaches is used to determine the background. Meanwhile, the color blob-based tracking is used to track foreground objects. Due to the self-occlusion problem, the tracking module is designed as a multi-blob tracking process to obtain similar multiple trajectories. The authors devise an algorithm to

[3] A hidden Markov model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unknown parameters. The challenge is to determine the hidden parameters from the observable parameters. The extracted model parameters can be used to perform further analysis, e.g. for pattern recognition applications.

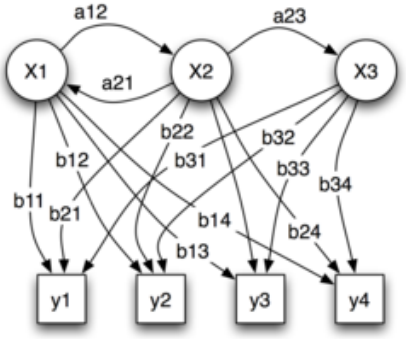


Figure 6: Probabilistic parameters of a hidden Markov model, with states (x), possible observations (y), state transition probabilities (a), and output probabilities (b).

merge these trajectories into a representative one. After applying the Douglas-Peucker algorithm to approximate a trajectory, it can compare two arbitrary trajectories. The above mechanism enables to conduct real-time event detection if a number of wanted trajectories are pre-stored in a video surveillance system.

For detecting abnormalities, by using a clustering-based approach, in surveillance videos requires the appropriate definition of similarity between events [30]. Here they propose a multi-sample-based similarity measure, where HMM training and distance measuring are based on multiple samples. These multiple training data are acquired by a novel dynamic hierarchical clustering (DHC) method. By iteratively reclassifying and retraining the data groups at different clustering levels, the initial training and clustering errors due to overfitting will be sequentially corrected in later steps. This algorithm can dynamically correct initial overfitting errors as the number of training samples increase (i.e. data clusters become larger). In addition, it is not sensitive to the absolute values of similarity, because simple comparison operation instead of eigenvalue decomposition is needed in the proposed approach.

The optimal system for abnormal event detection should be fully automatic and process video data in real time. Mecocci et al. [2] introduces a way to, automatically and in real-time, detect abnormal behavioural events. Video surveillance systems today show two main drawbacks, namely, not adaptive to different operative scenarios meaning that they only work in well known and structured world, and they generally need the assistance of a human operator in order to recognize and to tag spesific visual events. The paper describes a system capable of automatically adapting to different scenarios without any human intervention, and uses robust selflearning techniques to automatically learn the typically behaviour of the targets in each spesific operative environment.

The system, as seen in figure 7, uses an improved version of the Altruistic Vector Quantization algorithm (AVQ), which is a rework of the original idea by Johnson et. al. [3]. The system is capable of running in real-time, after a preliminary run-in period of about 40 minutes, in order to learn all the typical visual trajectories.

Another approach for recognizing unusual events in video are based on key frame extraction. Which can be implemented by dividing a video sequence into overlapping parts, and for each part extract the three most relevant key frames. In [31] these key

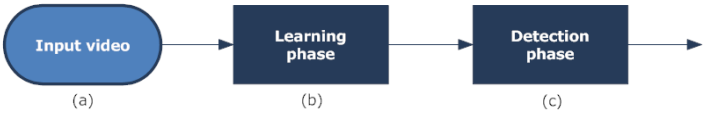


Figure 7: (a)Trajectories of all moving objects in the field-of-view. Each of the trajectories consists of a series of centroid positions for each object at each frame, (b)the Anomaly Detection System (ADS) module must learn, through the AVQ algorithm, the statistic behavior of the moving object, and (c)anomaly detection is performed.

frames are used to assign a curveness measure to each part. The higher the curveness value of each part is, the more unpredictable events it presents. This feature is the basis for extracting unpredictable event in surveillance video.They map a video sequence to a polygonal trajectory by mapping each frame to a feature vector and joining the vectors presenting consecutive frame by line segments. Shape analysis of the obtained polygonal curve allows us to detect frames representing unpredictable events as seen in figure 8.

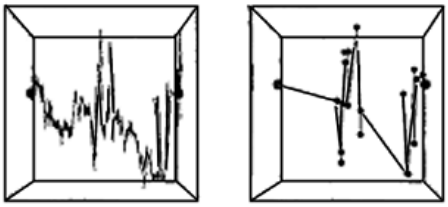


Figure 8: (a)video trajectory with 2379 vertices from a video clip. (b)a simplified polygon with 20 most relevant frames (black dots).

The system presented in [32] takes as input a video stream obtained from an airborne moving platform and produces an analysis of the behavior of the moving objects in the scene. To achieve this functionality, the system relies on two modular blocks. The first one detects and tracks moving regions in the sequence. It uses a set of features at multiple scales to stabilize the image sequence, that is, to compensate for the motion of the observe,then extracts regions with residual motion and uses an attribute graph representation to infer their trajectories. The second module takes as input these trajectories, together with user-provided information in the form of geospatial context and goal context to instantiate likely scenarios. The approach is based on two modules. The first one achieves detection and tracking of moving objects from the video stream collected by the Unmanned Airborne Vehicle (UAV), while the second takes the inferred trajectory of each object and user-provided contextual information to recognize the behavior of the moving objects among a large number of potential scenarios.

The proposed system in [33], aims at performing two tasks real time; detection of abandoned or stolen objects. For what concerns abandoned objects, it is assumed that the cause of an alarm consists in the person who has left the object in the scene, while for stolen objects, the cause of an alarm consists in the person who has taken an object from the guarded environment. The system global structure allows a functional subdivision in three different processing levels. More precisely each level is organized in sub-modules,

devoted to specific task. In low-level image processing module mainly point-based operations on single acquired image pixels are performed. These levels identify the pixel region, blobs, through the respective bounding boxes coordinates. The middle-level image processing modules provide the interface between the low level modules working at pixel level and the high level ones which working at features level. The high-level modules process the data, extracted in the previous module and through classification they interpret the scene and notify the operator with an alarm.

Modelling events is one of the key problems in dynamic scene understanding when salient and autonomous visual changes occurring in a scene need to be characterised as a set of different object temporal events. The authors of [34] propose an approach to understand complex outdoor scenarios which is based on modelling temporally correlated events using dynamic Bayesian networks (DBNs). A partially coupled hidden Markov model (PCHMM) is exploited whose topology is determined automatically using the Bayesian information criterion (BIC). Causality discovery and events modelling are also tackled using a multi-observation hidden Markov model (MOHMM).

With concerns about terrorism and global security on the rise, it has become vital to have in place efficient threat detection systems that can detect and recognize potentially dangerous situations, and alert the authorities to take appropriate action. Of particular significance is the case of unattended objects in mass transit areas. It is described in [35] a general framework that recognizes the event of someone leaving a piece of baggage unattended in forbidden areas. The approach involves the recognition of four sub-events that characterize the activity of interest. When an unaccompanied bag is detected, the system analyzes its history to determine its most likely owner(s), where the owner is defined as the person who brought the bag into the scene before leaving it unattended. Through subsequent frames, the system keeps a lookout for the owner, whose presence in or disappearance from the scene defines the status of the bag, and decides the appropriate course of action.

In [36] the authors present an unsupervised technique for detecting unusual activity in a large video set using many simple features. There are no complex activity models and no supervised feature selections used. The video is divided into equal length segments and extracted features are classified into prototypes, from which a prototype-segment co-occurance matrix is computed.

A method to utilize the *hard to describe* but *easy to verify* property of unusual events without building *explicit models* of normal events are presented here. One compare each event with all others observed to determine how many *similar* events exists. If there are many similar events, one can conclude that the events are most likely to be *normal*, and not of much interest. If an event is not similar to other it is most likely to be an unusual and interesting event. Detecting unusual events in a large data set does not require modeling of normal events, but rather the ability to compare two events and measure their similarity.

The overall goal is to extract simple and reliable features which are descriptive, which is similar to this thesis, i.e. can be used by an unsupervised algorithm to discover the important image features in a large video set in order to detect any unusual events.

3 Choice of methods

The work of this thesis was based on a combination of two scientific methods. One part consisted of a literature study, and another part consisted of an experimental research [37].

The literature study gave the opportunity to discover the existing knowledge and work in the field of abnormal event detection. Tis was the first part of the entire process, and also and ongoing process during the whole project period.

The experimental research was attempted by us to maintain control over all factors that may affect the result of an experiment. In doing this, we attempted to predict what could occur.

Before the developing stage the test video was observed and analyzed. It was made a description of the abnormal event of object dropping. It was done to make a clear understanding of what the object in the video was doing before, during and after the event had occurred. The result of the visual and subjective analysis was a clear knowledge of the link between several distinctive features.

Throughout the development- and experimental stage the test video went through repeated testing with well known factors involved. The test video was chosen because of its clear and straightforward presentation of the event in question. The video was ran through algorithms for feature analysis and event detection written in Matlab. Since the test video was analyzed a priori to the computer simluations during the development we had a knowledge about what we were supposed to look after, and in that sense we could ensure a certain validity to the result.

The validity being, instead of developing a program to be fitted to a specific video, to yield the desired result, the a priori analysis gave basis for specific measured properties to act as threshold values.

4 Algorithms and implementation

4.1 System architecture

The chart displayed in figure 9 outlines the main building blocks of the systems. Starting at the input video-element in the top-left-corner of the figure, and following the arrows, one is able to interpret the flow of the system.

The system consists of four major blocks. These are;

- Background estimation
- Object tracking
- Feature extraction
- Feature analysis

Given an input video source, the system first determines the background of the scene. Next, it reads frame by frame and starts by tracking moving objects subtracted from the background. Each of the objects have several features extracted from it, and in the final stage these features are analyzed to determine whether the event of object dropping has taken place or not

Each of the blocks, and their parts, are described in-depth in the section 4.3.

4.2 Matlab functions

The implementation part of the thesis has been developed in MATLAB [1]. A list of all the functions that are part of the abnormal event detection system can be found in table 1.

4.2.1 Description of functions

The Tracking-function is the main function of the entire system. It is invoked by calling it and sending with a video file as input. The function calls all other functions, automatically, in a spesific order. The output yields either true or false based on the whether the analysis determine the happening of the event of object dropping.

The Background-function makes the determination of what is considered as the background in the scene. It utilizes two other functions; the MeanPixelIntensity-function and StandardDeviation-function. The input to this function is an userdefined number of frames. The mean pixel intensity is computed on this set of frames, and one frame which regarded as the background is returned. It also computes the standard deviation on the same set of frames which is stored in a new separate frame. Both of these new frames are passed on to the next function.

FramePreProcessing, have as input the two frames computed in the Background-function, a confidencelevel-variable, and the first frame of the video. Each frame gets

[1] MATLAB is a numerical computing environment and programming language. Created by The MathWorks, MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages.

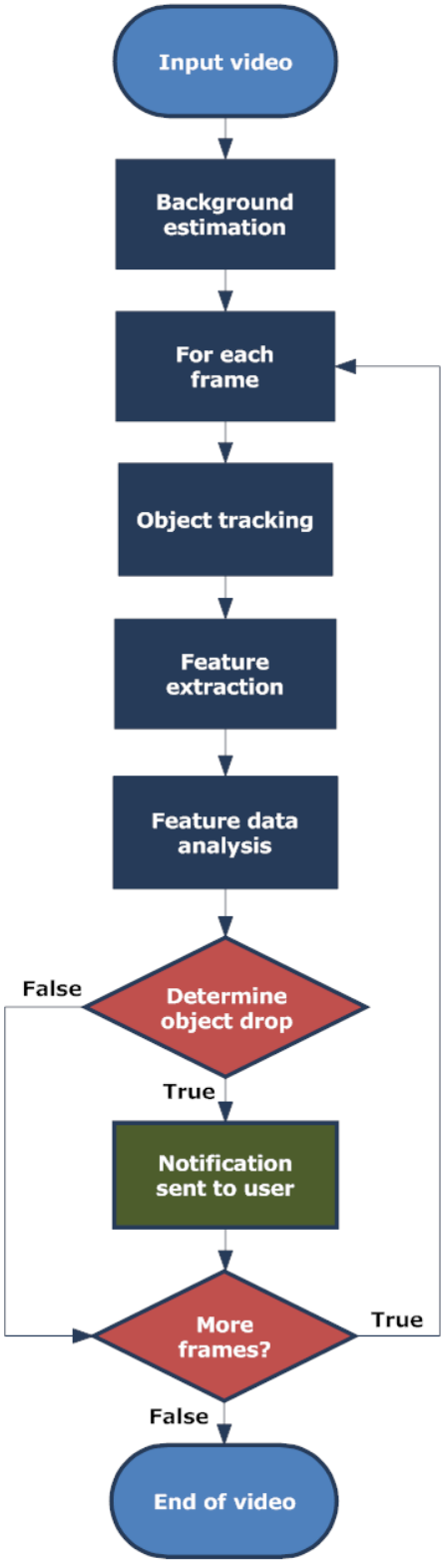


Figure 9: System architecture.

List of functions	
Name	**Description**
Tracking	Main-file which invokes all other functions.
Background	Computes the background image.
MeanPixelIntensity	Compute the mean intensity of pixels of a user defined set of frames.
StandardDeviation	Compute the standard deviation of all pixels in the same set of frames as in MeanPixelIntensity.m.
FramePreProcessing	Takes a color frame as input, converts it to binary, performs morphological operations on it, and outputs a binary image.
Gray2Binary	Makes a binary image based on the absolute difference between the current frame and the mean background image, the standard deviation, and a user defined confidencelevel.
Bwlabel	Label connected components in binary image.
Regionprops	Measure the properties of image regions (blob analysis).
XYMotion	Compute displacement of blobs based on centroid positions between consecutive frames.
Sort	Sort features in regard to labeling.
Filter	Applies a median filter to the extracted feature data
Plotting	Plots all extracted features in graphs for visual comparison.
FeatureComparison	Analysis of extracted features against user defined criteria's.

Table 1: System functions developed in Matlab

converted to gray-scale image, and further to a binary image using the Gray2Binary-function. The conversion to binary starts by computing the absolute difference between the mean background and the current frame. The result is a binary image that is based on the standard deviation and the confidencelevel-variable used as thresholds. Next, morphological operations are performed on the binary frame, such as removing pixels considered as noise, and all open areas inside blobs are closed. Finally, the Bwlabel-function is utilized to label the connected components (blobs) in the frame. The output is a frame with labeled objects.

Regionprops-function extracts several features from each blob in the binary-frame it takes as input, based on the labels from the previous function. The output from this function is a matrix consisting of several features for each of the blobs in the binary frame.

The XYMotion-function computes the directional information of each blob based on the 'Center of mass'-feature from the Regionprops-function, and outputs two separate matrices, each for the horizontal translation and the vertical translation.

In the Sort-function the feature data from both Regionprops- and XYMotion-function are sorted. The sorting is based on size of the 'Area'-feature from the Regionprops-function. The sorting is done to solve labeling problem described in section 5.3. The output are the sorted versions of the matrices given as input.

After the feature extraction, and data have been sorted the Plotting-function is invoked. It takes the sorted feature data and plot them. This is used to perform a visual analysis.

The final function of the system is the FeatureComparison-function. The input is the set of sorted matrices with all the feature data. Each feature is compared against the user defined criteria's. The function performs analysis on the following four features; area, center of mass, ratio and directional information. Each part yields true or false based on these criteria's, and the system conclude that an object dropping has occurred if all criteria's return true. A user notification is sent.

4.3 Implementation

4.3.1 Background estimation

To be able to track any moving objects in a video, it must be subtracted from the background. This returns an object as the foreground, hence image (c) in figure 1. In this system, to estimate what is considered background and what is not, it starts by reading a user defined number of frames from the video source. The chart in figure 10 outlines the structure of the background estimation process.

The output of the background estimation are two variables; the estimated background image and standard deviation, both of which come from the set of frames given by the user as input.

The estimated background is the mean pixel intensity for a set of frames. The result is a single frame (image) that is representative of the set of frames given as input by the user. The background-image is derived by taking the mean value of all pixels at each position in the set of frames. All mean values of each pixels are stored in a new separate matrix. This image will now be considered as the background of the scene, and used to extract the foreground objects of the future frames.

The second variable which this background estimation process outputs, is the standard deviation. The result of this process, similar to the background estimation, is also a single frame. The standard deviation is computed much in the same way as the background. It processes pixel by pixel, on the same set of frames as used to compute the background, the standard deviation of the pixels are calculated instead of the mean value.

The standard deviation stage has nothing to do with the estimation the background image of the video source. It is used at a later stage to properly subtract any moving objects from the background image, together with a confidencelevel variable, by the mean of thresholding. Which means that any pixel in the current processed frame belongs to the foreground if it's intensity is higher then the standard deviation of that pixel multiplied with a userdefined confidencelevel.

4.3.2 Object tracking

The chart in figure 11 outlines the structure of the object tracking process.

To detect any moving objects in the scene, each frame is subtracted from the mean background image. This results in a foreground image containing one or more objects. This is converted into binary, making the background pixels black and the foreground pixels, scene objects, white. This is done by using the standard deviation and a confidencelevel as a threshold.

Further each frame goes through a morphological filtering step, which removes small objects in the scene. These are regarded as noise elements. Additionally any blobs that are not filled, due to bad segmentation are closed.

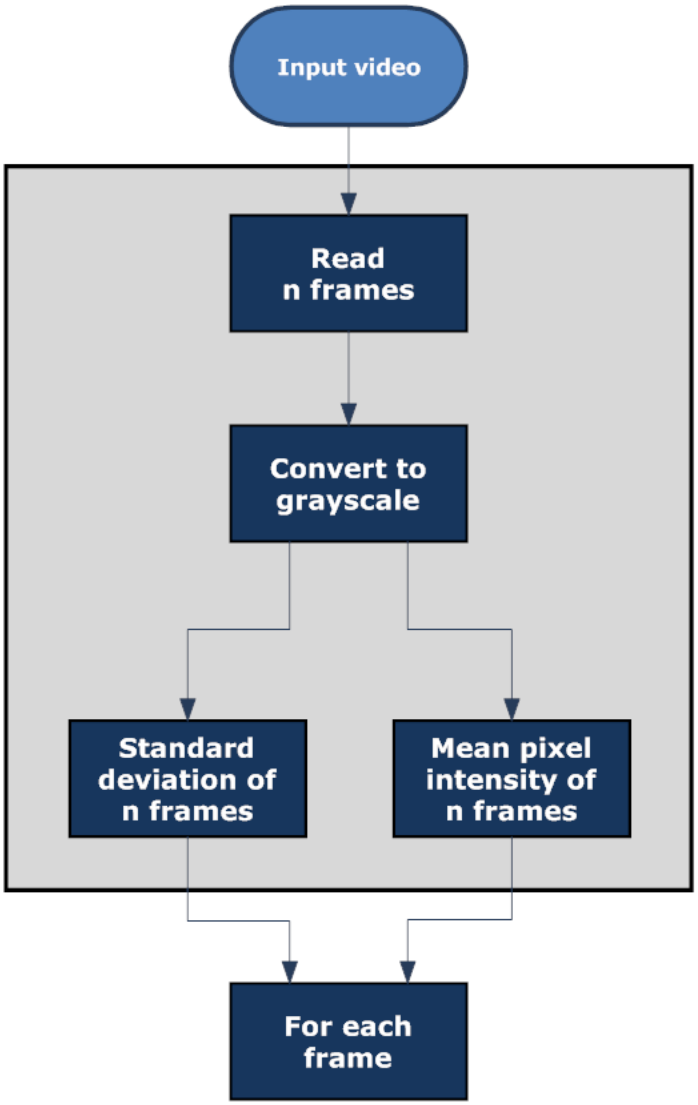


Figure 10: Background estimation outline.

