NTNU
Innovation and Creativity

Automatic recognition of unwanted behavior

Erik Sundnes Løvlie

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Ketil Bø, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The use of video surveillance in public areas is ever increasing. With that increase, it becomes impractical to continue using humans to view and respond to the surveillance video streams, due to the massive amount of information that must be processed. If one hope to use surveillance to avoid personal injuries, damage to property and so forth, instead of merely a forensic tool after the fact, humans must be replaced by artificial intelligence.

This thesis will examine the whole process of recognizing unwanted human behaviors from videos taken by surveillance cameras. Algorithms for recognizing specific, unwanted behaviors from surveillance video streams in real-time will be designed and implemented.

Assignment given: 20. January 2006
Supervisor: Ketil Bø, IDI

Abstract

The use of video surveillance in public areas is ever increasing. With that increase, it becomes impractical to continue using humans to view and respond to the surveillance video streams, due to the massive amount of information that must be processed. If one hope to use surveillance to avoid personal injuries, damage to property and so forth, instead of merely a forensic tool after the fact, humans must be replaced by artificial intelligence.

This thesis examines the whole process of recognizing unwanted human behaviors from videos taken by surveillance cameras. An overview of the state of the art in automated security and human behavior recognition is given. Algorithms for motion detection and tracking are described and implemented. The motion detection algorithm uses background subtraction, and can deal with large amounts of random noise. It also detects and removes cast shadows. The tracking algorithm uses a spatial occupancy overlap test between the predicted positions of tracked objects and current foreground blobs. Merges/splits are handled by grouping/ungrouping objects and recovering the trajectory using distance between predicted position and foreground blobs. Behaviors that are unwanted in most public areas are discussed, and a set of such concrete behaviors described. New algorithms for recognizing chasing/fleeing scenarios and people lying on the floor are then presented.

A real-time intelligent surveillance system capable of recognizing chasing/fleeing scenarios and people lying on the floor has been implemented, and results from analyzing real video sequences are presented. The thesis concludes with a discussion on the advantages and disadvantes of the presented algorithms, and suggestions for future research.

Declarations

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Trondheim, Norway, 16th June 2006

Erik Sundnes Løvlie

Preface

This Master's Thesis was written at the department of computer and information science (IDI) at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor, *Ketil Bø*, for giving valuable input during the work on this thesis. I would also like to thank *Knut Ragnar Holm*, who was kind enough to explain to me some of the basic theory of camera calibration.

Trondheim, Norway
16th June 2006

Erik Sundnes Løvlie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an overview of the problem area, and defines the objectives of this thesis. Concepts and terminology is introduced, and an overview of the rest of the thesis is given.

The chapter contains the following sections:

- Background
- Objectives
- Concepts and Terminology
- Overview of the Thesis

1.1 Background

The use of video surveillance is increasing rapidly, with the United Kingdom leading the way. There is no reason to believe this trend is not here to stay, given the increasingly violent acts of crime and terrorism targeting civilians. However, with an increasing number of areas being covered by an increasing number of cameras, human operators watching the video feeds have become a real bottleneck (fig. 1.1). No matter how well trained and alert, a human operator can watch only a very limited number of video feeds at once while still being able to detect all incidents and respond in a timely manner. This makes video surveillance an expensive and error prone process, and as a consequence it is not used for real-time interdiction, but rather as a deterrent and as a forensic tool after the fact.

For this reason automating video surveillance is currently undergoing massive research. The goal is to increase the number of incidents that are detected while decreasing the number of human operators needed. Intelligent video surveillance is also being used in psychological research [34]. Another application is in automatically extracting keyframes from a surveillance video.

The intelligent video surveillance systems that exist today are only *semi-automatic*, since the decision on how to react is still left to a human operator. The system is a tool

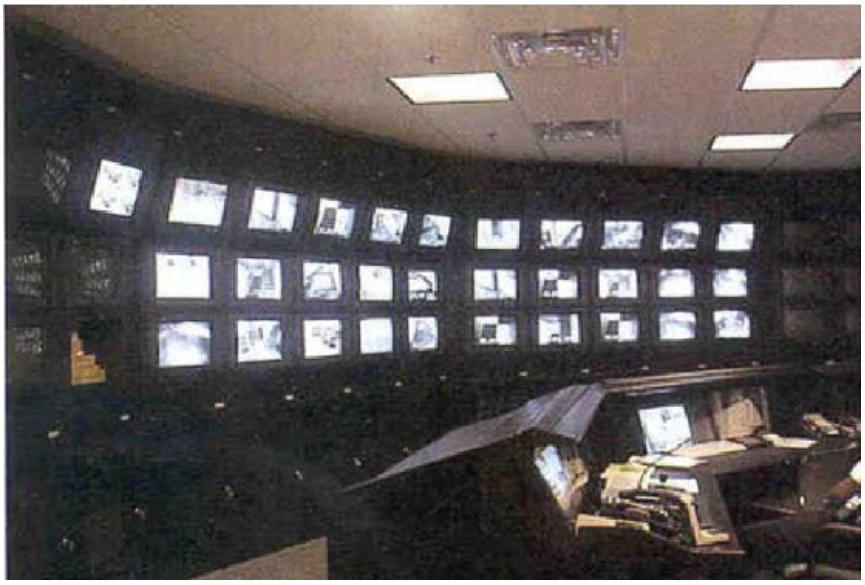


Figure 1.1: *Manual video surveillance [27].*

used by an operator to make the surveillance more effective. It achieves this by analyzing the video input and attracting the attention of the operator if an incident is detected.

Attracting the attention of the operator could be done by annotating the input frames to aid in the interpretation, or by issuing an alarm only when an incident is detected. The alarm is a notification to the human operator, not to the people under surveillance. Its intention is to allow the operator to decide whether a response must be made. A *false alarm* occurs when the system attracts the attention of the operator when there was in fact nothing of interest happening in the scene. This could happen because the designers made the system oversensitive to "guarantee" that no real incidents go by undetected, or because of noisy input. The challenge in designing an intelligent video surveillance system is to minimize the number of false alarms while maximizing the number of real incidents that are recognized.