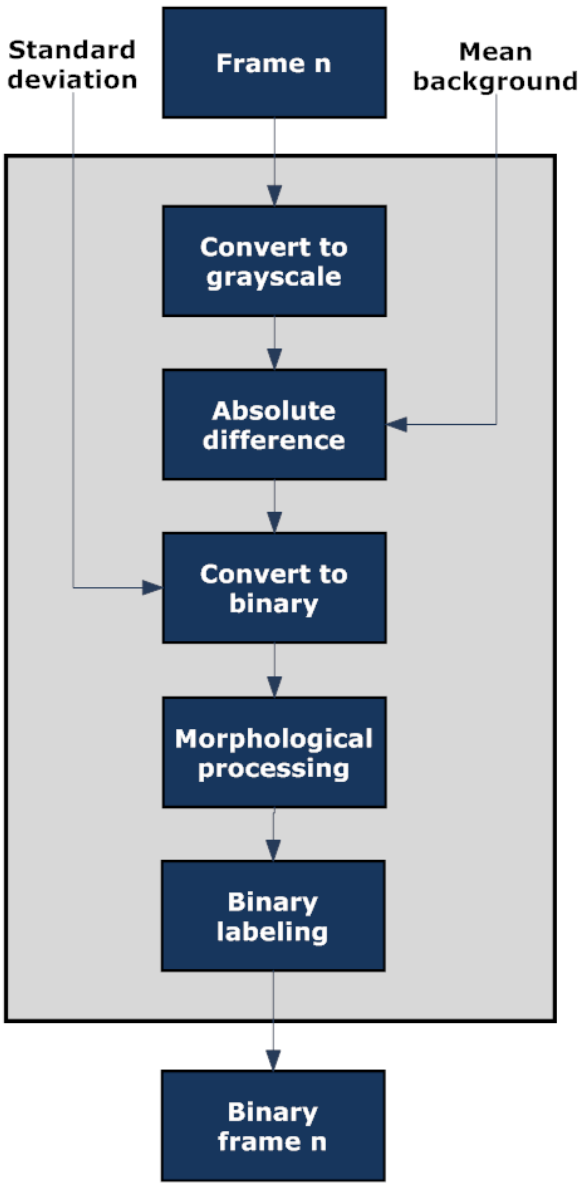


Figure 11: The outline for the object tracking of the system.

The final step in this part of the system, is the labeling of the blobs. The built-in Matlab function *bwlabel.m* labels all blobs in an image from left border to right border. The downside of this function is that the labels are not maintained for the subsequent frames. The blob are always labeled in ascending order starting with label *1* given to the left-most blob in the scene, and so on moving right. Hence the left-to-right labeling process.

In the event of object drop, the blob carrying an object is much bigger then the object it's carrying. The solution for the labeling problem was to sort the blobs according to it's size, giving the biggest blob label *1*, the second biggest blob label *2*, etc. This would mean that after the object drop has taken place, the first blob would still be the biggest, and still have the same original assigned label.

This approach resulted in a successful tracking of the objects in the scene. Result of the labeling experiment in described in-depth in section 5.3.

4.3.3 Feature extraction

The chart in figure 12 outlines the structure for the feature extraction, and it can be seen in the figure the system extracts several features. All the features are extracted for each object present in the seen, regardless of presence or action the object have in the video.

At this stage of the system, for each frame all objects (blobs) in the scene have several descriptive properties measured as seen in 2.

Blob features	
Name	**Description**
Area	Actual number of pixels belonging to a blob
Center of mass	Specifies the center of mass of the region
Directional information	Translation of blob centroids between consecutive frames
Width-height-ratio	Ratio between the width and the height of the blobs bounding box
Numel	Actual number of blobs in current frame
Minor axis	Minor axis of an ellipse which is fitted a blob

Table 2: Blob features to be measured.

The 'Area' is a scalar which represents the size of a blob. It is the actual number of pixels in the region, although this value might differ slightly from the value returned by bwarea, which weights different patterns of pixels differently.

The 'Center of mass' (centroid) is vector of 1-by-n dimensions in length, that specifies the center point of a region. For each point it is worth mentioning that the first element of the centroid is the horizontal coordinate (or x-coordinate) of the center of mass, and the second element is the vertical coordinate (or y-coordinate). All other elements of centroid are in order of dimension.

The figure 13 illustrates the centroid and bounding box. The region consists of the white pixels; the green box is the bounding box, and the red dot is the center of mass.

In the feature 'Directional information', which also is a 1-by-n dimensions in length vector, the translation between centroid-position is stored. It computes the distance, in pixels, for all blobs center of mass points between two consecutive frames. For each blob two vectors are extracted, the first with information about the horizontal displacement

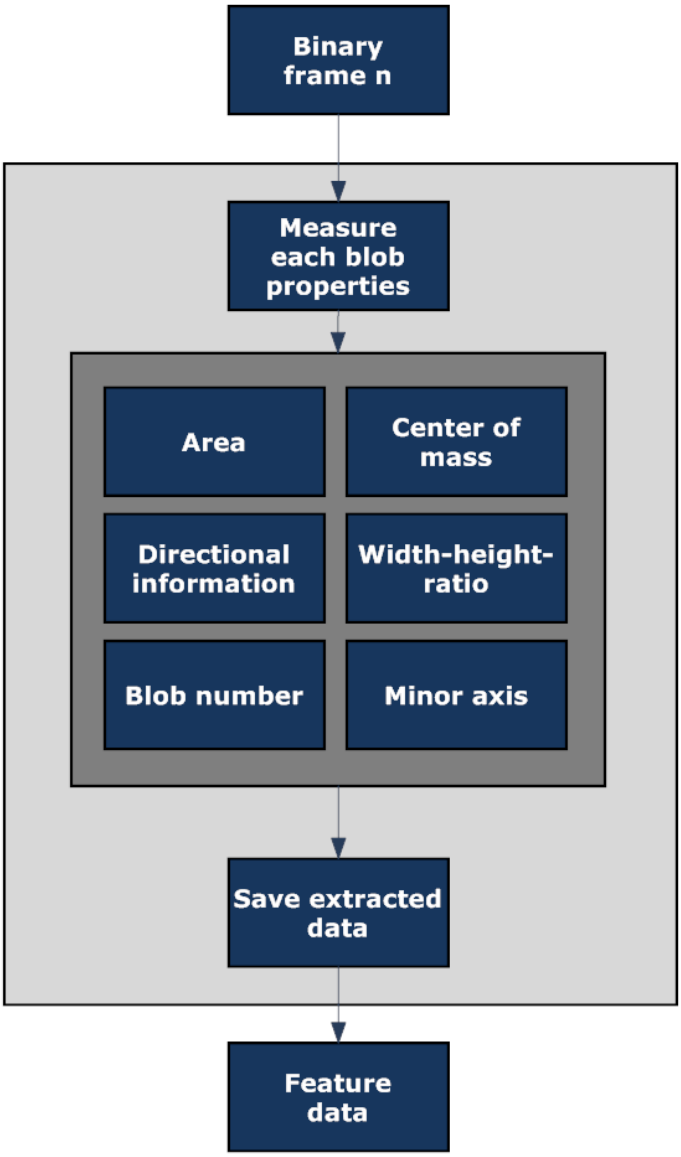


Figure 12: The outline for the feature extraction of the system.

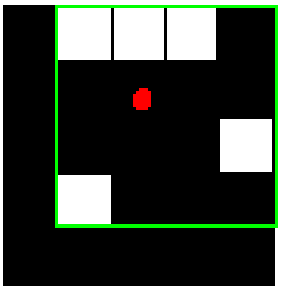


Figure 13: Center of mass, and bounding box of set of pixels.

(in the x-direction), and the second with information about vertical displacement (in the y-direction).

The 'Width-height-ratio' is the scalar of the ratio between the width and the height of the bounding box. Each blob in a frame can be fitted with rectangle representing the outer borders. This feature is a ratio of the width and height of this rectangle.

The 'Numel', which is short for number of elements, contains the actual number of blobs present in any given frame.

The last feature is the 'Minor axis', is a scalar-value. It is the length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region. This property is supported only for 2-D input label matrices. For an object (e.g. person standing), it is possible to fit an ellipse around this blob. The minor axis would represent the approximated width of the object, and on the other side the major axis would give information about the objects height.

4.3.4 Feature analysis

The final part of the proposed system is the feature analysis part. Throughout the processing of the input video, several features are extracted. To detect any abnormal events, the feature data has to be analyzed. For this two sets of data is required, the extracted data from the input video sources, and the user defined criteria's for which describe the event of object dropping.

The feature data has to be analyzed over time. Although, certain features might yield a satisfying result by only comparing two consecutive frames to determine whether an object drop took place, like the 'Center of mass', and the 'Numel'-features. Others need to be examined over a time period.

The systems feature analysis stage is based on the 'Numel'-feature. The analysis do not execute until this parameter increases. Which means there are one or more new objects in the scene. The system then starts to process each feature data separately, and comparing each of them against the individual set of criteria's.

System notification to the user will happen only when the extracted features lay within the range of the user defined criteria's.

A set of predefined user criteria's, selected from several plot-analysis, make up the basis for the analysis are listed in table 3. This table contains the set criteria's defining an object drop.

To lower the computational cost of the system, the feature analysis part shown in figure 14 will not be executed at each frame. This part of the system will be based on the Number of blobs-feature. The **numel**-feature is controlled at each frame. If the difference in value between $\mathrm{frame}_n - \mathrm{frame}_{n-1} >= 1$, then there is new objects in the scene, one or more, and the system would execute the analysis-function, otherwise the system moves to the next frame.

Analysing data at the exact frame of where the presumably object drop took place, against data of the previous frame will most likely not yield reliable results. The feature data analysis should take several frames into consideration, and look at the variation of the data over a time period. Therefore a search window has to be defined.

The search window, which is of a variable size, consists of a number of frames that are related to the input videos attribute of frames per second (fps). The windows size must be based on the framerate of the input video, and not on a preset number frames

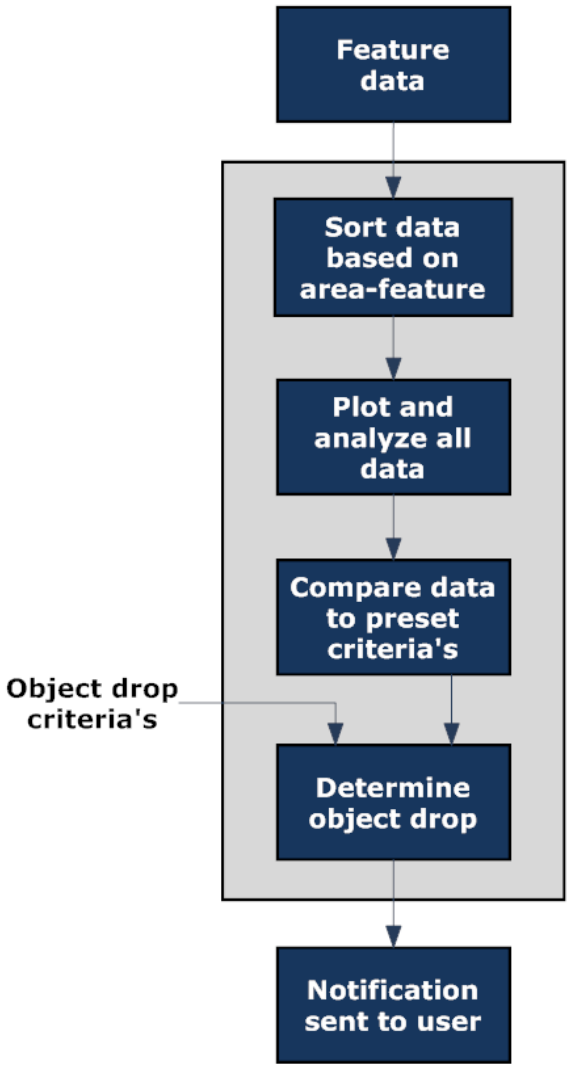


Figure 14: The outline for the feature analysis of the system.

User criteria's	
Feature	**Description**
Area	A sudden significant decrease in the blob size, and no sudden significant increase during previous frames. Significant sudden increase in size of new blob at same frame
Center of mass	Check if distance between current blob, and a new blob. Distance must be equal or less then 2 times the minor axis of the current blob (the minor axis length holds some relation to the size of the object, it's height
Directional information	Objects translation history shows a gradually increase in displacement before the object drop point
Width-height-ratio	A ratio equal or higher then 0.5 accounts for an object standing, or bending slightly
Numel	Number of objects increases by one where the drop takes place
Minor axis	Represent the width of the current blob, and is used as a threshold value

Table 3: User defined criteria's for the abnormal event of object drop.

only. If the window size was set to 10 frames, this could contain very much information if the framerate had been 10 fps. On the other side, a window size of 10 frames could contain very little information if the framerate had been 100 fps.

Search window					
Video	**Frames(#)**	**FPS**	**Start**	**Object drop**	**End**
Drop 1	225	30	118	179	182
Drop 2	134	14	50	87	97
Drop 3	232	25	83	152	154
Drop 4	200	14	49	145	149
Mean		20	65		5
Std.dev.			24		4

Table 4: Time of event statistics, measured in frames, related to object dropping.

Table 4 outline statistics on four different videos related to the event of object dropping. **Frames(#)** gives the total number of frames in the video, and the **Fps** is the framerate. The **start** and **stop** columns represents the beginning and ending of the action related to the object dropping. Everything before and after these frames are considered to be a non-event. The **object drop** gives the frame for which the drop takes place. The rows of **Mean** and **Standard deviation** corresponds to the number of frames before and after the drop.

The actions by an object, both before and after the actual object drop, vary in length. Therefore the search window must be of a variable size. According to table 4 the time of related action before the object drop is considerable longer then after the object drop. From this one can conclude that the search window does not need to be of similar length on both sides of the object drop, since most related action happen before the actual

dropping.

The feature analysis consist of comparing analyzing data of four features; area, center of mass, ratio and directional information. The analysis-sage is not executed at everyframe, but is rather dependent on the change in value of the 'Numel'-feature. If this feature increases in value by 1, the system knows that a second object has entered the scene and begin analysing the extracted data. Based on the framerate of the input video, the searchwindows size is defined, and with this parameter a spesific number of frames are taken into consideration when analysing the data.

The 'Area'-feature utilize data from both the first and second object. Vectors with the difference between the consequtive frames are computed. The absolute difference is not applied here. In order to examine any possible significant variations in size one need to take into consideration both positive and negative values. The two new vectors make up the basis for a third vector, which contains the absolute difference between the first two vectors. Since the first two vectors have both positive and negative values, the frame where the first object descrease in size and the second object increase in size generates a significant difference between the two values. The standard deviation of the third vector acts as a threshold. Most values in the third vector, inside the searchwindow, do not deviate much from the original values except at the point of the assumed drop.

The analyses of the 'Center of mass'-feature does not take into consideration any history of the first objects movement. It computes the Euclidian distance between the first and second object center of mass points where assumed object dropping takes place. To determine whether the object are close enough to come from the same object, a threshold is applied. For a threshold-value the 'Minor axis'-feature is utilized. When the first object have a standing posture. The minor axis of this blob can be said to represent the objects width. If the distance between the to center points are equal or less then two times the minor axis they are regarded as close.

The 'Ratio'-feature is analyzed based on data inside the defined searchwindow. Both the mean and the standard deviation of the feature data for the first object inside the searchwindow is computed. This is performed to see the variations in the ratio of the bounding box of the object. A ratio at approximately 0.5 account for an object standing or slightly bending. If the ratio is bigger then 0.5 the object might be bending as to put down an object on the ground. The algortihm checks whether the ratio is bigger then the searchwindows mean-value and standard deviation. If so, the object's ratio have increased enough to determine the bending posture.

The final comparison in order to determine an occurance of the event of object dropping, is the analyses of the 'Directional information'-feature. The searchwindow is utilized. The first objects feature data is used to compute a new vector which consists of the difference between consequtive frames. The standard deviation is computed and acts as a threshold. Now one wants to find the frame where the objects came to an almost standstill. This is done by checking each value in the new vector against the standard deviation. When the standstill-point is found, this make up the startpoint for the further anlyses towards the assumed point of the object drop. An object about to make a drop, first comes to an almost standstill, then gradually increase it's displacement towards the point of the drop. For each value in the new vector from the standstill-point the difference between consequtive frames is calculated. If none of the difference-values between the frames are equal or less the zero, meaning the object translation increases for each

frame, the final comparison returns the value of true.

As a final check the system gather the result of the four analysis-parts to see if the results are similar. If so, the object dropping userdefined criteria's have been fulfilled, and the system will assume that the event has occured.

5 Experimental results

5.1 General

Throughout the running of the video used in the experimental stage, there are several features extracted and logged from each blob in each processed frame. The features extracted are listed in table 2.

All the extracted data has been illustrated in a 2-D plot, and in section 5.4 each of these plots will be analyzed in depth, as to describe the correlation between the data, and what actually takes place in the video sequence. The plots are analysed to see if one can describe when the event of object dropping takes place, and if there is a relationship between all the extracted data as to where the actual object dropping occurs.

5.2 Data filtering

The extracted data goes through a filtering process to attenuate for the noise-elements due to the segmentation of the blobs from the background.

For the filter process the medfilt1-function was employed. That is a built-in Matlab-function which implements one-dimensional median filtering. A nonlinear technique that applies a sliding windows to a sequence of values. The median filter replaces the center value in the window with the median value of all points within the window.

The window size n, used for the filtering process, was set to 7. There was several filtering trials with different odd-numbered values ranging from 5 to 13. The value 7 was finally chosen as it yielded the best results. When using values of $n = 5$ there was still big variations in the plots, and for $n = 13$, the plots were too smooth as to erase the crucial points in the video-sequence where the interesting part took place.