1.2 Objectives

This thesis will examine *the whole process* of recognizing unwanted human behaviors from videos taken by surveillance cameras. Algorithms for *motion detection* and *tracking* will be described and implemented. Then unwanted behaviors that are of particular interest will be discussed, focusing on behaviors that could cause physical or psychological injury or discomfort to someone present. Finally algorithms that recognize specific unwanted behaviors will be designed and implemented.

1.3 Concepts and Terminology

- **Alarm**: a notification to the human operator that the scene under surveillance should be watched closely.
- **Blob**: a Binary, Large OBject. A connected component in an image where all pixels in the component have the same intensity level.
- **BSA**: background subtraction algorithm
- **Difference image**: the output of a BSA, that is: the background subtracted from the input image.
- **FG/BG**: abbreviation of foreground/background.
- **Foreground segmentation**: see *motion detection*.
- **Frame**: an image that is part of a sequence of images (a video) is usually called a frame.
- **Frame rate**: frames per second.
- **Motion detection**: the process of finding which pixels in a frame that correspond to moving objects in the scene. In this thesis, motion detection is regarded as synonymous with *foreground segmentation* and *preprocessing*.
- **Motion image**: the output of a motion detector, that is: an image of the same dimensions as the input image, where each pixel is classified as moving or not moving.
- **Moving pixels**: pixels marked by the motion detector.
- **Preprocessing**: see *motion detection*.
- **Reference image**: an image of the scene with no foreign objects, also called background.
- **ROI**: Region Of Interest
- **Scene**: the view of a surveillance camera.
- **Tracking**: the process of establishing a correspondence between blobs in the foreground in two consecutive frames.
- **VP**: vertical vanishing point. The point where images of edges in the scene that are perpendicular to some reference plane converge.

1.4 Overview of the Thesis

Chapter 1: Introduction
Background information, objectives and terminology.

Chapter 2: Behavior Recognition Theory and Previous Work
Gives an overview of the state of the art in human behavior recognition and automated security. Describes in detail algorithms for *motion detection* and *tracking*.

Chapter 3: Behavior Recognition
Discusses which behaviors are unwanted in the context of this thesis, and lists a set of concrete such behaviors that can be recognized. Then new algorithms for recognizing chasing/fleeing scenarios and people lying on the floor are presented.

Chapter 4: Implementation
Describes an implementation of the algorithms presented in chapters 2 and 3. The system parameters are explained, and usage information for the compiled executables given.

Chapter 5: Demonstration
Presents results obtained when testing in different environments. Results are presented from feet position estimation, recognition of chasing/fleeing scenarios and recognition of people lying on the floor, with all relevant system parameters.

Chapter 6: Conclusions
Summary of the thesis. Discussion of the advantages and disadvantages of the proposed algorithms, and suggestions for future research.

Chapter 2

Behavior Recognition Theory and Previous Work

This chapter gives an overview of the state of the art in behavior recognition and automated security, and describes the basic algorithms for *motion detection* and *people tracking*.

The chapter contains the following sections:

- Individual and Crowd Behavior
- Motion Detection
- People Tracking
- Individual Behavior Analysis

2.1 Individual and Crowd Behavior

Behavior can be analyzed on two different levels: the behavior of an *individual*, and the behavior of a *crowd*. The first is trying to detect the actions of an individual human, and the second is using statistical methods to detect "crowd actions" based on motion flow, speed and direction of the crowd itself. These two approaches are radically different, and so are the steps leading up to the actual behavior recognition. Each method requires some form of preprocessing which extracts the interesting features from the video stream.

Recognizing individual behavior requires detailed information about the people present in the scene. This naturally leads to the following sequence of steps:

1. **Motion Detection**: segmenting foreground from background in the individual frames of the surveillance video.
2. **Object Classification**: determining which moving objects correspond to people.
3. **Tracking**: tracking people as they move around the scene, keeping a record of previous positions and characteristics.

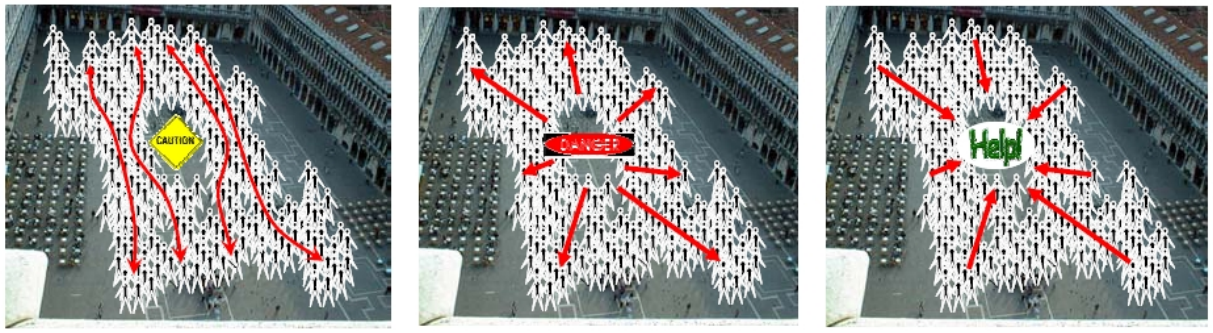


Figure 2.1: *Extracting security information from observing crowd motion [27]*
(a) Something is unusual (b) Something is dangerous (c) Something is attracting a crowd

4. **Behavior Analysis**: analyzing the features extracted in the previous steps in order to recognize the behavior of each person.

The reason for attempting to analyze crowd behavior instead of individual, is that it can be very difficult to get useful information when trying to recognize the actions of individuals in a crowded scene. This is because segmentation of individual objects become difficult due to frequent occlusions, that is people partially or completely covering other people in the scene. The motion detection step mentioned above would produce large blobs containing many of the people in the scene, which makes analysis of individual behavior impossible. Analyzing the motion of the crowd as a whole is often enough to determine that something unusual is going on, by detecting a disturbance in the typical flow. It is also possible to distinguish between different types of situations, such as "obstacle/unusual", "dangerous" and "attraction" [27] (see fig. 2.1). To recognize the behavior of a crowd, the first step is usually to find the *optical flow*.