For computing the median, the medfilt1-function was given two inputs, a vector x and a window-size n.

$$y = medfilt1(x, n); \tag{5.1}$$

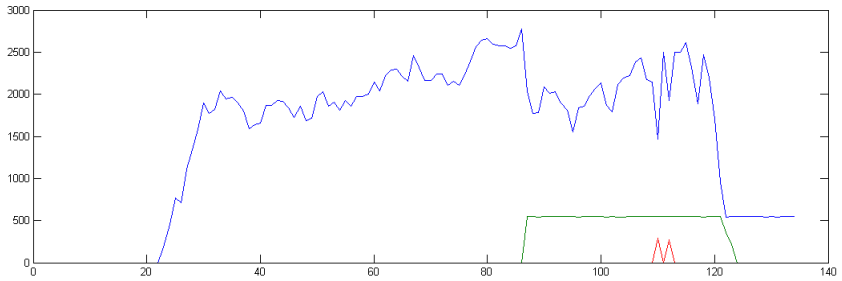
where the output y has the same length as x, n is an odd number, the median for $y(k)$ is computed like this:

$$y(k) = medfilt(x(k - (n - 1)/2 : k + (n - 1)/2)); \tag{5.2}$$

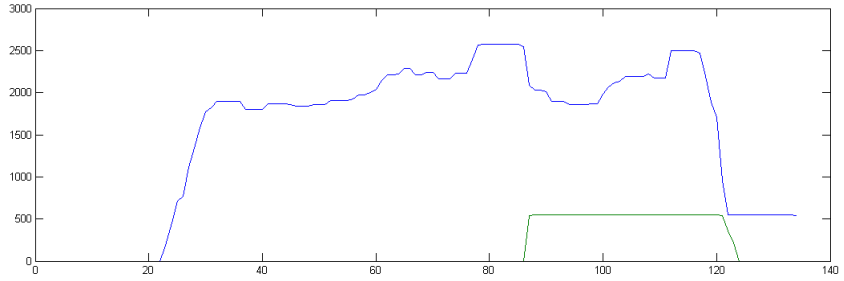
Since the median filter replaces the center value in the window with the median value of all points within the current window, the windows size was set as an odd number.

5.2.1 Results of filtering

In figure 15 there are two plots of the same feature, the above is before filtering, and the plot below is after applying the median filter. Here the filtering is applied on the extracted data to smooth out the variation between the consecutive frames. This will enhance the quality of the data to be able to interpret it, and determine whether a object dropping has occurred.



(a) Before filtering



(b) After filtering

Figure 15: (a) Plot of the extracted data of the area-feature property, and (b) is the same data after the filtering process with window size n equal to 7.

Another functional way of utilizing the filtering process is presented in figure 16, where the peaks in the first object are noise elements due to bad background segmentation. Because these peaks only last for one frame at the time, it is assumed that these peaks are due to noise. The filter is applied, and the peaks in discarded.

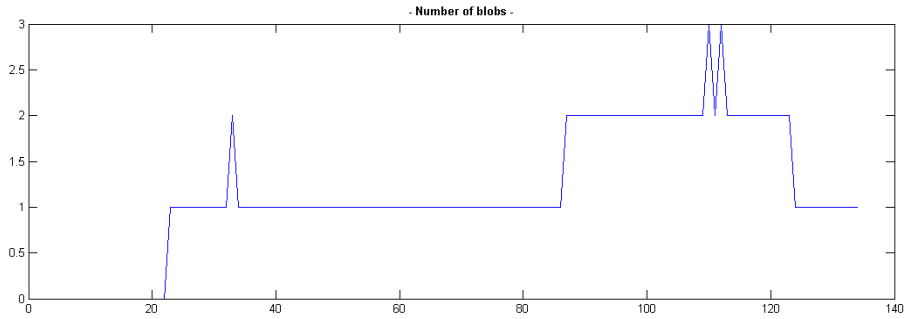
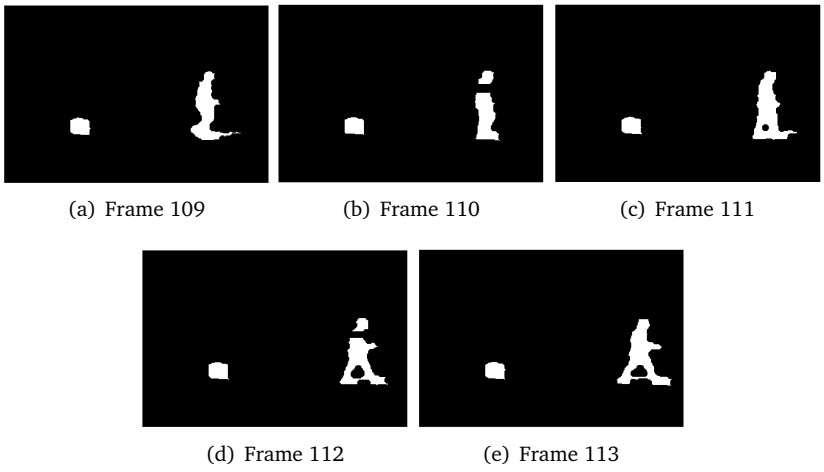


Figure 16: Feature date with noise elements due to bad background segmentation.

With the binary images displayed in figure 17 it is proved that the peaks in figure 16 are noise elements. As one can see at frames 110 and 112 the head of the object is temporary separated from the rest of the body.

5.3 Blob labeling

A problem which occurred during the feature extraction was the labeling of the blobs in the scene. In order to store the extracted data in a logical manner, the labeling of each



(a) Frame 109 (b) Frame 110 (c) Frame 111

(d) Frame 112 (e) Frame 113

Figure 17: A sequence of binary images displaying a result of bad background segmentation.

blob should be static. Meaning that the blob should have the same label the whole time it is present in the scene.

In figure 18 there is a distinct difference in the plots of both (a) and (b), although they both come from the same data. The content of this video source used here consist of one object entering the scene at the left border, and exiting the scene at the right border. During the movement from left to right, it stops approximately in the middle of the scene and makes the object drop. From this point the scene consists of two object, hence the appearing of the second object from frame 87 and forward.

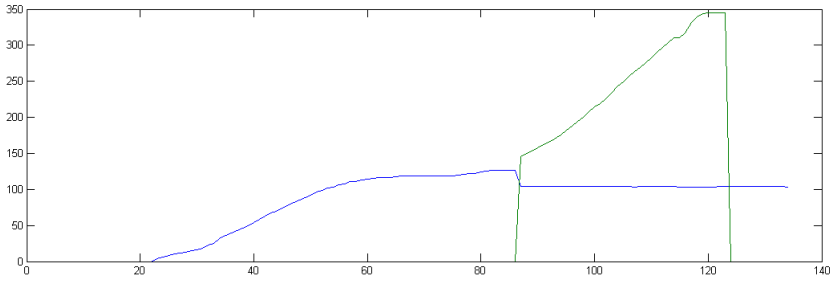
From the point of the object dropping, and since the Matlab labeling-function labels object in a left-to-right manner, the newest object (the left-most object) will be given the label which originally belonged to the first object (blue line). The result of this is shown in figure 18a, where at frame 87 the blue line and the green line crosses each other. This is the point where the labels of the objects switch.

In figure 18b the labeling problem has been solved as explained in section 4.3.2. Here, both the green and blue lines are continuous before, during and after the object drop has occurred.

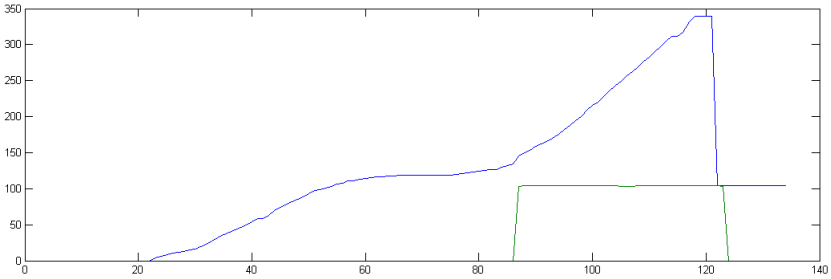
Since the labeling problem was solved by features sorted based on the 'Area'-feature, a problem occurred when the first object left the scene. An example of this is illustrated in figure 18b. When the dropped object was the only object in the scene, it became automatically the biggest blob (blue line). This part of the video sequence is not the interesting part of what we are analysing. The data from the point and out is disregarded in the analysis part. Although it is a problem that should be solved in future work.

5.4 Plot-analysis of extracted features

In the following subsections each of the features involved in the object dropping determination process are described. The extracted data has been plotted in graphs as to be able to visually confirm what relation the data has to the event in question.



(a) Before labeling



(b) After labeling

Figure 18: Plots of the x-centroid feature before (a), and after (b) sorting the labeling problem for the video data.

5.4.1 Area

The Area-feature contains a positive integer number, which contains the actual number of pixels for each blob, hence the name area. In figure 19 one see the plot of the extracted data.

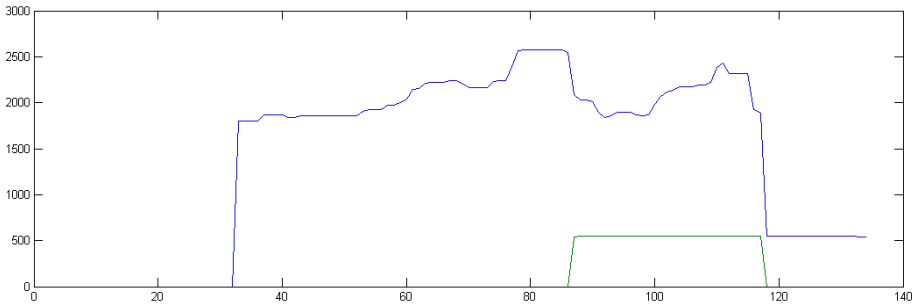
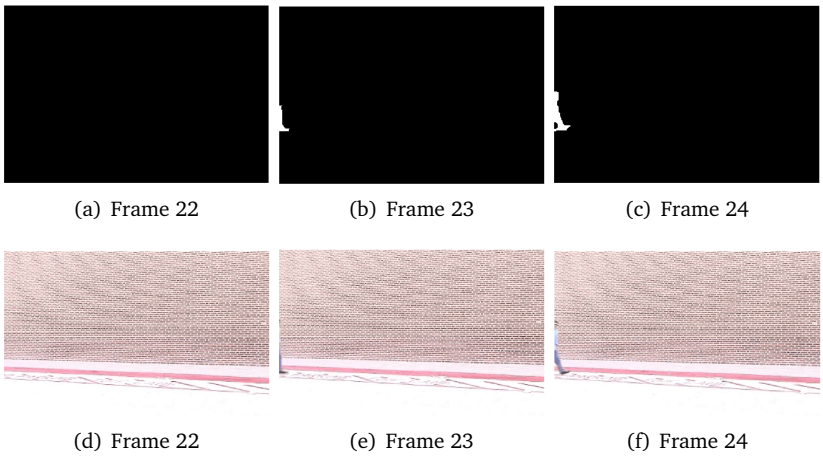


Figure 19: The data values from the area-feature for all blobs in the video sequence.

The x-axis represent the timeline of the video (in frames), while the y-axis represent number of pixels.

If one examines the data plotted in figure 19, it is shown that there is a sudden increase of value for the blue at frame 23. This is the point where object first enters the scene, as displayed in figure 20, there are three frames, ranging from 22 to 24. The sudden shift in value can be explained like this; in frame 22 there are no blobs present in the scene, therefore the area value is zero. In the two next frames, one can see that there is a blob entering the scene, and it increases in size from frame 23 to 24, and therefore

we have this sudden increase in the blue value.

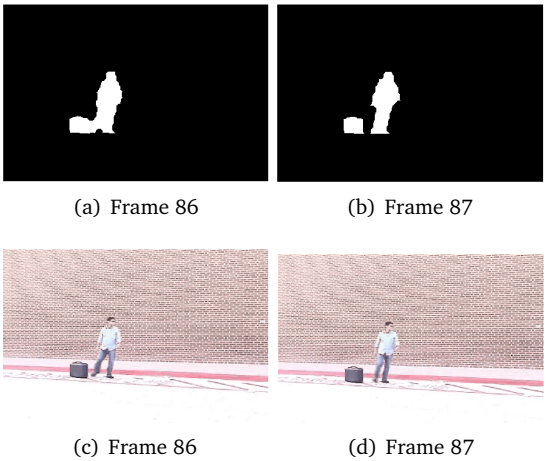


(a) Frame 22 (b) Frame 23 (c) Frame 24

(d) Frame 22 (e) Frame 23 (f) Frame 24

Figure 20: (a-c) are binary images, and (d-f) are rgb images of frames 22-24.

The next point of interest to be analyzed in this area-feature plot is between frames 86 and 87. At this point there is a sudden drop in the area for the first object, and in addition a second object appears in the scene. This could be explained in at least two ways. Firstly, the first object (blue line) might just have been sitting down, and therefore decreasing the value of the area, and the second object (green line) could have entered the scene from either sides.



(a) Frame 86 (b) Frame 87

(c) Frame 86 (d) Frame 87

Figure 21: (a and b) are binary images, and (c and d) are rgb images of frames 86 and 87.

Secondly, as shown in figure 21, we see that there is a blob-split happening between the frames 86 and 87, which creates the sudden decrease in the blue line, and a sudden appearance of the green line.

The figure 21, visually confirms that there is a object dropping taking place in this point in the video sequence, and not that the first object is e.g. sitting down, as to make blob smaller which the plot indicates, and the second object is entering the scene from

one of the either sides. Without the binary images in figure 21 this feature alone is not enough to determine whether an object dropping has taken place. Therefore, further analysis of the blobs feature data have to be completed.

5.4.2 Number of blobs

The "Numel"-feature keeps track of the total number of blobs in the current frame being processed. The plot for this feature can be viewed in figure 22.

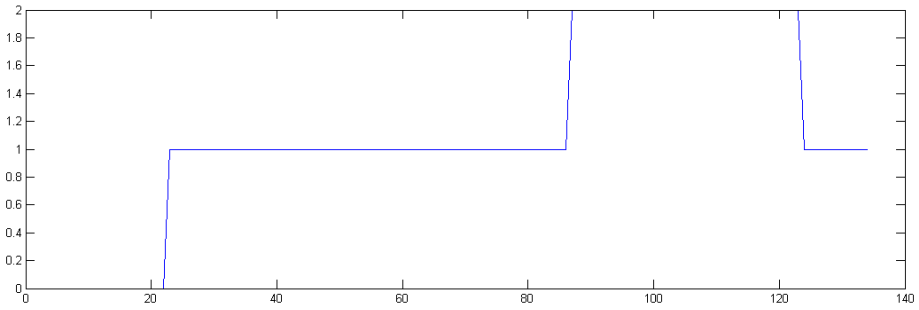


Figure 22: The exact number of blobs in the each processed frame.

From frame 23 there is a object entering the scene, as seen in figure 20. Which account for the instant increase in value in the plot. Further, there is another sudden increase in value at frame 87, this can be seen in figure 21. Here, the second object makes an appearance.

5.4.3 Width-height-ratio

The "Width-height-ratio"-feature contains values for the ratio between the width and height of the bounding box. The figure 23 displays the plot for this feature.

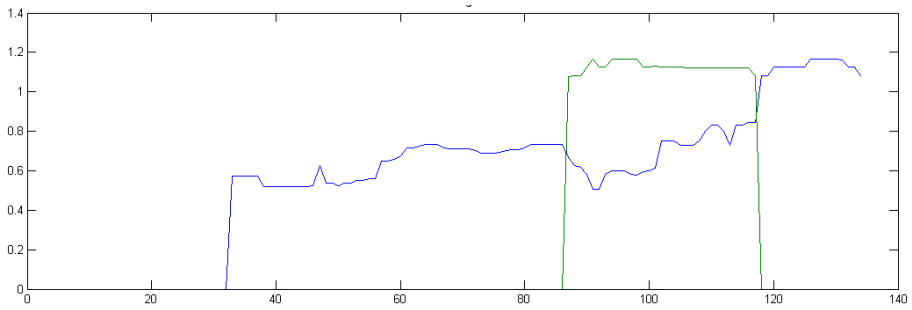


Figure 23: The ratio between the bounding box width and height.

This feature represents the ratio between the width and height of the bounding box. When the object is walking across the scene, one arm is moving, and one arm is still as the object might be carrying a bag. Then the blob of the object would fit into a rectangle, in which the height would be bigger then the width, returning a value which is below one.

From about frame 87 the green line has a constant ratio. This represents the dropped object in the video, and with a constant ratio it can be concluded that the object does not change in terms of shape.

5.4.4 Center of mass (x-axis)

The "Center of mass" x-axis feature contains only the x-coordinate of the center of mass for each blobs. This can be seen in a plot in figure 24.

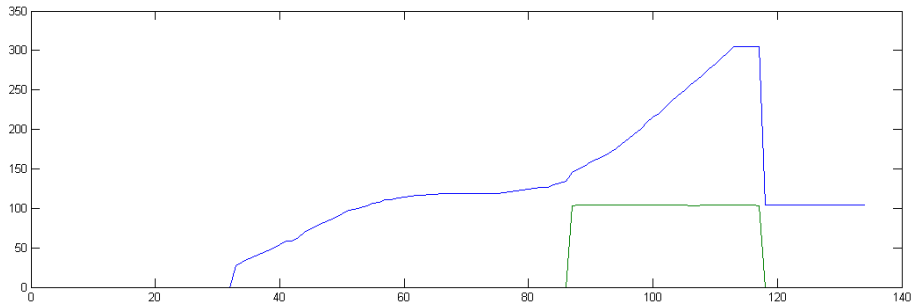


Figure 24: Center of mass x-coordinates for each blob.

This feature keeps track of all the centroid points for each blobs only along the x-axis. In this video there is one object entering the scene from the left side and moving across the scene toward the right. This means that the x-coordinate for this blob starts at the value zero and increases in an almost linear shape throughout the video, as seen in figure 24. Vice versa if the plot started with a high x-value and decreased approximately linear throughout the duration of the video, would mean that the object entered the scene from the right and moving across toward the left.

This plot reveals a lot of information about where the object enters the scene, in which direction it moves, and whether it moves with constant speed or if there is no movement at all.

There are points during the video sequence where the objects motion slows down. This accounts for the first objects (blue line) approaching a horizontal run from frames 60 to 85, and the steep curve of the green line from frames 85 to 120, respectively.

5.4.5 Center of mass (y-axis)

This feature contains only the y-coordinate of the center of mass for each blobs, which can be seen in a plot in figure 25.

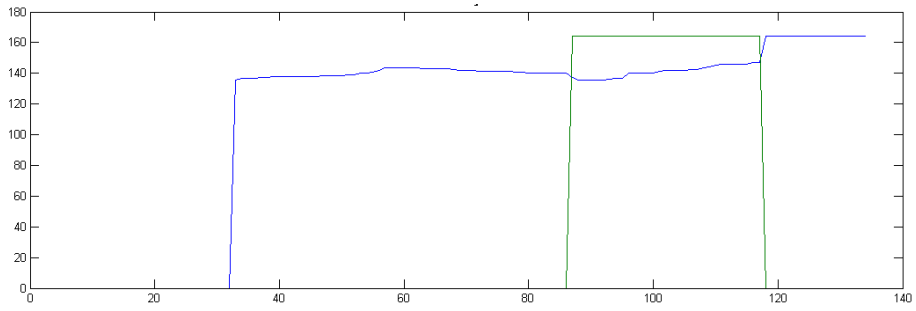


Figure 25: Center of mass y-coordinates for each blob.