The algorithms described and designed in this thesis are intended to be used in areas where the people-density is low enough to make detection of individual behavior possible. The rest of this chapter will therefore focus on explaining the theory and previous work related to individual behavior recognition.

2.2 Motion Detection

This section deals with the task of segmenting the foreground in the frames of the surveillance video. This thesis assumes single-sensor setup. Multi-camera setups are beyond the scope of this thesis.

2.2.1 Basic Principles

When a new frame comes in from the image acquisition system, it first enters the motion detector. The motion detector's task is to mark those pixels in the new frame which belong to moving objects in the scene. In other words: *to segment the foreground*.

There is a difference between segmenting the foreground and detecting moving pixels, depending on the definition of foreground. In this thesis, the foreground consists of any movable objects. That means people at rest are also part of the foreground, as are non-human objects that can be moved by the people in the scene. This is because stationary objects can play a part in the kind of situations that will be analyzed, being for instance victims of violence (i.e. unconscious person). Depending on the algorithms used in the motion detector, detecting stationary objects can be a task for the motion detector or the *tracker* (see chapter 2.3).

Foreground segmentation is usually achieved by subtracting a *model of the background* of the scene from each input frame (using a Background Subtraction Algorithm, or BSA). The background model can be static, or it can be updated to incorporate changes to the background, for instance changing lighting conditions. The result of the subtraction is the *difference image*, and large absolute values in this image corresponds to places where the new frame differs substantially from the background model. An object entering the scene will therefore produce relatively large absolute values in the area of the difference image it occupies. How large the difference is depends on the contrast between the object (the foreground) and the area of the scene that it covers (the background), or the *FG/BG contrast* at that location. Thresholding the difference image will then give an estimate of the moving pixels in the scene.

A very simple model of the background is an image of the scene captured at a time when no movable objects are present [18, 29, 30, 32, 41, 43]. If this image is never changed, it will mark both moving and stationary objects as foreground when subtracted from new frames. This is very simple to implement and computationally inexpensive while the system is online. It is also adequate in many environments, particularly some indoor environments. One drawback is that the scene must be empty before the background model can be constructed, which is not feasable in certain environments. Another serious drawback is that since the background model is never updated, the motion detector will essentially stop working if the lighting conditions change.

More advanced approaches include subtracting from the current frame the previous frame in the sequence, or maintaining more complex background models obtained using statistical analysis of the incoming frames [20, 45]. Many motion detectors in the litterature have some mechanism for updating the background model while online [47]. This is to cope with changing illumination, weather conditions, and so on. These motion detectors will then ignore stationary objects once they have been incorporated in the background model (due to the update process). Stationary objects are then an issue for the *tracker*, instead of the motion detector.

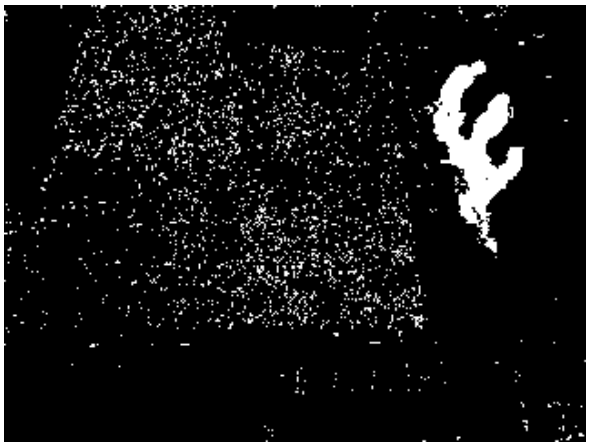
Figure 2.2 shows the motion detection in different stages, including the extensions that deal with noise and cast shadows that will be described in the next sections.

2.2.2 Problems

In practice the simple background-subtraction approach outlined above will lead to very poor results. One of the problems is noise introduced either in the camera or in analogue

(a) Input image

(b) Difference image, computed as the difference of filtered input and background images.



(c) Thresholded difference image. The threshold used was 10.

(d) Noise cleaned image. The size of the structuring element was 3×3, and the minimum region size was 200.

(e) Detected shadows overlayed on image. The algorithm used was [19]. The cncc threshold used was 0.7, and the size was 9×9.

(f) Segmented regions overlayed on image

Figure 2.2: *Motion detection step by step.*

transmission. The author can report quite a lot of random noise introduced by inexpensive digital cameras used in testing. CCTV surveillance cameras currently in use are usually not digital, and images produced by these are therefore often noisy due to the analogue transmission over a considerable distance.

Another problem is image degradation due to the compression. As noted in [42], the JPEG compression algorithm discards some color information from the image because it is less important to the human eye than brightness information. This can cause problems to algorithms that rely on color information.

The biggest problems are however *cast shadows* (fig. 2.3(a)), *reflections* and *low FG/BG contrast* (fig. 2.3(b)). Cast shadows and reflections should not be part of the segmented foreground. A simple BSA will however include shadows and reflections in the foreground due to the large absolute values in the difference image. Cast shadows are usually a bigger problem than reflections, due to the surfaces encountered in the environments where people surveillance is interesting. Cast shadows become a big problem when more than one person is in the scene, because even with no occlusion the blobs may melt together if the shadows are overlapping (fig. 2.3(c)). Also, shadows can be cast on nearby walls, producing what seems to be a new object entering the scene (fig. 2.3(d)).

A low FG/BG contrast results in the opposite problem: objects that are partially or completely left out of the foreground due to low absolute values in the difference image. Relatively often the color of a piece of clothing on a person's body matches the color in the background. This often has the effect of seperating the limbs from the torso of people under surveillance. Moreover, as the person is moving, the background relative to the person changes, resulting in a rapidly changing FG/BG contrast. This can cause a lot of merging and splitting between blobs in the foreground, even with only a single person in the scene. This could lead to errors in tracking.

All these problems can significantly reduce the quality of the output from the motion detector, thereby making behavior recognition more difficult at a later stage in the process. Next some algorithms for dealing with these problems are presented. Note that due to the complexity of this problem and the fact that the system must be operating in real-time, one cannot expect to get perfect results from the motion detector. This means the tracker must be able to deal with imperfect motion detection.

2.2.3 Algorithms

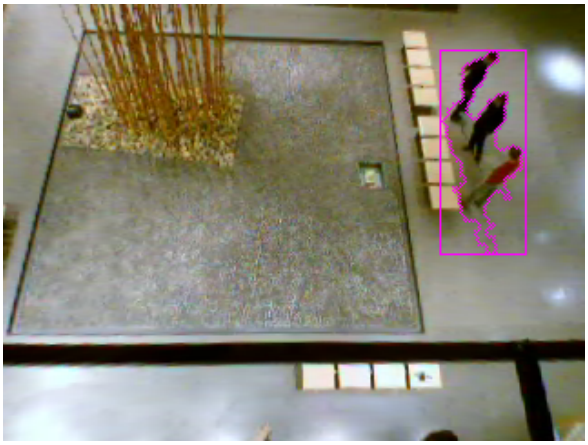
Suppressing Random Noise

As there are several images being used in the motion detector, there are different ways of suppressing noise depending on the image in question. As in [42, page 35], these are the images used:

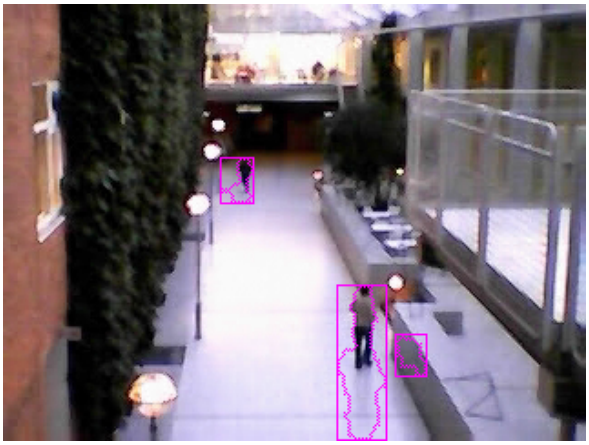
- **Background image**: if this can be created offline, noise is not a problem. The background image can be created using a pixelwise averaging or median filter over time [30]. This will make noise negligible if done over enough frames, assuming the background is not updated while online.