As for the "Center of mass" x-axis feature, the same applies for this feature but only in the vertical direction. If the object enters the scene from above and moves downwards and exits the scene in the bottom there would be a plot similar to figure 24, and vice

versa if the object entered at the bottom and moved upwards.

The plot in figure 25 gives an approximately constant horizontal line, meaning the y-coordinates do not vary that much in value between each frame. From this we can conclude that the object is moving horizontally, but we can not determine in which direction it is moving.

Based on both the x- and y-coordinates we can determine the exact movement of the blobs across the scene.

5.4.6 Directional information (x-axis)

The "Directional displacement"-feature extracts values of the translation of the center of mass point in the horizontal direction for each blob between two consecutive frames, which is illustrated in figure 26.

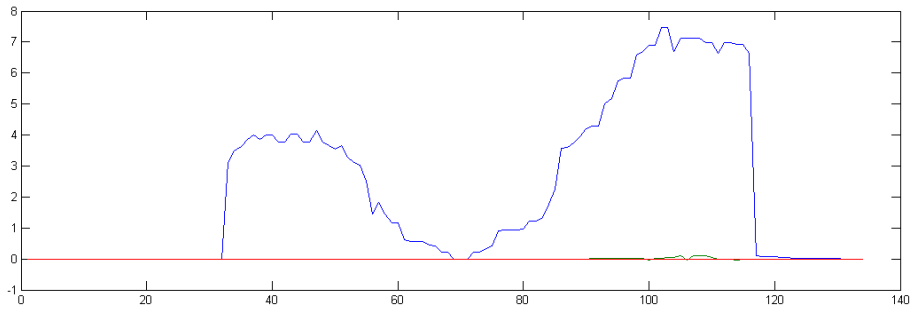


Figure 26: Translation of blobs x-centroids between two consecutive frames.

The values plotted in figure 26 is the translation (distance in pixels) of the center of mass for each blob between consecutive frames.

From frame 0 until frame 86, there is an object moving across the scene, and it is moving in an approximately horizontal way. Therefore, there is not much translation in the x-coordinates between two frames.

The sudden peak at frame 87 belongs to the dropped object. In the previous frame the x-coordinate for the object was zero, and in the following frame the value jumped to approximately 150. Which means there was a huge translation between frames 86 and 87. Further on, from frame 88 and throughout there is again approximately no motion between frames, giving an almost straight line again.

Near the end of the video sequence there is a sudden drop in the x-value. This is the point where the subject is exiting the scene. At this point the value drops approximately 300. To be more precise, the value should be 320, which is the same as the current video frame width. Right before the subject leaves the scene at the rigth side, the x-coordinate is at it's highest. After the subject has exited the scene it's x-value drops instantly to zero. This results in a negative translation, which equals the width of the video frame, and gives the sudden fall in value.

5.4.7 Directional information (y-axis)

The directional displacement features extracts values of the translation of the center of mass point for each blob between two consecutive frames, but in the vertical direction. This is illustrated in figure 27.

Similar to figure 26, this figure 27 is the translation of the center of mass for each

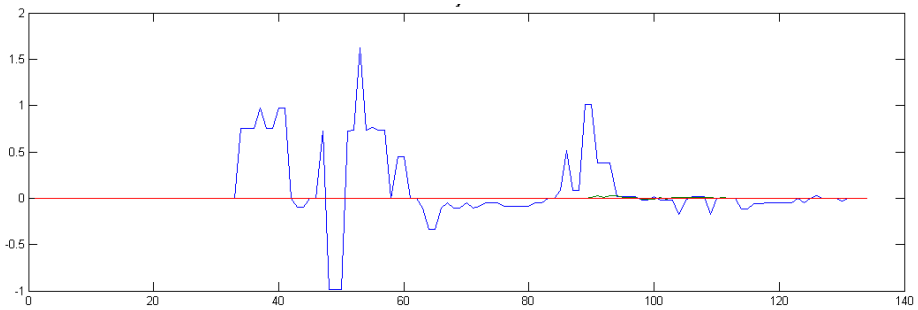


Figure 27: Translation of blobs y-centroids between two consecutive frames.

blob between consecutive frames in the vertical direction.

This plot shows four distinctive peaks, one at frame 23, two at frame 87 and one at frame 124. All of which can be explained like the peaks in figure 26. The first peak at frame 23 comes from a subject enters the scene. For the first twenty-two frames the centroid for the subject had the value zero, which gave no translation. At frame 23 the objects y-coordinate changed to about 150, and this resulted in a huge translation between the objects centroids in frames 22 and 23.

Next, at frame 87 there are two peak, with unequal height. The larger one accounts for the object, after the blob-split. This objects y-coordinate has been, until this point, zero of value. The sudden appearance of a new object, as the peak at frame 23, results in a huge increase in the y-coordinate, which gives this peak.

Both the directional information of x- and y-axis may have different grades of importance. The usefulness of the feature data depends on the video source. If the object is moving horizontal across a scene the x-axis-feature would contain much more related data then the y-axis-feature, and vice versa if the object is moving vertically across the scene. Both features would be equally useful if the object was to move diagonally.

5.5 Experimental results summary

The features separately do not say much about if an object have been dropped or not. If we examine the extracted features together, there are certain conclusion that can be drawn.

The feature keeping track of the total number of blobs in the current frame, reveals that a second object entered the scene at frame 87. Both the features with the centroids for the x- and y-axis show that both the coordinate of the first blob, and the coordinate of the new second blob are not far apart. From this we conclude that the new object did not walk in from any of the sides, but rather made sudden appearance approximately in the middle of the scene, and not very far apart from the position of the first blob. This can also be confirmed by the motion-features, which reveals a high sudden translation, and similar increase in value equal to the centroid-value at the same frame.

If one looks at the area-feature, one can see that at frame 86 the blob-area is approximately 2500 pixels. At the next frame there are now two blobs, with areas of approximately 2000 and 500 pixels. One can conclude from this feature only that the two new blobs with both sizes equal to the first blob being a result of a blob-split, but one can not be a hundred percent certain of this since it may be due to bad segmentation also.

By jointly analyzing all of these features combined together one can conclude that there have been an object dropping taking place.

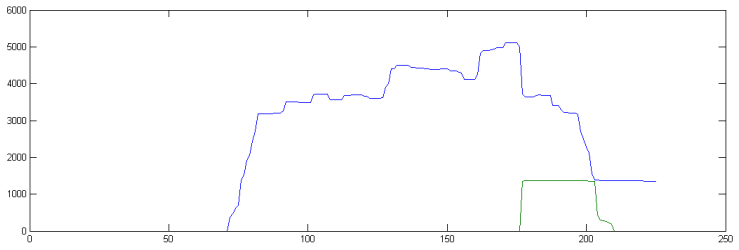
6 Evaluation of experimental results

In the experimental and developing stage in chapter 5 the system was repeatedly tested with them same video source. In this chapter it is made a detailed analysis of the extracted features from three videos not used in the programming stage.

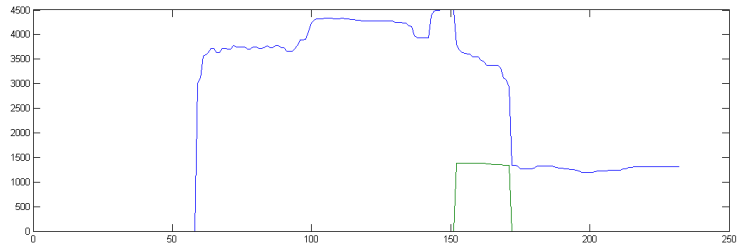
6.1 Feature evaluation

6.1.1 Area

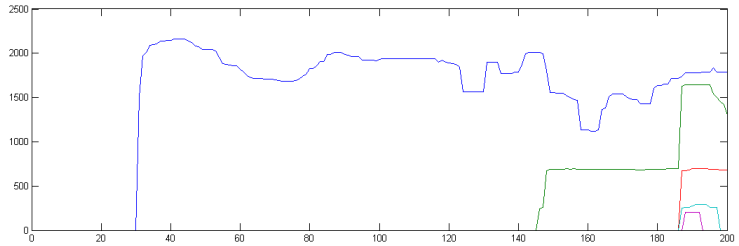
In figure 28 there are three plots where each plot is the same feature extracted from three different sources. Every line in each plot represent objects in the scenes, and the values correspond to the actual size of each object (the number of white connected pixels in a binary image).



(a) Video 1



(b) Video 2



(c) Video 3

Figure 28: The results of the extracted Area-features from the different video sources.

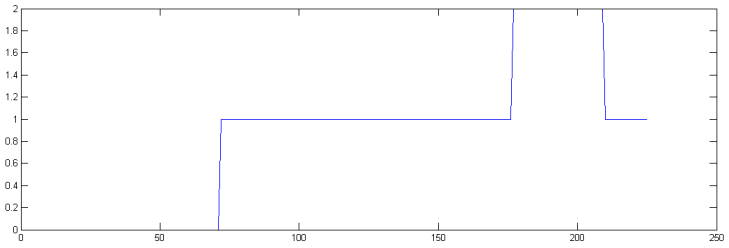
The interesting parts in these plots are the time when the second object enters the scene. This is where the green line makes it's first appearance, and in all three videos

there are significant decrease in the size of the blue line at the same frame. In section 5.4.1 a conclusion of what this means where drawn, and similar conclusions can be drawn from these plots as well.

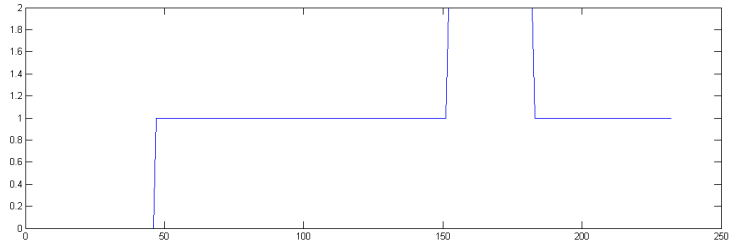
At the frames where the second objects appear all three plots have significant decrease in size. Each fall itself is not very conclusive to what has taken place, but when the second object have approximately the same size as the decrease of the first object, certain conclusions can be drawn. It can be seen as the two objects share a connection, and the connection is that both objects used to be the same object.

6.1.2 Number of blobs

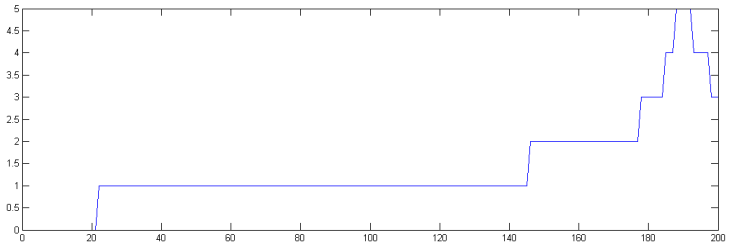
The "Numel"-feature in figure 29 keep track of the total number of blobs in each frame. The whole feature analysis is based on this feature. Whenever an the feature increases by 1, the system will perform the analysis, which yields a result equal true or false regarding an object dropping has taken place.



(a) Video 1



(b) Video 2



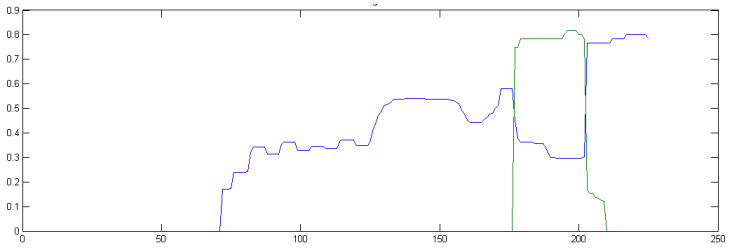
(c) Video 3

Figure 29: The results of the extracted Numel-features from the different video sources.

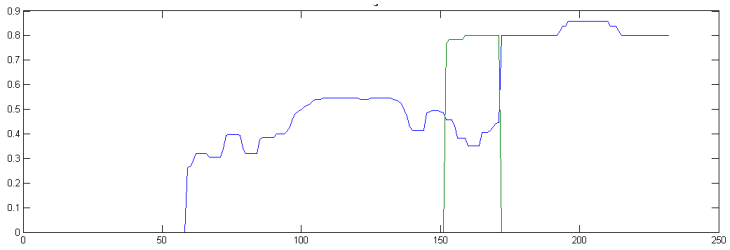
False changes in this feature may occur. By false changes it is referred to blob-splits due to bad background segmentation. These non-interesting splits of objects do not typically have a long duration, and are treated as noise-elements. The noise-elements gets filtered. Blob splits that are not filtered will trigger the systems analysis-stage.

6.1.3 Width-height-ratio

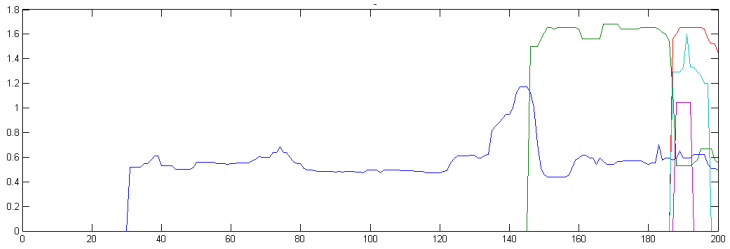
This feature in figure 30 represents the ratio between the width and height of the bounding box surrounding an object. It is used to describe the posture of the objects. It is calculated by $\texttt{ratio} = \frac{\texttt{width}}{\texttt{height}}$ yielding a number higher than zero, and below or above 1. A value below 1 means that the height of the bounding box is bigger then the width, and vice versa if the ratio is above 1.



(a) Video 1



(b) Video 2



(c) Video 3

Figure 30: The results of the extracted Ratio-features from the different video sources.

When a subject is walking across a scene, one arm is moving, and one arm almost still if the subject might be carrying a bag. The blob of the object would fit into a rectangle, in which the height would be longer then the width, returning a value which is below 1. The ratio would still have small variations due to arm and leg movement, but no significant variations.

Right before subjects are about to make a object drop they tends to bend the upper part of their body, and creating a bending posture similar to figure 4. This result in significantly change in ratio of the bounding box. This feature is most interesting before the actual blob-split. At the time of the blob-split the subject has already started to walk away from the dropped object, and the ratio is approaching what is was before the bending began.

The ratio of the second objects, the green lines, have an approximately constant value

in the plots. This accounts for a non-moving objects, hence a suitcase being dropped.

Where the ratio begin to change significantly and where it returns back to "normal" may vary a bit. It depends on how fast the object makes the drop, if the object stop completely or make drop while in motion, or if the object stay near the object after the drop or move away immediately after. What is known through the analysis of these result are that the ratio changes a certain time before and is back to normal almost right after the blob-split. Therefore this feature must be analysed over time, hence the paragraph on search window in section 4.3.4.

6.1.4 Center of mass

The "Center of mass"-feature stores information about the position of all objects at each frame, x-coordinates and y-coordinates in separate variables in figures 31 and 32 respectively.

A small example on how to read one plot; by examining figure 31a the blue line makes a sudden appearance at approximately frame 70. From this frame and forward the value slowly decrease. From this one can say that the object entered the scene at the right border, and moved across the scene to the left. A decrease in the x-value correspond to an object moving from right to left since the coordinate (0,0) is in the left-bottom corner.

These plots reveals a lot information about where the objects entered the scene, in which direction it moved, whether it moved with a constant speed from frame to frame, or if there were no movement at all.

The primary tasks of this feature is to show where the object are located, and most important of all to show the distance between two objects. The system want to determine whether an object dropping has occurred. Among other features, the distance from one object to the second, newly appeared object, is a crucial piece of evidence. If the objects appeared at each end of the scene, the distance would be to big to be considered that the two objects used to be the same.

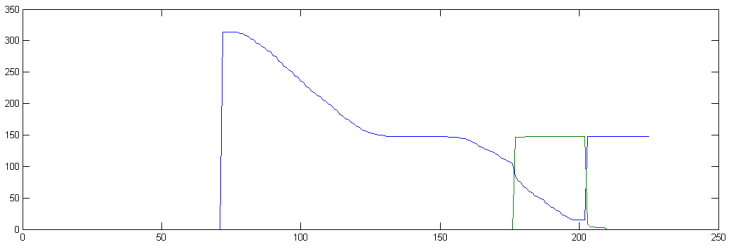
The "Minor axis"-feature is used as threshold to determine whether the distance between the two objects are to big or not. In this system, the Minor-axis is extracted from the first object. An ellipse is fitted around the object, and when the object is standing the Minor axis can be considered as the width of the object. If two object have a distance smaller then two times the Minor axis, the distance is inside the preset threshold, and the system accepts the object dropping criteria for this feature.

6.1.5 Directional information

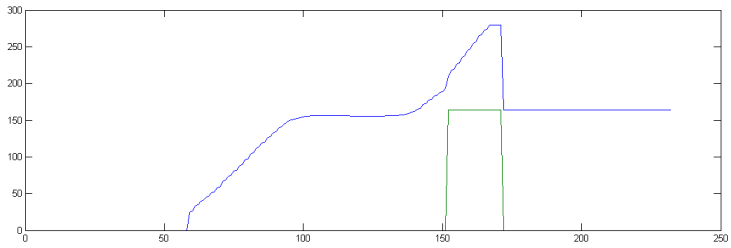
The "Directional information"-feature says something about the displacement of objects between consecutive frames. Similar to posture change before object dropping in the "Ratio"-feature, there are also motion changes in the Directional information-feature regarding object-behavior prior to the object dropping. In figure 33 there is in all three plots, approximately in the middle of each video, a sequence with very little or no motion by the first object.

The decrease in motion is common for all three videos used in this test, and is the part in the videos where the object makes the actual object dropping. After the drop has been made the motion of the object increases again.

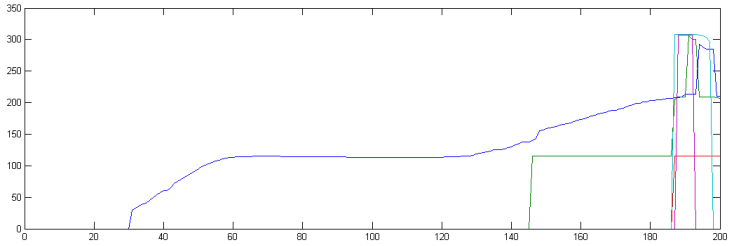
Figure 34 is in this case not very relative. The motions along the y-axis are very constant throughout all three videos since the objects are moving in a horizontal direction only (see last paragraph in section 5.4.7).