(a) Motion detection: shadow problem

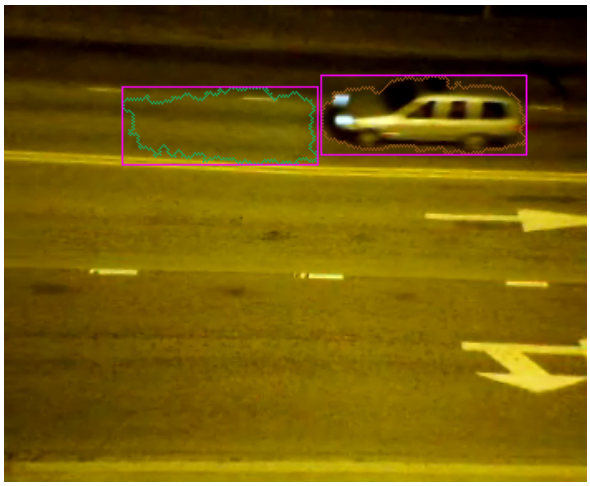
(b) Motion detection: contrast problem



(c) Motion detection: overlapping shadows

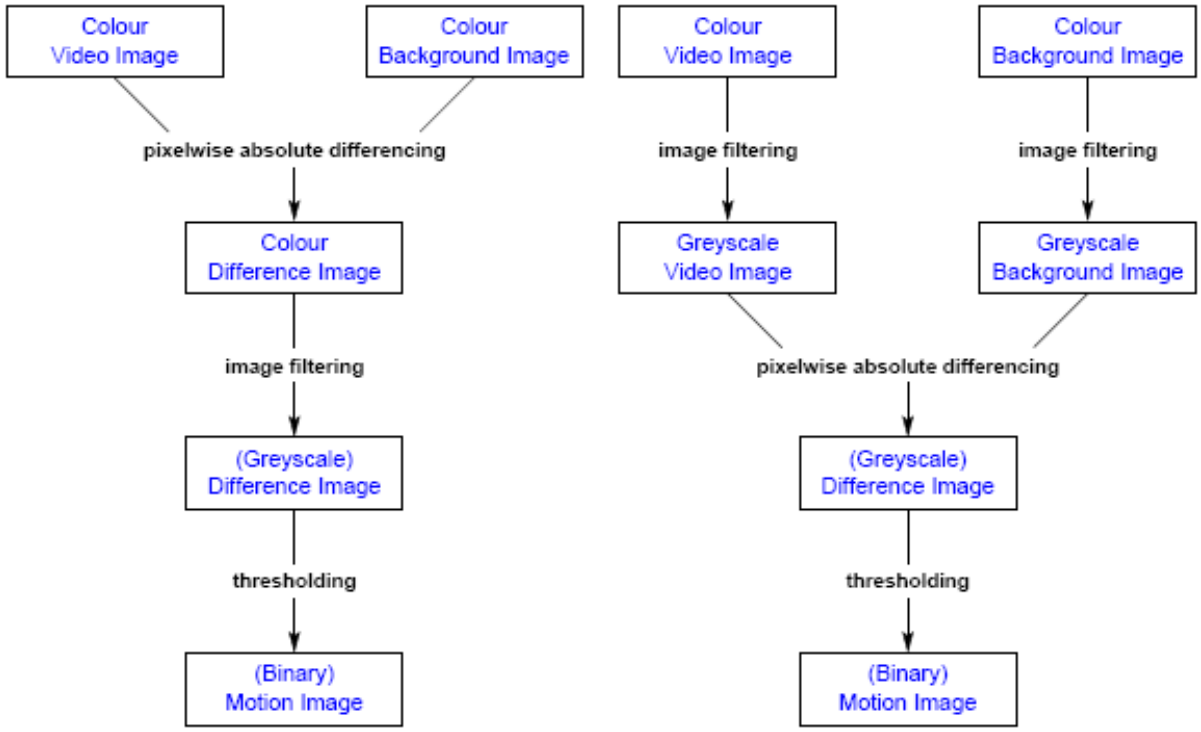


(d) Motion detection: shadows cast on walls



(e) Motion detection: illumination

Figure 2.3: *Main problems in motion detection: cast shadows and low FG/BG contrast. In figure 2.3(a) the shadow has been added to the foreground. In figure 2.3(b) the color of the trousers the person is wearing matches the color of the floor at that location, resulting in a low FG/BG contrast. Figure 2.3(c) shows the main problem regarding behavior recognition: blobs melting together due to overlapping shadows. Figure 2.3(e) shows a car that illuminates the road in front of it, thereby causing an error in foreground segmentation. This could also happen with a person carrying a torch, or wearing or carrying some highly reflective cloth or object.*



(a) Method a: filtering the difference image (b) Method b: differencing the filtered images

Figure 2.4: *Two alternative methods for creating the difference image [42, page 38].*

- **Input image**: noise is in general strong, depending on the environment and image acquisition system.
- **Difference image**: this image is the absolute difference between the input and background image, so any noise in the input image will also be present in the difference image. Each pixel in this image is a single number, typically an 8-bit integer.
- **Motion image**: this image is created from the thresholded difference image. Thus the amount of noise present in the motion image depends on the amplitude of the noise in the input and difference image.

It is most important to reduce the amount of noise in the difference image, as this is the foundation for the whole segmentation process. As noted in [42, page 38], there are two alternative methods of creating the difference image (see fig. 2.4):

(a) filtering the difference image

(b) differencing the filtered images

Filtering here means to map an RGB color image to a greyscale image. [42, page 38] obtained best results using *method b* and the filtering function CIE recommendation 709:

$$Y = .2125R + .7154G + .0721B$$

After thresholding the difference image, some random noise will be present in the form of isolated pixels. The amount of noise depends on the noise amplitude and the value of the threshold. Most of this noise can be removed by postprocessing the motion image using a combination of morphological operations and a flood-fill algorithm [20]. In [30] the author used the following approach with good results:

1. Removing one-pixel thick noise using one iteration of morphological erosion.
2. Region-based noise removal: using a flood-fill algorithm to find connected components, and removing those with an area below a certain threshold value.
3. Reconstructing by applying two iterations of dilation, followed by one iteration of erosion.

Figure 2.5 shows just how successful the noise cleaning approach described above can be. The video sequence used was the same as in figure 2.2, but the difference image threshold was decreased to 3. Note how good the end result is compared to the strong noise present in the thresholded difference image.

Detecting Shadows

Several algorithms for detecting and removing shadows are described in the litterature [10, 19, 38]. The author has implemented and evaluated the algorithms in [10] and [19].

There are several observations that may be used to detect shadow pixels:

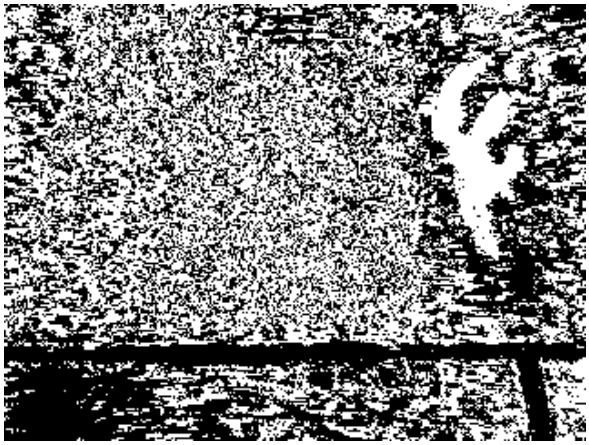
1. Pixels in a shadow are darker than the background.
2. Pixels in a shadow usually have the same color as the background.
3. The texture in a shadow is a scaled (darker) version of the background.

Detecting shadow pixels using color

The algorithm described in [10] requires color input images, and makes use of assumption 1 and 2 above to detect potential shadow pixels. It converts the input images to the HSV color space, in order to separate the color and brightness information. Then an image called a shadow mask is created, satisfying the following properties:

$$SP(x,y) = \begin{cases} 1 & \alpha \leq \frac{I^V(x,y)}{B^V(x,y)} \leq \beta \\ & \bigwedge((I^S(x,y) - B^S(x,y)) \leq \tau_S \\ & \bigwedge|(I^H(x,y) - B^H(x,y)| \leq \tau_H \\ 0 & otherwise \end{cases}$$

The rationale behind this is that a shadow pixel will be darker than the background, meaning the ratio of the input V to the background V will be less than 1 (β should be less than 1), and have a similar color, meaning the difference between the input and background



(a) Thresholded difference image. The threshold used was 3.

(b) Eroded

(c) Small regions removed

(d) Segmented regions (after shadow removal) overlayed on image

Figure 2.5: *This figure shows the drastic increase in random noise that occurs when the threshold on the difference image is decreased. The threshold used here was 3. All other parameters are the same is in figure 2.2. Note that the end result is almost the same as that of figure 2.2.*

hue and saturation is small. The shadow mask SP will have the value 1 where it is likely that the pixel is inside a shadow, and 0 all other places. Shadow pixels can then be removed from the motion image by AND'ing with the inverted shadow mask. See appendix A for matlab code.

This algorithm is simple and computationally inexpensive, and the results are reasonably good (fig. 2.6(a)). However, since it does not consider the texture, it will mark some pixels inside objects as shadow pixels. This will produce some holes inside objects when shadow pixels are removed. Performing shadow detection and removal before the morphological dilations mentioned in section 2.2.3 helps fill in some of the holes, but there will still be incorrectly detected shadow pixels.