(a) Video 1

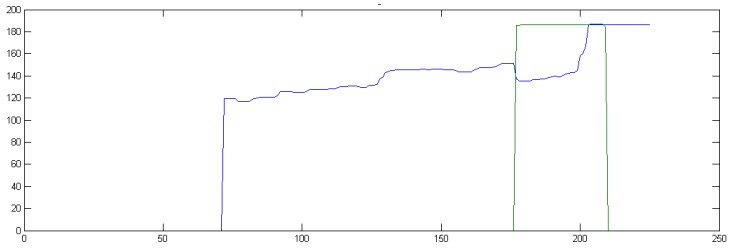


(b) Video 2

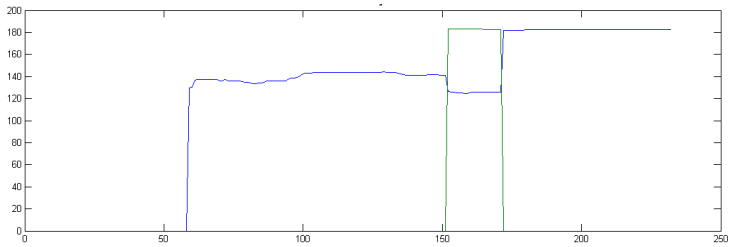


(c) Video 3

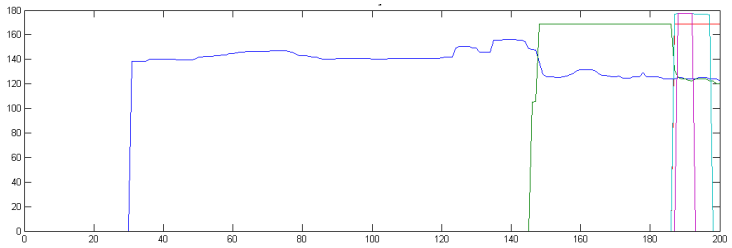
Figure 31: The results of the extracted Center of mass-features (x-axis) from the different video sources.



(a) Video 1

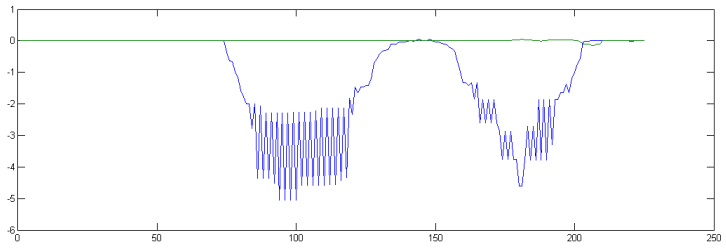


(b) Video 2

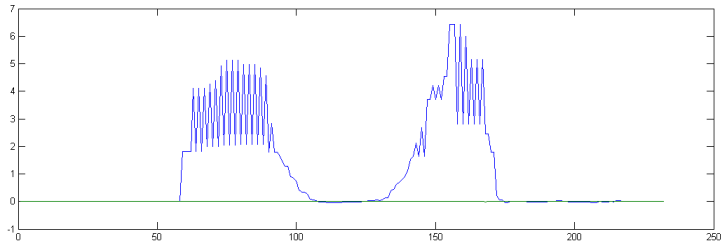


(c) Video 3

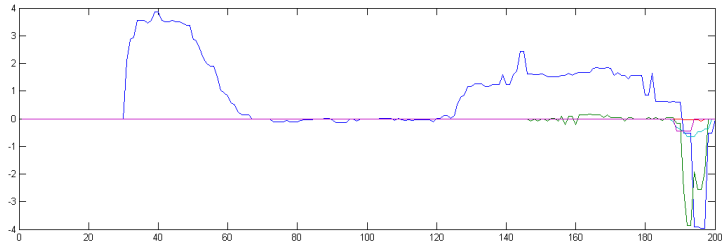
Figure 32: The results of the extracted Center of mass-features (y-axis) from the different video sources.



(a) Video 1

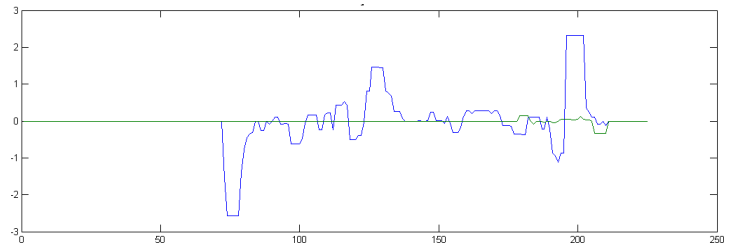


(b) Video 2

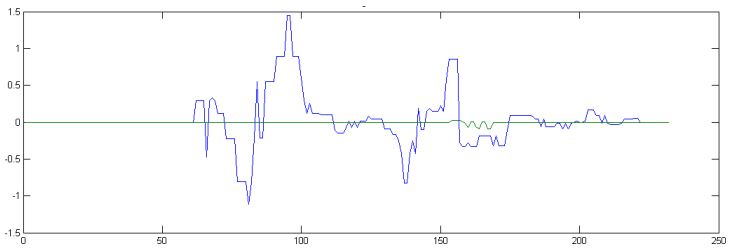


(c) Video 3

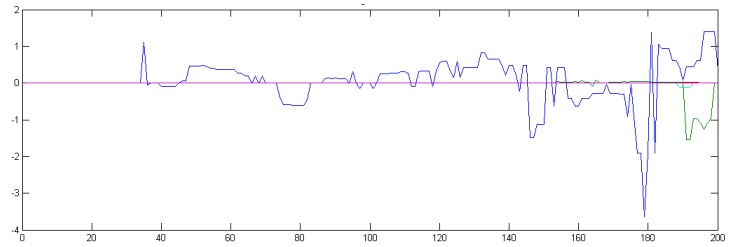
Figure 33: The results of the extracted Directional information-features (x-axis)) from the different video sources.



(a) Video 1



(b) Video 2



(c) Video 3

Figure 34: The results of the extracted Directional information-features (y-axis)) from the different video sources.

6.2 Evaluation overview

Each feature by themselves is not enough to determine whether the abnormal event of object dropping has occurred. All features in a combination has to be considered to make this determination.

The system analysis is triggered based on the "Numel"-feature. When the system is triggered it performs an analysis on all the extracted data. These data are compared against a set of user defined criteria's (see table 3). These criteria's act as threshold values.

When the necessary computation is made the system yields the result in form of true or false, if each of the four feature-comparison parts concur on the frame of the object dropping. True if a object dropping has taken place, and false if not. The system also returns where in the timeline of the video the event has taken place.

6.3 Experimental results

The output of the feature analysis-stage is displayed in table 5. Four videos where used in the final experiment and test of the outworked system. All of the four video were utilized, one by one, by the system. The output of the system where four numbers per video. The outputnumberbs represents the framenumbers which the system assumed an object dropping has taken place. In the feature analysis-stage there is four tests, which all yields one framenumber. The system concurs to an object dropping if all these framenumber are the same.

For the experiement there were utilized three videos in addition to the one video used during the implementation and testing-stage. The tables second colomn shows the framenumber of the actual object dropping in each of the four videos. The next four colomns list the framenumbers which the feature analysis-algorithm returned. The four colomns are the features of Area, Center of mass, Ratio and the Directional information. The extracted data of these four features are compared against the userdefined criteria's. In each video, and for each feature the exact similar framenumber is returned, which means that the event of object dropping has been recoqnized by the system.

Experimental results					
Video	**Drop**	**Area**	**Center**	**Ratio**	**Dir. info**
Video 1	178	178	178	178	178
Video 2	87	87	87	87	87
Video 3	152	152	152	152	152
Video 4	145	145	145	145	145

Table 5: Eperimental results of the feature comparison-stage of the proposed system.

Outlined in table 5 it is the result of the feature analysis-stage. From this table one can see that the system returned the same framenumbers for all features for all four videos. This means that the system recoqnized the object dropping, based on the userdefined criteria's, at the similar frames four all faetures in all of the four videos.

7 Conclusion

Abnormal events in video surveillance scenarios can be concluded as something that seldom occurs, or something that is previously unseen. There are many different types of events that can be described as abnormal, but the common description is thath these events deviates from most of the action that occur in a scene.

To be able to model abnormal events we had start off by performing a visual subjective analysis on a video training set. We had to be able to get the understanding of what was actually happening in the video sequence before, during and after the event of object dropping. By examing several videos with the same type of event we discovered that it was possible to define a set of criteria's for the event. The set of criteria's was set to act as threshold values saying that an object dropping had occurred if all these predefined values were fulfilled.

We extracted several features from the objects in their entire presence in the scene. A combination of these low-level features did contain sufficient information to make a description on what characterizes the abnormal event of object dropping.

With the ability of predefining the behavior of an object it was a very good platform for developing a system. The system was developed around the extraction and analyses of the low-level features. The extracted features was compared against our predefined criteria's. Each feature compared against each criteria separatly, by the means of thresholding. For each feature the system returned the framenumber for which it assumed that the object dropping have taken place. If the every feature comparison returned the same framenumber the feature analyse-stage concurred in the sense that they agreed on where the object dropping took place.

By making a general description of what the event in question contains, before, during and after the actual point of the object dropping we were able to derive a set of predefined criteria's. If the feature analysis fulfilled these criteria's we could conclude that an object dropping had occured, and that we were able to determine exactly the point where this event occurred.

8 Further work

There are several stages of the proposed system that have room for improvement in the future.

A highly important step in the whole process is the background subtraction. This part lay the foundation for good foreground extraction of the moving objects. Poorly performed background segmentation will at a later stage yield insufficient feature data about the objects. Developing new methods and algorithms for performing background subtraction is a highly relative subject to continue with in the future.

The tracking of the objects should be improved by enhancing the labeling process. At this point the system is sufficient for handling the event of one moving object. To enhance the system the labeling process must be improved in order to track several moving objects in the same scene at the same time. In addition to be able to handle several objects in the scene, the labeling should ideally give a static label to each of the objects as long as they are presence in the scene.

Discussed in chapter 4, the analysis of the data should include several frame. The analysis stage, regarding the search window, needs improvements in order to sufficiently handle all features. Some of the analysis is at this point only conducted at one frame, and not for several frames as it optimally should be done. Although some data is sufficiently analysed by when conducted at one frame, the searchwindow should apply for all features.

A clear extension to the system would be to include handling for other types of abnormal events, beside object dropping. This would include developing new sets of criteria's for each new event, and implement them into the already existing system.

Bibliography

[1] McCahill, M. & Norris, C. *CCTV In London*. University of Hull, Centre for Criminology and Criminal Justice, March 2002.

[2] A. Mecocci, M. P. & Fumarola, A. Automatic detection of anomalous behavioural events for advanced real-time video surveillance. *IEEE International Symposium on Computational Intelligence for Measurement Systems and Applications, 29-31 July 2003*, 187–192.

[3] Johnson, N. & Hogg, D. Learning the distribution of object trajectories for event recognition. *Proceedings of the 6th British conference on Machine vision (BMVC'95), 1995, Birmingham, United Kingdom*, 583–592.

[4] Da-shan Gao, J. Z. & ping Xin, L. A novel algorithm of adaptive background estimation. *Proceedings of the International Conference on Image Processing, 7th-10th October 2001, Thessaloniki, Greece*, 395–398 vol.2.

[5] A. Monnet, A. Mittal, N. P. & Visvanathan, R. Background modeling and subtraction of dynamic scenes. *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003), 13th-16th October 2003, Nice, France*, 1305–1312 vol.2.

[6] S. Soatto, G. D. & Wu, Y. N. Dynamic textures. *Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV 2001), 9th-12th July 2001, Vancouver, Canada*, 439–446 vol.2.

[7] Christof Ridder, O. M. & Kirchner, H. Adaptive background estimation and foreground detection using kalman-filtering. *Proceedings of International Conference on Recent Advances in Mechatronics, ICRAM'95, 14th-16th August 1995, Instanbul, Turkey*, 193–199 vol.1.

[8] C.R. Wren, A. Azarbayejani, T. D. & Pentland, A. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence, July of 1997, Los Alamitos, CA, USA*, 780–785 vol.19.

[9] Jia, L. & Liu, Y. A novel thresholding approach to background subtraction. *IEEE Workshop on Applications of Computer Vision (WACV), Colorado, USA, 2008*, 1–6.

[10] S.S Polmottawegedara, R. M. & Davari, A. Tracking moving targets. *Proceeding of the Thirty-Eighth Southeastern Symposium on System Theory (SSST), Cookeville, Tennessee, USA, 2006*, 80–84.

[11] A. Elgammal, R. Duraiswami, D. H. & Davis, L. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. *Proceedings of the IEEE July 2002*, 1151–1163 Vol. 90.

[12] Friedman, N. & Russell, S. Image segmentation in video sequences, a probabilistic approach. *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI 97), 1st-3rd August 1997, San Francisco, CA, USA*, 175–181.

[13] D. Comaniciu, V. R. & Meer, P. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence, May 2003*, (5), 564–577 vol.25.

[14] M.S. Arulampalam, S. Maskell, N. G. & Clapp, T. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing, February 2002*, (2), 174–188 vol.50.

[15] S. Calderara, R. C. & Prati, A. A dynamic programming technique for classifying trajectories. *14th International Conference on Image Analysis and Processing (ICIAP 2007), 10th-14th Sept. 2007, Modena, Italy*, 137–142.

[16] Mecocci, A. & Pannozzo, M. A completely autonomous system that learns anomalous movements in advanced videosurveillance applications. *IEEE International Conference on Image Processing (ICIP 2005), 11th-14th Sept. 2005, Genova, Italy*, 586–589.

[17] I. Haritaoglu, D. H. & Davis, L. W4: real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000*, 22(8), 809–830.

[18] Yue Zhou, S. Y. & Huang, T. Detecting anomaly in videos from trajectory similarity analysis. *IEEE International Conference on Multimedia and Expo (ICME 2007), 2th-5th July 2007, Beijing, China*, 1087–1090.

[19] Michal Irani, B. R. & Peleg, S. Detecting and tracking multiple moving objects using temporal integration. *Proceedings of the Second European Conference on Computer Vision (ECCV '92), 19th-22th May 1992, Santa Margherita Ligure, Italy*, 282–287.

[20] W.E.L. Grimson, C. Stauffer, R. R. & Lee, L. Using adaptive tracking to classify and monitor activities in a site. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1998), 23th-25th June 1998, Santa Barbara, CA, USA*, 22–29.

[21] Choi, E. & Lee, C. Optimizing feature extraction for multiclass problems. *IEEE Transactions on Geoscience and Remote Sensing, 2001*, 39(3), 521–528.

[22] H. Peng, F. L. & Ding, C. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Los Alamitos, CA, USA, 2005*, 27(8), 1226–1238.

[23] Shi-Fei Ding, Zhong-Zhi Shi, V.-C. W. & Li, S.-S. A novel feature extraction algorithm. *Proceedings of 2005 International Conference on Machine Learning and Cybernetics, 18th-21th August 2005, Guangzhou, China*, 3, 1762–1767.

[24] Nixon, M. S. & Aguado, A. S. Feature extraction and image processing. *Academic Press, Linacre House, Oxford, United Kingdom, 2002*.

[25] Li, Z. & Tan, Y.-P. Event detection using multimodal feature analysis. *IEEE International Symposium on Circuits and Systems (ISCAS), Kobe, Japan, 2005*, 4, 3845–3848.

[26] Bernard Boulay, Francois Bremond, M. T. & Antipolis, I. S. Human posture recognition in video sequence, 14th november 2003.

[27] Fuentes, L. & Velastin, S. Advanced surveillance: From tracking to event detection. *IEEE Latin America Transactions (Revista IEEE America Latina), September 2004*, 1–1 Vol.2.

[28] Remagnino, P. & Jones, G. A. Classifying surveillance events from attributes and behaviour. *Proceedings of the British Machine Vision Conference BMVC, 10-13 September 2001, Manchester, United Kingdom*, Section 8: Modelling Behaviour.

[29] M.H. Liao, Duan-Yu Chen, C.-W. S. & Tyan, H.-R. Real-time event detection and its application to surveillance systems. *Proceedings of the 2006 IEEE International Symposium onCircuits and Systems (ISCAS '06), 21th-24th May 2006, Island of Kos, Greece*, 4 pp.–.

[30] Fan Jiang, Y. W. & Katsaggelos, A. K. Abnormal event detection from surveillance video by dynamic hierarchical clustering. *IEEE International Conference on Image Processing (ICIP 2007), 16th Sept. - 19th Oct. 2007, San Antonio, Texas*, 145–148 vol.5.

[31] Latecki, L. & de Wildt, D. Automatic recognition of unpredictable events in videos. *Proceedings of the 16th International Conference on Pattern Recognition (ICPR 2002), 11th-15th August 2002, Quebec, Canada*, 889–892 vol.2.

[32] G. Medioni, I. Cohen, F. B.-S. H. & Nevatia, R. Event detection and analysis from video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence, August 2001*, "873–889 Vol.23.

[33] S. Ferrando, G. G. & Regazzoni, C. Classification of unattended and stolen objects in video-surveillance system. *IEEE International Conference on Video and Signal Based Surveillance (AVSS '06), November 2006, Sydney, Australia*, 21–21.

[34] Xiang, T. & Gong, S. Discovering bayesian causality among visual events in a complex outdoor scene. *IEEE Conference on Advanced Video and Signal Based Surveillance, Miami, USA, 2003*, 177–182.

[35] M. Bhargava, Chia-Chih Chen, M. R. & Aggarwal, J. Detection of abandoned objects in crowded environments. *IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS 2007), 5th-7th Sept. 2007, London, England*, 271–276.

[36] Hua Zhong, J. S. & Visontai, M. Detecting unusual activity in video. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), 27th June-2nd July 2004, Washington D.C., USA*, 819–826 Vol.2.

[37] Leedy, P. & Ormrod, J. Practical research, planning and design. *Pearson Education Inc, 2005*.

A System source code

A.1 Tracking

Listing from source file `Tracking.m`.

```
%% Tracking — main file

function [tValues] = Tracking(video)

clc; % Clears the command window
tic; % Timer start

vidObj = mmreader(video);
nFrames = vidObj.NumberOfFrames;
% vidObj = mmreader('drop_4.avi');

% Mean pixel intensity background image, and a standard deviation image
% (both based on the 'n' first frames of the input video)
n = 25;
[meanBackground, stdDev] = Background(vidObj,n);
toc;
disp '— background complete —';

% Confidence level is used to determine which pixels in a binary image to
% get assigned values of 0 and 1. The standard deviation is multiplied with
% this value to set the level of confidence.
confidenceLevel = 1.8;

% Automatically remove blobs (regard as noise) smaller then 'z' pixels
z = 150;

% Creating variables in memory (preallocated for speed)
blobArea(1,1) = 0;
blobWidthHeightRatio(1,1) = 0;
blobCentroidX(1,1) = 0;
blobCentroidY(1,1) = 0;
blobMinorAxisLength(1,1) = 0;

% Search window for feature vector data comparison. The size is based on
% the framerate of the input video data. Because determine the window of a
% fixed length of frames can give much or little information back to the
% system. If the framerate is 10fps, then a preset of 10 frames would give
% much useful data since much action can take during one second. IF the fps
% is 100, then 10 frames is only 1/10 of a second and would probably not
% contain much useful info. The search window is set to 2xfps
sWindow = 2*(vidObj.FrameRate);

% Window size for Median Filtering
mf = 9;

for ii = 2:nFrames
    % Read one frame, and convert to binary
    vidFrameCur = read(vidObj,ii);

    % Crop images to get rid of date/time info in frames from axis camera
    vidFrameCur = imcrop(vidFrameCur,[0 0 320 220]);

    [binFrameCur] = FramePreProcessing...
        (vidFrameCur,meanBackground,stdDev,confidenceLevel,z);
```

```
    clear vidFrameCur;

    % Save binary to disk, for analysis together with plots
    %filename = (['Binary frames/frame_' int2str(ii) '.tif']);
    %imwrite(binFrameCur,filename);

    % Extract blob properties for current and previous frame
    [L,num] = bwlabel(binFrameCur,8);
    blobs = regionprops(L,'all');

    % Number of blobs at each frame
    Numel(ii) = num;

    % Dimensions of current frame
    [h w] = size(binFrameCur);

    % frame
    % Show the video
    %colorBin = label2rgb(binFrameCur,'jet');
    %colorBin = label2rgb(L);
    %figure('Position',[0,0,360,240]);
    imshow(binFrameCur,'Border','tight','InitialMagnification','fit');
    %title('— Processed video —','fontsize',12,'fontweight','b');
    hold on
    for jj = 1:numel(blobs)
        blobArea(jj,ii) = blobs(jj).Area;
        width = blobs(jj).BoundingBox(3);
        height = blobs(jj).BoundingBox(4);
        blobWidthHeightRatio(jj,ii) = width/height;

        blobMinorAxisLength(jj,ii) = blobs(jj).MinorAxisLength;

        % Store blobs centroid positions
        blobCentroidX(jj,ii) = blobs(jj).Centroid(1);
        blobCentroidY(jj,ii) = blobs(jj).Centroid(2);

        if(numel(blobs) == 1)
            rectangle('position',[blobs(1).BoundingBox(1); ...
                ,blobs(1).BoundingBox(2);,blobs(1).BoundingBox(3); ...
                ,blobs(1).BoundingBox(4);],'edgecolor','k','LineWidth',0.5);
        else
            rectangle('position',[blobs(1).BoundingBox(1); ...
                ,blobs(1).BoundingBox(2);,blobs(1).BoundingBox(3); ...
                ,blobs(1).BoundingBox(4);],'edgecolor','r','LineWidth',5);
            rectangle('position',[blobs(2).BoundingBox(1); ...
                ,blobs(2).BoundingBox(2);,blobs(2).BoundingBox(3); ...
                ,blobs(2).BoundingBox(4);],'edgecolor','k','LineWidth',0.5);
        end
        %plot(blobs(jj).Centroid(1), blobs(jj).Centroid(2), 'r*');

    end

    filename = (['drop_2/newBinary2/frame' int2str(ii) '.png']);
    print('-dpng','-r300',filename);


    hold off
    %axis equal; axis off;
    drawnow;


%       aa = ii;
%         if (ii==aa)
%         % Save binary to disk, for analysis together with plots
%         pause;
%         aa = aa+1;
%         end
```

```
end
close all;

% Compute speed and motion from centroid position between consequtive frames
[xmotion ymotion speed] = XYMotion(blobCentroidX,blobCentroidY);

% Sort area-feature descending according to size
[sA,idx]   = sort(blobArea,'descend');
% Use idxorder to sort other features according to area-feature
idxorder = sub2ind(size(blobArea),idx,...
    repmat(1:size(blobArea,2),size(blobArea,1),1));

% Sort all feature-data matrices before plotting
% (sort according to 'idxorder')
sCY = blobCentroidY(idxorder);
sCX = blobCentroidX(idxorder);
%sP = blobBoundryPerimeter(idxorder);
sWHR = blobWidthHeightRatio(idxorder);
sXM = xmotion(idxorder);
sYM = ymotion(idxorder);
%sS = speed(idxorder);
%sMaAx = blobMajorAxisLength(idxorder);
sMiAx = blobMinorAxisLength(idxorder);

% Median filtering
[fA,fCY,fCX,fWHR,fNumel,fXM,fYM,fMiAx] = ...
    Filter(sA,sCY,sCX,sWHR,Numel,sXM,sYM,sMiAx,mf);

toc;
% Plot all extracted features
Plotting(fA,fCY,fCX,fWHR,fNumel,fXM,fYM,fMiAx);
% Plot testing
% Plotting(drop2_fA,drop2_fCY,drop2_fCX, ...
%     drop2_fWHR,drop2_fNumel,drop2_fXM,drop2_fYM,drop2_fMiAx);

% Compare feature vector data returning threshold values for each feature
 tValues = FeatureComparison(drop1_fA,drop1_fCY,drop1_fCX, ...
    drop1_fWHR,drop1_fNumel,drop1_fXM,drop1_fYM,drop1_fMiAx,sWindow);
%
 tValues = FeatureComparison(drop2_fA,drop2_fCY,drop2_fCX, ...
    drop2_fWHR,drop2_fNumel,drop2_fXM,drop2_fYM,drop2_fMiAx,sWindow);
%
 tValues = FeatureComparison(drop3_fA,drop3_fCY,drop3_fCX, ...
    drop3_fWHR,drop3_fNumel,drop3_fXM,drop3_fYM,drop3_fMiAx,sWindow);
%
 tValues = FeatureComparison(drop4_fA,drop4_fCY,drop4_fCX, ...
    drop4_fWHR,drop4_fNumel,drop4_fXM,drop4_fYM,drop4_fMiAx,sWindow);

tValues = FeatureComparison(fA,fCY,fCX,fWHR,fNumel,fXM,fYM,fMiAx,sWindow);
```

A.2 Background

Listing from source file `Background.m`.

```
%% Compute the mean background pixel intensity and standard deviation

function [meanBackground, stdDev] = Background(vidObj,nFrames)

% Preallocated for speed
temp{1,1} = 0;

for ii = 1:nFrames
    frame = read(vidObj,ii);
    % cropping
    frame = imcrop(frame,[0 0 320 220]);
```

```
    %frameNormalized = Normalize(frame);
    %grayFrameNormalized = double(rgb2gray(frame));
    grayFrame = double(rgb2gray(frame));
    temp{ii} = grayFrame;
end

% Calculate the mean pixel intensity from the 25 frames
meanBackground = MeanPixelIntensity(temp);

% Calculate the standard deviation from the 25 frames
stdDev = StandardDeviation(temp);
```

A.3 MeanPixelIntensity

Listing from source file `MeanPixelIntensity.m`.

```
%% Compute the mean/median background pixel intensity

function [meanBackground] = MeanPixelIntensity(val)

% Frame dimensions
[rows,cols] = size(val{1,1});
% Number of frames to be processed
n = numel(val);
% Temporary variables stored in memory
meanBackground = zeros(rows,cols);
m = zeros(1,n);


% For each pixel position (ii,jj) in the new background image, one add
% together each pixel at position (ii,jj) from frame 1 to n. Then, finally,
% one divide the pixel sum m with n to get mean value.
for ii = 1:rows
    for jj = 1:cols
        for kk = 1:n
            %m =+ val{1,kk}(ii,jj); % mean
            m(kk) = val{1,kk}(ii,jj); % median
        end
        %meanBackground(ii,jj) = m/n; % mean
        meanBackground(ii,jj) = median(m); % median
    end
end
```

A.4 StandardDeviation

Listing from source file `StandardDeviation.m`.

```
%% Compute the standard deviation for a set of values 'val'

function [stdDev] = StandardDeviation(val)

% Info: http://en.wikipedia.org/wiki/Standard_deviation

% Frame dimensions
[rows,cols] = size(val{1,1});
% Number of frames to be processed
n = numel(val);
% New temporary image stored in memory
stdDev = zeros(rows,cols);
setOfValues = zeros(1,n);
m = zeros(1,n);
```

```
% At each pixel position (ii,jj) of the 'n' frames in 'val' one extract all
% values from position (ii,jj) and add these to calculate the mean, and
% also store the values in a vector. The vector along with the mean is used
% to calculate the standard deviation at each pixel position (ii,jj).
for ii = 1:rows
    for jj = 1:cols
        for kk = 1:n
            m(kk) = val{1,kk}(ii,jj);
            %m =+ val{1,kk}(ii,jj); % mean
            %setOfValues(kk) = val{1,kk}(ii,jj); % median
        end
        M = m/n; % mean
        %M = mean(m); % median
        stdDev(ii,jj) = sqrt(sum((setOfValues - M).^2)/(n-1));
        %stdDev(ii,jj) = std(M,1);
    end
end
```

A.5 FramePreProcessing

Listing from source file `FramePreProcessing.m`.

```
%% Preprocessing on each frame

function [binFrame] = FramePreProcessing(val,meanBackground...
                        ,stdDev,confidenceLevel,z)

vidFrame = val;

% SKIPPING THE NORMALIZATION-PART

% Normalize RGB values in vidFrames
% vidFrameNormalized = Normalize(vidFrame);
% clear vidFrame;

% grayFrameNormalized = double(rgb2gray(vidFrameNormalized));
% clear vidFrameNormalized;

% Compute absolute difference of two images
% frameDiff = imabsdiff(grayFrameNormalized,meanBackground);
% clear grayFrameNormalized;

grayFrame = double(rgb2gray(vidFrame));
clear vidFrame;

% Compute absolute difference of two images
frameDiff = imabsdiff(grayFrame,meanBackground);
clear grayFrame;

% Make binary image based on the absolute difference between the
% current frame and the mean background image, and the stdandard
% deviation.
binFrame = Gray2Binary(frameDiff, stdDev, confidenceLevel);
clear frameDiff;

% Removes objects in the binary image that is smaller then z pixels
binFrame = bwareaopen(binFrame,z);

% Imclose(im,se) performs morphological closing on the
% grayscale or binary image 'bw' with the structuring element 'se'. SE
% must be a single structuring element object, as opposed to an array
% of objects.
binFrame = imclose(binFrame,strel('disk',5));
```

A.6 Gray2Binary

Listing from source file `Gray2Binary.m`.

```
%% Convert graytone image into binary

function [binFrame] = Gray2Binary(frameDiff, stdDev, confidenceLevel)

% Size of the image
[rows,cols] = size(frameDiff);

% New temporary image
temp = zeros(rows,cols);

for ii = 1:rows
    for jj = 1:cols
        if (frameDiff(ii,jj) > (stdDev(ii,jj)*confidenceLevel))
            temp(ii,jj) = 1;
        else
            temp(ii,jj) = 0;
        end
    end
end

binFrame = temp;
```

A.7 Bwlabel

Listing from source file `bwlabel.m`.

```
function [L,numComponents] = bwlabel(BW,mode)
%BWLABEL Label connected components in 2-D binary image.
%   L = BWLABEL(BW,N) returns a matrix L, of the same size as BW, containing
%   labels for the connected components in BW. N can have a value of either
%   4 or 8, where 4 specifies 4-connected objects and 8 specifies
%   8-connected objects; if the argument is omitted, it defaults to 8.
%
%   The elements of L are integer values greater than or equal to 0.  The
%   pixels labeled 0 are the background.  The pixels labeled 1 make up one
%   object, the pixels labeled 2 make up a second object, and so on.
%
%   [L,NUM] = BWLABEL(BW,N) returns in NUM the number of connected objects
%   found in BW.
%
%   Note: Comparing BWLABEL and BWLABELN
%   ------------------------------------
%   BWLABEL supports 2-D inputs only, whereas BWLABELN support any
%   input dimension.  In some cases you might prefer to use BWLABELN even
%   for 2-D problems because it can be faster.  If you have a 2-D input
%   whose objects are relatively "thick" in the vertical direction,
%   BWLABEL will probably be faster; otherwise BWLABELN will probably be
%   faster.
%
%   Class Support
%   -------------
%   BW can be logical or numeric, and it must be real, 2-D, and
%   nonsparse.  L is double.
%
%   Example
%   -------
%       BW = logical([1 1 1 0 0 0 0 0
%                     1 1 1 0 1 1 0 0
%                     1 1 1 0 1 1 0 0
%                     1 1 1 0 0 0 1 0
```

```
%                     1 1 1 0 0 0 1 0
%                     1 1 1 0 0 0 1 0
%                     1 1 1 0 0 1 1 0
%                     1 1 1 0 0 0 0 0]);
%       L = bwlabel(BW,4);
%       [r,c] = find(L == 2);
%
%   See also BWAREAOPEN, BWEULER, BWLABELN, BWSELECT, LABEL2RGB.

%   Copyright 1993-2005 The MathWorks, Inc.
%    Revision : 1.29.4.5  Date : 2006/06/1520 : 08 : 28

iptchecknargin(1,2,nargin,mfilename);
iptcheckinput(BW, {'logical' 'numeric'}, {'real', '2d', 'nonsparse'}, ...
              mfilename, 'BW', 1);

if (nargin < 2)
    mode = 8;
else
    iptcheckinput(mode, {'double'}, {'scalar'}, mfilename, 'N', 2);
end

if ~islogical(BW)
    BW = BW ~= 0;
end


[M,N] = size(BW);

% Compute run-length encoding and assign initial labels.
[sr,er,sc,labels,i,j] = bwlabel1(BW,mode);
if (isempty(labels))
    numLabels = 0;
else
    numLabels = max(labels);
end

% Create a sparse matrix representing the equivalence graph.
tmp = (1:numLabels)';
A = sparse([i;j;tmp], [j;i;tmp], 1, numLabels, numLabels);

% Determine the connected components of the equivalence graph
% and compute a new label vector.

% Find the strongly connected components of the adjacency graph
% of A.  dmperm finds row and column permutations that transform
% A into upper block triangular form.  Each block corresponds to
% a connected component; the original source rows in each block
% correspond to the members of the corresponding connected
% component.  The first two output% arguments (row and column
% permutations, respectively) are the same in this case because A
% is symmetric.  The vector r contains the locations of the
% blocks; the k-th block as indices r(k):r(k+1)-1.
[p,p,r] = dmperm(A);

% Compute vector containing the number of elements in each
% component.
sizes = diff(r);
numComponents = length(sizes);  % Number of components.

blocks = zeros(1,numLabels);
blocks(r(1:numComponents)) = 1;
blocks = cumsum(blocks);
blocks(p) = blocks;
labels = blocks(labels);

% Given label information, create output matrix.
```

```
L = bwlabel2(sr, er, sc, labels, M, N);
```