Detecting shadow pixels using texture

The algorithm described in [19] uses all three observations above. It compares both the brightness and the texture of the input image with the background to find shadow pixels.

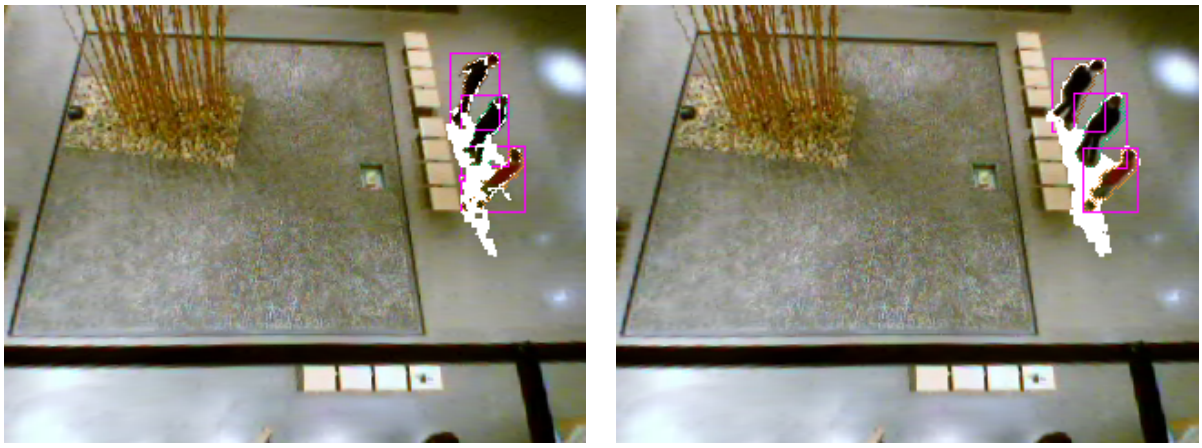
The algorithm works as follows: for each pixel in the current input image which is darker than the corresponding background pixel, an $n \times n$ normalized cross-correlation (NCC) with the background image is calculated. In [19], $n = 7$. The cross-correlation is only computed for pixels at the same positions. For shadow pixels the computed NCC will be close to 1, so shadow pixels can be removed by thresholding based on the NCC.

The NCC applies to greyscale images, not color images. To make use of color information, [19] defines the *hsL color space* and the *color normalized cross-correlation* (CNCC). First the input image is converted to the hsL color space. It is done by first converting the image to the well-known HSL space, and then converting the hue and saturation components to euclidean coordinates. The (h,s) vector is scaled so its length is equal to the saturation in HSL space. Next the CNCC between each pixel in the input and the corresponding pixel in the background is calculated, as with the NCC. Finally, as with the NCC, shadow pixels can be found by thresholding. See appendix B for matlab code.

As this algorithm compares texture similarity, and not just color, the results are generally better than the previous algorithm [10] (fig. 2.6(b)). It is computationally quite expensive, however, when compared to the algorithm in [10]. Although few holes should be produced inside objects using this algorithm, it is still best to remove shadow pixels before performing the morphological dilations mentioned in section 2.2.3.

Other shadow removal algorithms

It is possible to avoid using colors and simply use the normalized cross-correlation on greyscale images to compare texture similarity [19, 23]. Edges and gradients can be used [38]. Arnstein Ressem used active contours to remove shadows [37].



(a) Using brightness and color (b) Using brightness, color and texture

Figure 2.6: *Two alternative methods for detecting and removing shadow pixels. Pixels marked as shadow are shown white. The screenshots are from the author's implementations of the algorithms in [10] and [19] respectively. Compare to figure 2.3(c). See appendix A and B.*

2.3 People Tracking

This section deals with the task of tracking people in the surveillance video from frame to frame. This thesis assumes single-sensor setup. Multi-camera setups are beyond the scope of this thesis.

2.3.1 Basic Principles

The task of an *object tracker* is to establish a correspondence between blobs in successive motion images. The purpose of this is of course to make an analysis of the movements and behavior of the objects present in the scene possible. Part of this process may be to perform *object classification*, for instance classify a blob as an object in the set (car, human, animal, other). A *people tracker* is an object tracker with the specific goal of tracking blobs corresponding to humans.

In some environments it is safe to treat every blob in the motion image as corresponding to a human, for instance in some indoor environments, as non-human objects such as animals and cars are not likely to be moving around in these environments. Usually, however, an object classification algorithm must be used since different types of moving objects are expected [24, 26]. Typical features to use in the classification are area, perimeter, contour, histogram, motion pattern, etc.

Irrespective of object type, blobs in the current motion image must be matched to blobs in the previous motion image. Note that blobs in the previous motion image have already been tracked, and are therefore called *tracked objects*. As noted by Aggarwal in [4], there are two general approaches to tracking: using iconic or structural correspondence

models. The former means using correlation templates and the latter means extracting image features. In automated security (and in human motion analysis in general), using image features is the most common approach. This is because humans are non-rigid and articulated objects, and therefore relatively large changes in shape, size and position can occur between frames.

To use features in the tracking process one must first *extract* the interesting features from each blob, and then *match* these with the extracted features of previous frames (that is: the features of the tracked objects). Features commonly used are the bounding box and the centroid of each blob [18, 28, 32, 44]. Other possible features are color histograms, contour and body parts. If a structural model of the scene is available, the 3D position can also be used [32].

The matching can be performed by checking for overlapping bounding boxes, similarity in width and height of bounding boxes, similarity in the deviation of the centroid from the center of the bounding boxes [44], distance between the centroids, distance between the median coordinates [20], etc.

Some approaches to automated security utilize contextual information in the segmentation and tracking processes. If the surveillance system has access to a structural model of the scene it is monitoring, it may be able to compute the 3D positions of the humans present and use this information to increase the robustness of the tracking algorithm [11, 32, 43].

[39] uses a CAD model and a distributed surveillance system consisting of networked monocular cameras to track walking humans through a building. This is desirable because using multiple cameras effectively increases the system's field of view (FOV), because a person disappearing from the view of one camera can be picked up by another (handover). This means that as long as a person is in view of at least one camera at any time, it is possible to track that person throughout the area (a building, for instance).