A.8 Regionprops

Listing from source file `regionprops.m`.

```
function outstats = regionprops(varargin)
%REGIONPROPS Measure properties of image regions (blob analysis).
%   STATS = REGIONPROPS(L,PROPERTIES) measures a set of properties for each
%   labeled region in the label matrix L. Positive integer elements of L
%   correspond to different regions. For example, the set of elements of L
%   equal to 1 corresponds to region 1; the set of elements of L equal to 2
%   corresponds to region 2; and so on. STATS is a structure array of length
%   max(L(:)). The fields of the structure array denote different properties
%   for each region, as specified by PROPERTIES.
%
%   PROPERTIES can be a comma-separated list of strings, a cell array
%   containing strings, the string 'all', or the string 'basic'. The set of
%   valid measurement strings includes:
%
%     'Area'              'ConvexHull'    'EulerNumber'
%     'Centroid'          'ConvexImage'   'Extrema'
%     'BoundingBox'       'ConvexArea'    'EquivDiameter'
%     'SubarrayIdx'       'Image'         'Solidity'
%     'MajorAxisLength'   'PixelList'     'Extent'
%     'MinorAxisLength'   'PixelIdxList'  'FilledImage'
%     'Orientation'                       'FilledArea'
%     'Eccentricity'                      'Perimeter'
%
%   Property strings are case insensitive and can be abbreviated.
%
%   If PROPERTIES is the string 'all', then all of the above measurements
%   are computed. If PROPERTIES is not specified or if it is the string
%   'basic', then these measurements are computed: 'Area', 'Centroid', and
%   'BoundingBox'.
%
%   Perimeter should be used on a label matrix with contiguous regions, such
%   as L = bwlabel(BW). Otherwise, 'perimeter' gives unexpected results on
%   discontiguous regions.
%
%   Note - REGIONPROPS and binary images
%   ------------------------------------
%   REGIONPROPS does not accept a binary image as its first input.  There
%   are two common ways to convert a binary image to a label matrix:
%
%       1.  L = bwlabel(BW);
%
%       2.  L = double(BW);
%
%   Suppose that BW were a logical matrix containing these values:
%
%       1 1 0 0 0 0
%       1 1 0 0 0 0
%       0 0 0 0 0 0
%       0 0 0 0 1 1
%       0 0 0 0 1 1
%
%   The first method of forming a label matrix, L = bwlabel(BW), results
%   in a label matrix containing two contiguous regions labeled by the
%   integer values 1 and 2.  The second method of forming a label matrix,
%   L = double(BW), results in a label matrix containing one
%   discontiguous region labeled by the integer value 1.  Since each
%   result is legitimately desirable in certain situations, REGIONPROPS
%   does not accept binary images and convert them using either method.
%   You should convert a binary image to a label matrix using one of
```

```
%   these methods (or another method if appropriate) before calling
%   REGIONPROPS.
%
%   Example
%   -------
%   Label the connected pixel components in the text.png image, compute
%   their centroids, and superimpose the centroid locations on the
%   image.
%
%       bw = imread('text.png');
%       L = bwlabel(bw);
%       s  = regionprops(L, 'centroid');
%       centroids = cat(1, s.Centroid);
%       imshow(bw)
%       hold on
%       plot(centroids(:,1), centroids(:,2), 'b*')
%       hold off
%
%   Class Support
%   -------------
%   The input label matrix L can have any numeric class.
%
%   See also BWLABEL, BWLABELN, ISMEMBER, WATERSHED.

%   Copyright 1993-2007 The MathWorks, Inc.
%    Revision.4.2.3  Date : 2007/03/2719 : 10 : 53

officialStats = {'Area'
                 'Centroid'
                 'BoundingBox'
                 'SubarrayIdx'
                 'MajorAxisLength'
                 'MinorAxisLength'
                 'Eccentricity'
                 'Orientation'
                 'ConvexHull'
                 'ConvexImage'
                 'ConvexArea'
                 'Image'
                 'FilledImage'
                 'FilledArea'
                 'EulerNumber'
                 'Extrema'
                 'EquivDiameter'
                 'Solidity'
                 'Extent'
                 'PixelIdxList'
                 'PixelList'
                 'Perimeter'};

tempStats = {'PerimeterCornerPixelList'};

allStats = [officialStats; tempStats];

[L, requestedStats] = ParseInputs(officialStats, varargin{:});

if ndims(L) > 2
  % Remove stats that aren't supported for N-D input and issue
  % warning messages as appropriate.
  requestedStats = PreprocessRequestedStats(requestedStats);
end

if isempty(requestedStats)
  eid = sprintf('Images:%s:noPropertiesWereSelected',mfilename);
  msg = 'No input properties';
  error(eid,'%s',msg);
end
```

```
if (isempty(L))
  numObjs = 0;
else
  numObjs = round(double(max(L(:))));
end

% Initialize the stats structure array.
numStats = length(allStats);
empties = cell(numStats, numObjs);
stats = cell2struct(empties, allStats, 1);

% Initialize the computedStats structure array.
zz = cell(numStats, 1);
for k = 1:numStats
  zz{k} = 0;
end
computedStats = cell2struct(zz, allStats, 1);

% Calculate PixelIdxList
[stats, computedStats] = ComputePixelIdxList(L, stats, computedStats, ...
                                             numObjs);

% Compute statistics.
for k = 1:length(requestedStats)
    switch requestedStats{k}

    case 'Area'
        [stats, computedStats] = ComputeArea(L, stats, computedStats);

    case 'FilledImage'
        [stats, computedStats] = ComputeFilledImage(L,stats,computedStats);

    case 'FilledArea'
        [stats, computedStats] = ComputeFilledArea(L,stats,computedStats);

    case 'ConvexArea'
        [stats, computedStats] = ComputeConvexArea(L, stats, computedStats);

    case 'Centroid'
        [stats, computedStats] = ComputeCentroid(L, stats, computedStats);

    case 'EulerNumber'
        [stats, computedStats] = ComputeEulerNumber(L,stats,computedStats);

    case 'EquivDiameter'
        [stats, computedStats] = ComputeEquivDiameter(L, stats, computedStats);

    case 'Extrema'
        [stats, computedStats] = ComputeExtrema(L, stats, computedStats);

    case 'BoundingBox'
        [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);

    case 'SubarrayIdx'
        [stats, computedStats] = ComputeSubarrayIdx(L, stats, computedStats);

    case {'MajorAxisLength', 'MinorAxisLength', 'Orientation', 'Eccentricity'}
        [stats, computedStats] = ComputeEllipseParams(L, stats, ...
                                                      computedStats);

    case 'Solidity'
        [stats, computedStats] = ComputeSolidity(L, stats, computedStats);

    case 'Extent'
        [stats, computedStats] = ComputeExtent(L, stats, computedStats);
```

```
    case 'ConvexImage'
        [stats, computedStats] = ComputeConvexImage(L, stats, computedStats);

    case 'ConvexHull'
        [stats, computedStats] = ComputeConvexHull(L, stats, computedStats);

    case 'Image'
        [stats, computedStats] = ComputeImage(L, stats, computedStats);

    case 'PixelList'
        [stats, computedStats] = ComputePixelList(L, stats, computedStats);

    case 'Perimeter'
        [stats, computedStats] = ComputePerimeter(L, stats, computedStats);
    end
end

% Initialize the output stats structure array.
numStats = length(requestedStats);
empties = cell(numStats, numObjs);
outstats = cell2struct(empties, requestedStats, 1);

fnames = fieldnames(stats);
deleteStats = fnames(¬ismember(fnames,requestedStats));
outstats = rmfield(stats,deleteStats);

%%%
%%% ComputePixelIdxList
%%%
function [stats, computedStats] = ComputePixelIdxList(L, stats...
                                    ,computedStats,numobj)
%   A P-by-1 matrix, where P is the number of pixels belonging to
%   the region.  Each element contains the linear index of the
%   corresponding pixel.

computedStats.PixelIdxList = 1;

if ¬isempty(L) && numobj ≠ 0
  idxList = regionpropsmex(L, numobj);
  [stats.PixelIdxList] = deal(idxList{:});
end

%%%
%%% ComputeArea
%%%
function [stats, computedStats] = ComputeArea(L, stats, computedStats)
%   The area is defined to be the number of pixels belonging to
%   the region.

  if ¬computedStats.Area
    computedStats.Area = 1;

    for k = 1:length(stats)
      stats(k).Area = size(stats(k).PixelIdxList, 1);
    end
  end

%%%
%%% ComputeEquivDiameter
%%%
function [stats, computedStats] = ComputeEquivDiameter(L, stats, computedStats)
%   Computes the diameter of the circle that has the same area as
%   the region.
%   Ref: Russ, The Image Processing Handbook, 2nd ed, 1994, page
%   511.

  if ¬computedStats.EquivDiameter
```

```
    computedStats.EquivDiameter = 1;

    if ndims(L) > 2
      NoNDSupport('EquivDiameter');
      return
    end

    [stats, computedStats] = ComputeArea(L, stats, computedStats);

    factor = 2/sqrt(pi);
    for k = 1:length(stats)
      stats(k).EquivDiameter = factor * sqrt(stats(k).Area);
    end
  end

%%%
%%% ComputeFilledImage
%%%
function [stats, computedStats] = ComputeFilledImage(L,stats,computedStats)
%   Uses imfill to fill holes in the region.

  if ~computedStats.FilledImage
    computedStats.FilledImage = 1;

    [stats, computedStats] = ComputeImage(L, stats, computedStats);

    conn = conndef(ndims(L),'minimal');

    for k = 1:length(stats)
      stats(k).FilledImage = imfill(stats(k).Image,conn,'holes');
    end
  end

%%%
%%% ComputeConvexArea
%%%
function [stats, computedStats] = ComputeConvexArea(L, stats, computedStats)
%   Computes the number of "on" pixels in ConvexImage.

  if ~computedStats.ConvexArea
    computedStats.ConvexArea = 1;

    if ndims(L) > 2
      NoNDSupport('ConvexArea');
      return
    end

    [stats, computedStats] = ComputeConvexImage(L, stats, computedStats);

    for k = 1:length(stats)
      stats(k).ConvexArea = sum(stats(k).ConvexImage(:));
    end
  end

%%%
%%% ComputeFilledArea
%%%
function [stats, computedStats] = ComputeFilledArea(L,stats,computedStats)
%   Computes the number of "on" pixels in FilledImage.

  if ~computedStats.FilledArea
    computedStats.FilledArea = 1;

    [stats, computedStats] = ComputeFilledImage(L,stats,computedStats);

    for k = 1:length(stats)
      stats(k).FilledArea = sum(stats(k).FilledImage(:));
```

```
    end
  end

%%%
%%% ComputeConvexImage
%%%
function [stats, computedStats] = ComputeConvexImage(L, stats, computedStats)
%   Uses ROIPOLY to fill in the convex hull.

  if ~computedStats.ConvexImage
    computedStats.ConvexImage = 1;

    if ndims(L) > 2
      NoNDSupport('ConvexImage');
      return
    end

    [stats, computedStats] = ComputeConvexHull(L, stats, computedStats);
    [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);

    for k = 1:length(stats)
      M = stats(k).BoundingBox(4);
      N = stats(k).BoundingBox(3);
      hull = stats(k).ConvexHull;
      if (isempty(hull))
        stats(k).ConvexImage = false(M,N);
      else
        firstRow = stats(k).BoundingBox(2) + 0.5;
        firstCol = stats(k).BoundingBox(1) + 0.5;
        r = hull(:,2) - firstRow + 1;
        c = hull(:,1) - firstCol + 1;
        stats(k).ConvexImage = roipoly(M, N, c, r);
      end
    end
  end

%%%
%%% ComputeCentroid
%%%
function [stats, computedStats] = ComputeCentroid(L, stats, computedStats)
%   [mean(r) mean(c)]

  if ~computedStats.Centroid
    computedStats.Centroid = 1;

    [stats, computedStats] = ComputePixelList(L, stats, computedStats);

    for k = 1:length(stats)
      stats(k).Centroid = mean(stats(k).PixelList,1);
    end

  end

%%%
%%% ComputeEulerNumber
%%%
function [stats, computedStats] = ComputeEulerNumber(L,stats,computedStats)
%   Calls BWEULER on 'Image' using 8-connectivity

  if ~computedStats.EulerNumber
    computedStats.EulerNumber = 1;

    if ndims(L) > 2
      NoNDSupport('EulerNumber');
      return
    end
```

```
    [stats, computedStats] = ComputeImage(L, stats, computedStats);

    for k = 1:length(stats)
      stats(k).EulerNumber = bweuler(stats(k).Image,8);
    end
  end

%%%
%%% ComputeExtrema
%%%
function [stats, computedStats] = ComputeExtrema(L, stats, computedStats)
%   A 8-by-2 array; each row contains the x and y spatial
%   coordinates for these extrema:  leftmost-top, rightmost-top,
%   topmost-right, bottommost-right, rightmost-bottom, leftmost-bottom,
%   bottommost-left, topmost-left.
%   reference: Haralick and Shapiro, Computer and Robot Vision
%   vol I, Addison-Wesley 1992, pp. 62-64.

  if ~computedStats.Extrema
    computedStats.Extrema = 1;

    if ndims(L) > 2
      NoNDSupport('Extrema');
      return
    end

    [stats, computedStats] = ComputePixelList(L, stats, computedStats);

    for k = 1:length(stats)
      pixelList = stats(k).PixelList;
      if (isempty(pixelList))
        stats(k).Extrema = zeros(8,2) + 0.5;
      else
        r = pixelList(:,2);
        c = pixelList(:,1);

        minR = min(r);
        maxR = max(r);
        minC = min(c);
        maxC = max(c);

        minRSet = find(r==minR);
        maxRSet = find(r==maxR);
        minCSet = find(c==minC);
        maxCSet = find(c==maxC);

        % Points 1 and 2 are on the top row.
        r1 = minR;
        r2 = minR;
        % Find the minimum and maximum column coordinates for
        % top-row pixels.
        tmp = c(minRSet);
        c1 = min(tmp);
        c2 = max(tmp);

        % Points 3 and 4 are on the right column.
        % Find the minimum and maximum row coordinates for
        % right-column pixels.
        tmp = r(maxCSet);
        r3 = min(tmp);
        r4 = max(tmp);
        c3 = maxC;
        c4 = maxC;

        % Points 5 and 6 are on the bottom row.
        r5 = maxR;
        r6 = maxR;
```

```
        % Find the minimum and maximum column coordinates for
        % bottom-row pixels.
        tmp = c(maxRSet);
        c5 = max(tmp);
        c6 = min(tmp);

        % Points 7 and 8 are on the left column.
        % Find the minimum and maximum row coordinates for
        % left-column pixels.
        tmp = r(minCSet);
        r7 = max(tmp);
        r8 = min(tmp);
        c7 = minC;
        c8 = minC;

        stats(k).Extrema = [c1-0.5 r1-0.5
                            c2+0.5 r2-0.5
                            c3+0.5 r3-0.5
                            c4+0.5 r4+0.5
                            c5+0.5 r5+0.5
                            c6-0.5 r6+0.5
                            c7-0.5 r7+0.5
                            c8-0.5 r8-0.5];
      end
    end

  end

%%%
%%% ComputeBoundingBox
%%%
function [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats)
%   [minC minR width height]; minC and minR end in .5.

  if ~computedStats.BoundingBox
    computedStats.BoundingBox = 1;

    [stats, computedStats] = ComputePixelList(L, stats, computedStats);

    num_dims = ndims(L);

    for k = 1:length(stats)
      list = stats(k).PixelList;
      if (isempty(list))
        stats(k).BoundingBox = [0.5*ones(1,num_dims) zeros(1,num_dims)];
      else
        min_corner = min(list,[],1) - 0.5;
        max_corner = max(list,[],1) + 0.5;
        stats(k).BoundingBox = [min_corner (max_corner - min_corner)];
      end
    end
  end

%%%
%%% ComputeSubarrayIdx
%%%
function [stats, computedStats] = ComputeSubarrayIdx(L, stats, computedStats)
%   Find a cell-array containing indices so that L(idx{:}) extracts the
%   elements of L inside the bounding box.

  if ~computedStats.SubarrayIdx
    computedStats.SubarrayIdx = 1;

    [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);
    num_dims = ndims(L);
    idx = cell(1,num_dims);
    for k = 1:length(stats)
```

```
      boundingBox = stats(k).BoundingBox;
      left = boundingBox(1:(end/2));
      right = boundingBox((1+end/2):end);
      left = left(1,[2 1 3:end]);
      right = right(1,[2 1 3:end]);
      for p = 1:num_dims
        first = left(p) + 0.5;
        last = first + right(p) - 1;
        idx{p} = first:last;
      end
      stats(k).SubarrayIdx = idx;
    end
  end

%%%
%%% ComputeEllipseParams
%%%
function [stats, computedStats] = ComputeEllipseParams(L, stats, ...
                                                       computedStats)
%   Find the ellipse that has the same normalized second central moments as the
%   region.  Compute the axes lengths, orientation, and eccentricity of the
%   ellipse.  Ref: Haralick and Shapiro, Computer and Robot Vision vol I,
%   Addison-Wesley 1992, Appendix A.


  if ~(computedStats.MajorAxisLength && computedStats.MinorAxisLength && ...
       computedStats.Orientation && computedStats.Eccentricity)
    computedStats.MajorAxisLength = 1;
    computedStats.MinorAxisLength = 1;
    computedStats.Eccentricity = 1;
    computedStats.Orientation = 1;

    if ndims(L) > 2
      NoNDSupport({'MajorAxisLength', 'MinorAxisLength', ...
                   'Eccentricity', 'Orientation'});
      return
    end

    [stats, computedStats] = ComputePixelList(L, stats, computedStats);
    [stats, computedStats] = ComputeCentroid(L, stats, computedStats);

    for k = 1:length(stats)
      list = stats(k).PixelList;
      if (isempty(list))
        stats(k).MajorAxisLength = 0;
        stats(k).MinorAxisLength = 0;
        stats(k).Eccentricity = 0;
        stats(k).Orientation = 0;

      else
        % Assign X and Y variables so that we're measuring orientation
        % counterclockwise from the horizontal axis.

        xbar = stats(k).Centroid(1);
        ybar = stats(k).Centroid(2);

        x = list(:,1) - xbar;
        y = -(list(:,2) - ybar); % This is negative for the
                                 % orientation calculation (measured in the
                                 % counter-clockwise direction).

        N = length(x);

        % Calculate normalized second central moments for the region. 1/12 is
        % the normalized second central moment of a pixel with unit length.
        uxx = sum(x.^2)/N + 1/12;
        uyy = sum(y.^2)/N + 1/12;
```

```
        uxy = sum(x.*y)/N;

        % Calculate major axis length, minor axis length, and eccentricity.
        common = sqrt((uxx - uyy)^2 + 4*uxy^2);
        stats(k).MajorAxisLength = 2*sqrt(2)*sqrt(uxx + uyy + common);
        stats(k).MinorAxisLength = 2*sqrt(2)*sqrt(uxx + uyy - common);
        stats(k).Eccentricity = 2*sqrt((stats(k).MajorAxisLength/2)^2 - ...
                                       (stats(k).MinorAxisLength/2)^2) / ...
            stats(k).MajorAxisLength;

        % Calculate orientation.
        if (uyy > uxx)
          num = uyy - uxx + sqrt((uyy - uxx)^2 + 4*uxy^2);
          den = 2*uxy;
        else
          num = 2*uxy;
          den = uxx - uyy + sqrt((uxx - uyy)^2 + 4*uxy^2);
        end
        if (num == 0) && (den == 0)
          stats(k).Orientation = 0;
        else
          stats(k).Orientation = (180/pi) * atan(num/den);
        end
      end
    end

  end

%%%
%%% ComputeSolidity
%%%
function [stats, computedStats] = ComputeSolidity(L, stats, computedStats)
%   Area / ConvexArea

  if ~computedStats.Solidity
    computedStats.Solidity = 1;

    if ndims(L) > 2
      NoNDSupport('Solidity');
      return
    end

    [stats, computedStats] = ComputeArea(L, stats, computedStats);
    [stats, computedStats] = ComputeConvexArea(L, stats, computedStats);

    for k = 1:length(stats)
      if (stats(k).ConvexArea == 0)
        stats(k).Solidity = NaN;
      else
        stats(k).Solidity = stats(k).Area / stats(k).ConvexArea;
      end
    end
  end

%%%
%%% ComputeExtent
%%%
function [stats, computedStats] = ComputeExtent(L, stats, computedStats)
%   Area / (BoundingBox(3) * BoundingBox(4))

  if ~computedStats.Extent
    computedStats.Extent = 1;

    if ndims(L) > 2
      NoNDSupport('Extent');
      return
    end
```

```
    [stats, computedStats] = ComputeArea(L, stats, computedStats);
    [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);

    for k = 1:length(stats)
      if (stats(k).Area == 0)
        stats(k).Extent = NaN;
      else
        stats(k).Extent = stats(k).Area / prod(stats(k).BoundingBox(3:4));
      end
    end
  end

%%%
%%% ComputeImage
%%%
function [stats, computedStats] = ComputeImage(L, stats, computedStats)
%   Binary image containing "on" pixels corresponding to pixels
%   belonging to the region.  The size of the image corresponds
%   to the size of the bounding box for each region.

  if ~computedStats.Image
    computedStats.Image = 1;

    [stats, computedStats] = ComputeSubarrayIdx(L, stats, computedStats);

    for k = 1:length(stats)
      subarray = L(stats(k).SubarrayIdx{:});
      if ~isempty(subarray)
        stats(k).Image = (subarray == k);
      else
        stats(k).Image = logical(subarray);
      end
    end
  end


%%%
%%% ComputePixelList
%%%
function [stats, computedStats] = ComputePixelList(L, stats, computedStats)
%   A P-by-2 matrix, where P is the number of pixels belonging to
%   the region.  Each row contains the row and column
%   coordinates of a pixel.

  if ~computedStats.PixelList
    computedStats.PixelList = 1;

    % Convert the linear indices to subscripts and store
    % the results in the pixel list.  Reverse the order of the first
    % two subscripts to form x-y order.
    In = cell(1,ndims(L));
    for k = 1:length(stats)
      if ~isempty(stats(k).PixelIdxList)
        [In{:}] = ind2sub(size(L), stats(k).PixelIdxList);
        stats(k).PixelList = [In{:}];
        stats(k).PixelList = stats(k).PixelList(:,[2 1 3:end]);
      else
        stats(k).PixelList = zeros(0,ndims(L));
      end
    end
  end

%%%
%%% ComputePerimeterCornerPixelList
%%%
function [stats, computedStats] = ComputePerimeterCornerPixelList(L, ...
```

```
                                                        stats, computedStats)
  %   Find the pixels on the perimeter of the region; make a list
  %   of the coordinates of their corners; sort and remove
  %   duplicates.

  if ~computedStats.PerimeterCornerPixelList
    computedStats.PerimeterCornerPixelList = 1;

    if ndims(L) > 2
      NoNDSupport('PerimeterCornerPixelList');
      return
    end

    [stats, computedStats] = ComputeImage(L, stats, computedStats);
    [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);

    for k = 1:length(stats)
      perimImage = bwmorph(stats(k).Image, 'perim8');
      firstRow = stats(k).BoundingBox(2) + 0.5;
      firstCol = stats(k).BoundingBox(1) + 0.5;
      [r,c] = find(perimImage);
      % Force rectangular empties.
      r = r(:) + firstRow - 1;
      c = c(:) + firstCol - 1;
      rr = [r-.5 ; r    ; r+.5 ; r   ];
      cc = [c    ; c+.5 ; c    ; c-.5];
      stats(k).PerimeterCornerPixelList = [cc rr];
    end

  end

%%%
%%% ComputeConvexHull
%%%
function [stats, computedStats] = ComputeConvexHull(L, stats, computedStats)
%   A P-by-2 array representing the convex hull of the region.
%   The first column contains row coordinates; the second column
%   contains column coordinates.  The resulting polygon goes
%   through pixel corners, not pixel centers.

  if ~computedStats.ConvexHull
    computedStats.ConvexHull = 1;

    if ndims(L) > 2
      NoNDSupport('ConvexHull');
      return
    end

    [stats, computedStats] = ComputePerimeterCornerPixelList(L, stats, ...
                                                        computedStats);
    [stats, computedStats] = ComputeBoundingBox(L, stats, computedStats);

    for k = 1:length(stats)
      list = stats(k).PerimeterCornerPixelList;
      if (isempty(list))
        stats(k).ConvexHull = zeros(0,2);
      else
        rr = list(:,2);
        cc = list(:,1);
        hullIdx = convhull(rr, cc);
        stats(k).ConvexHull = list(hullIdx,:);
      end
    end
  end

%%%
%%% ComputePerimeter
```

```
%%%
function [stats, computedStats] = ComputePerimeter(L, stats, computedStats)

  if ¬computedStats.Perimeter
    computedStats.Perimeter = 1;

    if ndims(L) > 2
      NoNDSupport('ComputePerimeter');
      return
    end

    B = regionboundariesmex(double(L),8);

    for i = 1:length(B)
      boundary = B{i};
      ∆ = diff(boundary).^2;
      stats(i).Perimeter = sum(sqrt(sum(∆,2)));
    end
  end

%%%
%%% ParseInputs
%%%
function [L,reqStats] = ParseInputs(officialStats, varargin)

  L = [];
  reqStats = [];

  if (length(varargin) < 1)
    eid = sprintf('Images:%s:tooFewInputs',mfilename);
    msg = 'Too few input arguments.';
    error(eid,'%s',msg);
  end

  L = varargin{1};

  if islogical(L)
    eid = 'Images:regionprops:binaryInput';
    msg1 = 'Use bwlabel(BW) or double(BW) convert binary image to ';
    msg2 = 'a label matrix before calling regionprops.';
    msg = sprintf('%s\n%s',msg1,msg2);
    error(eid, '%s', msg);
  end

  iptcheckinput(L, {'numeric'}, {'real', 'integer', 'nonnegative'}, ...
                mfilename, 'L', 1);

  list = varargin(2:end);
  if (¬isempty(list) && ¬iscell(list{1}) && strcmp(lower(list{1}), 'all'))
    reqStats = officialStats;
    reqStatsIdx = 1:length(officialStats);

  elseif (isempty(list) || (¬iscell(list{1}) && strcmp(lower(list{1}),'basic')))
    % Default list
    reqStats = {'Area'
                'Centroid'
                'BoundingBox'};
  else

    if (iscell(list{1}))
      list = list{1};
    end
    list = list(:);

    officialStatsL = lower(officialStats);

    reqStatsIdx = [];
```

```
    eid = sprintf('Images:%s:invalidMeasurement',mfilename);
    for k = 1:length(list)
      if (¬ischar(list{k}))
        msg = sprintf('This measurement is not a string: "%d".', list{k});
        error(eid,'%s',msg);
      end

      idx = strmatch(lower(list{k}), officialStatsL);
      if (isempty(idx))
        msg = sprintf('Unknown measurement: "%s".', list{k});
        error(eid,'%s',msg);

      elseif (length(idx) > 1)
        msg = sprintf('Ambiguous measurement: "%s".', list{k});
        error(eid,'%s',msg);

      else
        reqStatsIdx = [reqStatsIdx; idx];
      end
    end

    reqStats = officialStats(reqStatsIdx);
  end

%%%
%%% NoNDSupport
%%%
function NoNDSupport(str)
%   Issue a warning message about lack of N-D support for a given
%   measurement or measurements.

  wid = sprintf('Images:%s:measurementNotForN-D',mfilename);

  if iscell(str)
    warn_str = sprintf('%s: %s ', ...
                       'These measurements are not supported if...
                       ndims(L) > 2.', sprintf('%s ', str{:}));
  else
    warn_str = sprintf('%s: %s', ...
                       'This measurement is not supported if...
                       ndims(L) > 2.', str);
  end

  warning(wid,'%s',warn_str);

%%%
%%% PreprocessRequestedStats
%%%
function requestedStats = PreprocessRequestedStats(requestedStats)
%   Remove any requested stats that are not supported for N-D input
%   and issue an appropriate warning.

  no_nd_measurements = {'MajorAxisLength'
                        'MinorAxisLength'
                        'Eccentricity'
                        'Orientation'
                        'ConvexHull'
                        'ConvexImage'
                        'ConvexArea'
                        'EulerNumber'
                        'Extrema'
                        'EquivDiameter'
                        'Solidity'
                        'Extent'
                        'Perimeter'};

  bad_stats = find(ismember(requestedStats, no_nd_measurements));
```

```
  if ¬isempty(bad_stats)
    NoNDSupport(requestedStats(bad_stats));
  end

  requestedStats(bad_stats) = [];
```