The ADVISOR system [11, 12, 43] uses a 3D model of the scene and camera calibration to facilitate robust long-term tracking of individuals across cameras, even if they join or leave groups.

See table 2.1 for an overview of different people tracking systems.

2.3.2 Problems

The most obvious problem in tracking is the often poor quality of the motion detector output. As noted in section 2.2.2, cast shadows and low FG/BG contrast can cause major problems in tracking. Overlapping shadows cause blobs to melt together, which makes tracking of individuals difficult. If one of the shadow removal algorithms in section 2.2.3 is used, it will sometimes remove pixels that are not actually inside a shadow, resulting in damaged blobs. Low FG/BG contrast will often cause the limbs of a person to be separated from the torso, which can confuse the tracker (fig. 2.3(b)).

Another serious problem in the tracking procedure is *occlusion*. This is difficult to avoid, however the frequency of occlusions is highly dependent on the camera position and angle [30, 44] (see fig. 2.7). As with cast shadows, the problem with occlusion is that it makes tracking of individuals difficult.

System	Area Indoor (I) Outdoor (O)	Sensor Colour (C) Grayscale (G)	Camera Single (S) Stereo (O) Multiple (M)	Detection Single Gaussian (S) Bimodal (B) Mixture of Gaussian (M)	Tracking People Single Isolated (S) Multiple Isolated (M) Multiple in Group (G)
Pfinder [49]	I	C	S	S	S
CMU [26]	O	C	M	S	M
LOTS [47]	O	G	S	S	M
MIT [45]	O	C	M	M	M
TI [33]	I	G	M	S	S
SRI [5]	I	G	O	S	M
W4 [20]	O	G	S	B	M,G
KidRooms [1]	I	C	M	S	M
S. Kiosk [22]	I	C	S,O	S	S
M. Mirror [46]	I	C	O	S	M
EasyLiving [2]	I	C	M,O	S	M

Table 2.1: *People tracking systems according to their sensor type and detection and tracking functions [20]. The Reading People Tracker [41, 43] is not shown, but is the tracker used in the EU funded project ADVISOR.*

Figure 2.7: *The frequency of occlusions can be controlled to a certain degree by altering the camera position and angle [44].*

To summarize, these are the main problems encountered in the tracking process:

1. *Single object split into multiple blobs.* Caused by low FG/BG contrast and occlusion.

2. *Multiple objects merged into one blob.* Caused by overlapping shadows and occlusion.

3. *Object disappearing partially or completely for a short time.* Caused by low FG/BG contrast and occlusion with static objects (obstacles).

The consequence is that it is difficult to track individuals robustly using only a single view of the scene. Using multiple cameras can obviously make the system more robust against occlusions, but makes the system significantly more complex. Also, the many thousands of monocular surveillance cameras already deployed around the world (London alone has more than 120,000 cameras) means a single view solution is attractive.

2.3.3 Algorithms

Region Tracking

Velastin describes in [17] a simple region tracking algorithm that the author has implemented with reasonably good results in [30]. The principles are similar to the region tracking algorithm used in the W^4 system [20]. *This algorithm assumes a single, fixed camera.*

The correspondence test is a simple spatial occupancy overlap test between the predicted location of a tracked object and the location of a blob in the motion image. The predicted location of a tracked object is computed using the last recorded position, speed and bounding box.

When searching for correspondence the following cases are possible:

1. A single region in this frame corresponds to a single tracked object
   Correspondence: A person is being tracked.

2. A single region in this frame corresponds to several tracked objects
   Merge: People have joined to form a group.

3. Several regions in this frame corresponds to a single tracked object
   Split: A group has split apart.

4. A region in this frame has no corresponding tracked object
   Enter: A new person or group has entered the area.

5. A tracked object has no corresponding region in the current frame
   Exit: A person or group has left the area.

Let $B_i(t), 0 \leq i < N_t$, be the blobs in the current motion image, and $B_j(t-1), 0 \leq j < N_{t-1}$, be the objects currently being tracked. [17] defines *the matching matrices* as follows:

$$M_{t-1}^{t}(i,j) = Matching\{B_i(t-1), B_j(t)\}$$
$$M_{t}^{t-1}(i,j) = Matching\{B_i(t), B_j(t-1)\}$$

M_t^{t-1} matches from the current frame to the previous, and M_{t-1}^t matches from the previous to the current. The concept of a *matching string* is now introduced:

$$S_{t-1}^{t}(i) = \bigcup_j j \text{ such that } M_{t-1}^{t}(i,j) = 1$$
$$S_{t}^{t-1}(i) = \bigcup_j j \text{ such that } M_{t}^{t-1}(i,j) = 1$$

$S_{t-1}^t(i)$ is a list of all the current blobs that match the tracked object i in the previous frame, $B_i(t-1)$, and $S_t^{t-1}(i)$ is a list of all the tracked objects in the previous frame that matches blob i in the current frame, $B_i(t)$. The cases found above can now be written in mathematical notation:

1. Correspondence:
$$B_i(t-1) \equiv B_j(t) \Longleftrightarrow \begin{cases} S_{t-1}^{t}(i) = j \\ S_{t}^{t-1}(j) = i \end{cases}$$

2. Merge:
$$B_i(t-1) \bigcup B_j(t-1) \equiv B_k(t) \Longleftrightarrow \begin{cases} S_{t-1}^{t}(i) = S_{t-1}^{t}(j) = k \\ S_{t}^{t-1}(k) = i \bigcup j \end{cases}$$

3. Split:
$$B_i(t-1) \equiv B_j(t) \bigcup B_k(t) \Longleftrightarrow \begin{cases} S_{t-1}^{t}(i) = j \bigcup k \\ S_{t}^{t-1}(j) = S_{t}^{t-1}(k) = i \end{cases}$$

4. Enter:
$$B_i(t) \equiv New \Longleftrightarrow \begin{cases} \forall j \; S_{t-1}^{t}(j) \neq i \\ S_{t}^{t-1}(i) = \emptyset \end{cases}$$

5. Exit:
$$B_i(t-1) \equiv Leaves \Longleftrightarrow \begin{cases} S_{t-1}^{t}(i) = \emptyset \\ \forall j \; S_{t}^{t-1}(j) \neq i \end{cases}$$

When the tracking algorithm detects a correspondence between an object and a blob, it updates the object with the new data in the blob, and appends the median coordinates to the trajectory. The median coordinates are not affected as much by the motion of a human's extremities compared to the centroid, and is therefore more suitable for measuring the position of a human in motion [20].