A.9 XYMotion

Listing from source file `XYMotion.m`.

```
%% Compute speed and motion of blobs center position between consequtive frames

function [Xmotion Ymotion speed] = XYMotion(blobCentroidX,blobCentroidY)

% predefine variable
Xmotion(1,1) = 0;
Ymotion(1,1) = 0;
speed(1,1) = 0;

% Get the number of blobs (n) and length a video (l)
[n l] = size(blobCentroidX);

% Extract centroid values from blobs properties, and calculates
% displacement between current and previous frame for each axis
for ii = 1:n
    for jj = 1:l
        if (jj == 1)
            dX = blobCentroidX(ii,jj) - blobCentroidX(ii,jj);
            dY = blobCentroidY(ii,jj) - blobCentroidY(ii,jj);
        else
            dX = blobCentroidX(ii,jj) - blobCentroidX(ii,jj-1);
            dY = blobCentroidY(ii,jj) - blobCentroidY(ii,jj-1);
        end
        % Store xy-motion for computing the motion in other function
        Xmotion(ii,jj) = dX;
        Ymotion(ii,jj) = dY;

        % Calculate displacement in from point A to B
        d = double(sqrt((dX.^2)+(dY.^2)));
        % Speed equal displacement divided by time, and time is per frame (1)
        % .04 is the time in second (1/25, hence 25 fps)
        speed(ii,jj) = d/.04;
    end
end
```

A.10 Sort

Listing from source file `sort.m`.

```
%SORT   Sort in ascending or descending order.
%   For vectors, SORT(X) sorts the elements of X in ascending order.
%   For matrices, SORT(X) sorts each column of X in ascending order.
%   For N-D arrays, SORT(X) sorts the along the first non-singleton
%   dimension of X. When X is a cell array of strings, SORT(X) sorts
%   the strings in ASCII dictionary order.
%
%   Y = SORT(X,DIM,MODE)
%   has two optional parameters.
%   DIM selects a dimension along which to sort.
%   MODE selects the direction of the sort
%      'ascend' results in ascending order
%      'descend' results in descending order
```

```
%   The result is in Y which has the same shape and type as X.
%
%   [Y,I] = SORT(X,DIM,MODE) also returns an index matrix I.
%   If X is a vector, then Y = X(I).
%   If X is an m-by-n matrix and DIM=1, then
%       for j = 1:n, Y(:,j) = X(I(:,j),j); end
%
%   When X is complex, the elements are sorted by ABS(X).  Complex
%   matches are further sorted by ANGLE(X).
%
%   When more than one element has the same value, the order of the
%   elements are preserved in the sorted result and the indexes of
%   equal elements will be ascending in any index matrix.
%
%   Example: If X = [3 7 5
%                    0 4 2]
%
%   then sort(X,1) is [0 4 2  and sort(X,2) is [3 5 7
%                      3 7 5]                    0 2 4];
%
%   See also ISSORTED, SORTROWS, MIN, MAX, MEAN, MEDIAN.

%   Copyright 1984-2005 The MathWorks, Inc.
%   Revision: 5.16.4.5  Date: 2005/06/21 19:24:05
%   Built-in function.

%   Cell array implementation in @cell/sort.m
```

A.11 Filter

Listing from source file `Filter.m`.

```
%% Median filtering

function [fA,fCY,fCX,fWHR,fNumel,fXM,fYM,fMiAx] = Filter(sA,sCY,sCX,sWHR ...
                                                    ,Numel,sXM,sYM,sMiAx,mf)


% Transpose all input matrices, since the both the plot-function and the
% median filtering-function work colomnwise.
fA = sA';
%sP = sP';
fCY = sCY';
fCX = sCX';
fWHR = sWHR';
fNumel = Numel';
fYM = sYM';
fXM = sXM';
% sS = sS';
fMiAx = sMiAx';

w = mf;

% Median filtering, smoothing of the plot-lines
[m n] = size(fA);

for ii = 1:n
    fA(:,ii) = medfilt1(fA(:,ii),w);
    %blobBoundryPerimeterFilt(:,ii) = medfilt1(sP(:,ii),7);
    fCX(:,ii) = medfilt1(fCX(:,ii),w);
    fCY(:,ii) = medfilt1(fCY(:,ii),w);
    fWHR(:,ii) = medfilt1(fWHR(:,ii),w);
    fXM(:,ii) = medfilt1(fXM(:,ii),w);
    fYM(:,ii) = medfilt1(fYM(:,ii),w);
    %blobSpeedFilt(:,ii) = medfilt1(sS(:,ii),9);
```

```
    fMiAx(:,ii) = medfilt1(fMiAx(:,ii),w);
end

fNumel = medfilt1(fNumel,w);
```

A.12 Plotting

Listing from source file `Plotting.m`.

```
%% Plotting the feature data for analysis

function [] = Plotting(fA,fCY,fCX,fWHR,fNumel,fXM,fYM,fMiAx)

% % Transpose all input matrices, since the bot the plot-function and the
% % median filtering-function work colomnwise.
% fA = fA';
% %sP = sP';
% fCY = fCY';
% fCX = fCX';
% fWHR = fWHR';
% fNumel = fNumel';
% fYM = fYM';
% fXM = fXM';
% % sS = sS';
% fMiAx = fMiAx';

% filter window
% w = mf;

% Median filtering, smoothing of the plot-lines
% [m n] = size(sA);
% for ii = 1:n
%     blobAreaFilt(:,ii) = medfilt1(fA(:,ii),w);
%     %blobBoundryPerimeterFilt(:,ii) = medfilt1(sP(:,ii),7);
%     blobCenterXFilt(:,ii) = medfilt1(fCX(:,ii),w);
%     blobCenterYFilt(:,ii) = medfilt1(fCY(:,ii),w);
%     blobWidthHeightRatioFilt(:,ii) = medfilt1(fWHR(:,ii),w);
%     blobXmotionFilt(:,ii) = medfilt1(fXM(:,ii),w);
%     blobYmotionFilt(:,ii) = medfilt1(fYM(:,ii),w);
% %     blobSpeedFilt(:,ii) = medfilt1(sS(:,ii),9);
%     blobMinorAxis(:,ii) = medfilt1(fMiAx(:,ii),w);
% end
% blobNumelFilt = medfilt1(fNumel,w);

% Plot all matrices, the DisplayName give each line different colour
figure
subplot(2,1,1)
plot(fA,'DisplayName','Area')
title('- Area -','fontsize',10,'fontweight','b')
subplot(2,1,2)
plot(fMiAx,'DisplayName','Blob minor axis')
title('- Blob minor axis -','fontsize',10,'fontweight','b')

figure
subplot(2,1,1)
plot(fCX,'DisplayName','Centroid x-axis')
title('- Centroid x-axis -','fontsize',10,'fontweight','b')
subplot(2,1,2)
plot(fCY,'DisplayName','Centroid y-axis')
title('- Centroid y-axis -','fontsize',10,'fontweight','b')

figure
subplot(2,1,1)
plot(fWHR,'DisplayName','Width-height-ratio')
title('- Width-height-ratio -','fontsize',10,'fontweight','b')
```

```
subplot(2,1,2)
plot(fNumel,'DisplayName','Number of blobs')
title('— Number of blobs —','fontsize',10,'fontweight','b')

figure
subplot(2,1,1)
plot(fXM,'DisplayName','Blob x—motion')
title('— Blob x—motion —','fontsize',10,'fontweight','b')
subplot(2,1,2)
plot(fYM,'DisplayName','Blob y—motion')
title('— Blob y—motion —','fontsize',10,'fontweight','b')

% figure
% subplot(2,1,1)
% plot(blobSpeedFilt,'DisplayName','Blob speed')
% title('— Blob speed (pixels per frame) —','fontsize',10,'fontweight','b')
% subplot(2,1,2)
% plot(sort_blobYmotion,'DisplayName','Blob y—motion')
% title('— Blob y—motion —','fontsize',10,'fontweight','b')
```

A.13 FeatureComparison

Listing from source file `FeatureComparison.m`.

```
%% Compare feature vector data returning threshold values for each feature

function threshMatrix = FeatureComparison(fA,fCY,fCX,fWHR,fNumel,...
    fXM,fYM,fMiAx,sWindow)


aData = fA'; % filtered area data
cxData = fCX'; % filtered centroid x—axis data
cyData = fCY'; % filtered centroid y—axis data
ratioData = fWHR'; % filtered ratio data
nData = fNumel'; % filtered number of blobs in current frame
mxData = fXM'; % filtered motion x—direction
myData = fYM'; % filtered motion y—direction
minaxData = fMiAx'; % filtered minor axis length

w = sWindow; % search window size (2 times the framerate)


% Matrix storing framenumber where a feature returns the value of '1',
% after the program check if all features have the value of '1' at a common
% frame. If so, this represents an object split.
threshMatrix(1,1) = 0;


nFrames = length(nData);

for ii = 2:nFrames

    % As long as the difference in the number of blobs between a frame
    % and its previous frame is not 1, meaning no new blobs, the system
    % just continues to the next frame. If the difference between two frame
    % are of value 1, do the computation of the feature data
    newObj = abs(nData(1,ii)—nData(1,ii—1));
    if(newObj == 1 && nData(1,ii) == 2)
        disp(ii);

        tArea = areaThreshold(ii,w,aData);
        if (tArea == 1)
            threshMatrix(1,1) = ii;
        else
            threshMatrix(1,1) = 0;
```

```
        end

        x1=cxData(1,ii);x2=cxData(2,ii);
        y1=cyData(1,ii);y2=cyData(2,ii);
        ma=minaxData(1,ii);
        tCentroids = centerThreshold(x1,x2,y1,y2,ma);
        if (tCentroids == 1)
            threshMatrix(2,1) = ii;
        else
            threshMatrix(2,1) = 0;
        end

        tRatio = ratioThreshold(ii,w,ratioData);
        if (tRatio == 1)
            threshMatrix(3,1) = ii;
        else
            threshMatrix(3,1) = 0;
        end

        tMotion = motionThreshold(ii,w,mxData,myData);
        if (tMotion == 1)
            threshMatrix(4,1) = ii;
        else
            threshMatrix(4,1) = 0;
        end
    end

end


% % % % % % % % AREA % % % % % % % %
function tArea = areaThreshold(ii,w,aData)
% Make sure that we don't get a negativ startnumber for the searchwindow
if (ii > w)
    windowStart = ii-w;
else
    windowStart = 1;
end

% Data to be compared
obj1 = aData(1,windowStart:ii);
obj2 = aData(2,windowStart:ii);

% Value diff between consecutive frames
for jj = 2:numel(obj1)
    obj1Diff(1,jj) = (obj1(1,jj)-obj1(1,jj-1));
    obj2Diff(1,jj) = (obj2(1,jj)-obj2(1,jj-1));
end

for jj = 1:numel(obj1Diff)
    % abs diff between first and second object at each
    % frame within the search window
    absDiff(1,jj) = abs(obj1Diff(1,jj)-obj2Diff(1,jj));
end

% standard deviation of the abdiff above, and will act as threshold
t = std(absDiff);

if (absDiff(1,end) > 2*t)
    % Return value of '1' if distance area is inside the threshhold, if not '0'
    tArea = 1;
else
    tArea = 0;
end
```

```
% % % % % % % % CENTROIDS % % % % % % % %
function tCentroids = centerThreshold(x1,x2,y1,y2,ma)
% Set the distance threshold
minaxThresh = 2*(ma);
% Centroids distance at the likely blob-split point
d = sqrt(((x1-y1)^2) + ((x2-y2)^2));
if (d ≤ minaxThresh)
    % Return value of '1' if distance d is inside the minaxthresh, if not '0'
    tCentroids = 1;
else
    tCentroids = 0;
end



% % % % % % % % RATIO % % % % % % % %
function tRatio = ratioThreshold(ii,w,ratioData)
% Make sure that we don't get a negativ startnumber for the searchwindow
if (ii > w)
    windowStart = ii-w;
else
    windowStart = 1;
end

% Data to be compared
obj = ratioData(1,windowStart:ii);
objMean = mean(obj);
objStdDev = std(obj);

if (max(obj) > (objMean+objStdDev))
    tRatio = 1;
else
    tRatio = 0;
end


% % % % % % % % DIRECTIONAL INFORMATION % % % % % % % %
function tMotion = motionThreshold(ii,w,mxData,myData)
% Make sure that we don't get a negativ startnumber for the searchwindow
if (ii > w)
    windowStart = ii-w;
else
    windowStart = 1;
end

% Find which direction, x or y, that contains the best information
xMean = mean(mxData(1,:));
yMean = mean(myData(1,:));

% Data to be considered
if (xMean > yMean)
    obj = mxData(1,windowStart:ii);
else
    obj = myData(1,windowStart:ii);
end

% Value diff between consecutive frames
for jj = 2:numel(obj)
    objDiff(1,jj) = (obj(1,jj)-obj(1,jj-1));
end

% standard deviation act as threshold
t = std(objDiff);

for ii = 1:numel(obj)
    if(obj(ii) < t)
        start = ii;
```

```
    end
end

kk(1,1) = 0;
for jj = start:numel(obj)-1
    kk(jj) = abs(obj(jj+1)-obj(jj));
end

if(min(kk) < 0)
    tMotion = 0;
else
    tMotion = 1;
end
```