(a) Tracking a single person

(b) Tracking several people

Figure 2.8: *Results from the region tracker implementation described in [30]. Note that the low FG/BG contrast together with the shadow removal algorithm caused the feet of the person in figure 2.8(a) to split from the rest of the body. In this implementation all regions were tracked, so the small region corresponding to the feet has its own history.*

If a merge is detected, the tracker creates a new object, and stores pointers to the old ones (before the merge). The new object represents a group of individuals. See problem number 2 in section 2.3.2. The pointers can be used at a subsequent split to recover the objects (individuals) before the merge. This is necessary to avoid losing the history of the tracked objects up until the merge.

If a split is detected, the tracker tries to find which object (individual) from before the preceeding merge corresponds to each new blob resulting from the split. It matches an object from before the merge with a blob from after the split by choosing the object that, assuming its speed and direction did not change, would have been closest to a certain blob. Additionally, an appearance model can be used to match, such as area, color, histogram [20], etc.

In the enter case, a new object is created and initialized with the new data in the blob.

The exit case could be the result of low FG/BG contrast, a full occlusion, or a partial occlusion where the area of the blob has been reduced enough for it to be removed by the region-based noise removal. See problem number 3 in section 2.3.2. One possible solution to this problem is to keep the tracked object for some time, and try to match it (using some model of appearance) to any new objects entering the scene. If no match is found within a reasonable amount of time, the object can be assumed to have left the area.

Problem number 1 in section 2.3.2 can be delt with by merging small adjacent blobs in the motion image before matching them to tracked objects [42, page 52].

See figure 2.8 for some example results.

Reading People Tracker

Using a region tracker as described above, gives reasonably good results. However, there are still a few problems. In a single sensor system, *occlusions are a big problem*. This means a lot of the time people will appear as groups instead of individuals. If no attempt is made at estimating the position of an individual while in a group, the group trajectory must be used as an estimate. This is usually too imprecise for analyzing the behavior of individuals, especially since in most environments individuals are merged into groups a significant amount of the total time they are in the scene. Groups are typically formed when pedestrians pass each other, or when friends are walking alongside each other.

The Reading People Tracker [42] tries to overcome both of these problems. Firstly, by allowing the use of *multiple sensors*. This will reduce the effect of occlusions, since an individual that is occluded from one camera, is likely to be visible to another covering the same scene. Secondly, the problem of tracking an individual within a group is handled by using two cooperating trackers: one region tracker, and one *active shape tracker*. In addition to being able to track individuals in groups, using two trackers focusing on different features makes the total tracking process more robust [42, page 56].

The region tracker is similar to the one described in the previous section, with a few added features (such as incorporating static regions into the background). The active shape tracker uses an active shape model of 2D pedestrian outlines to detect and track people, even when inside a group. It initializes the contour shapes from the output of the region tracker and also the output of a *head detector*. Offline system calibration is used to initialize thresholds and other configuration variables, and principal component analysis is used to capture the variations in pedestrian shape.

2.4 Individual Behavior Analysis

Behavior that is considered interesting in the case of automated security is intrusion, fighting, vandalism, accidents, and so on. Recognizing behavior is a very challenging task, since it often requires a certain understanding of people's intentions in order to avoid many false alarms. Take running for example: Is this person running from someone, or is he just trying to reach the elevator? And vandalism: is this person standing so close to the wall to spray graffiti, or is he just trying to read the timetable for the bus? And fighting: are these two people embracing or wrestling? There is a large potential for using contextual information here. For instance, if the layout of the scene is known one could determine that the person standing close to a wall could not be reading the timetables, since they are located elsewhere.

In the simplest cases a single frame is enough to recognize "unwanted behavior". [18] calls these simple events *position-based*. Examples of such events are unattended luggage, intrusion into forbidden areas and someone lying on the floor in a public area. The other type of event is one that requires many frames to recognize, and [18] calls this *dynamic-based*. An example of this kind of event is a fight, where you need to analyze the positions,

shapes and possibly other features in successive frames in order to obtain enough information to recognize the event as a fight. This project concentrates on recognition of dynamic-based events.

The ADVISOR [11, 12] is a sophisticated system intended to be used in metro stations. It can recognize many different types of group and individual behavior, such as fighting, vandalism, blocking and jumping over the barrier (neglecting to pay the fare). It uses multiple calibrated cameras with overlapping FOVs to get the input frames. The behavior recognition part of the system can recognize user defined scenarios. A *scenario* is defined in [11] as a combination of states and events. A *state* is defined as a situation describing an actor at time t, or a characteristic of an actor constant during two consecutive instants. An actor is either a person or a group. An *event* is a change of states at two consecutive times. [11] notes that one of the main problems in behavior recognition is the ability to define and reuse methods to recognize specific behaviors, given that the perception of behavior is strongly dependent on the context and camera view point. ADVISOR tries to overcome this problem by using a clever hierarchy of *operators* designed to be flexible and allow many different scenarios to be recognized. An operator is a reusable software component that contains all the knowledge needed to compute the output, which is a set of states and events. The operators compute states and events of varying complexity, with the output of one operator being used as input to the one above. The bottom operator of the hierarchy recognizes basic states, and the top recognizes the scenario of interest (such as a fight). [11] describes how the system can be used to recognize the scenario "a group is fighting".

In [13] a method using a finite state machine is proposed and implemented with automated security in an office environment in mind. The system needs extensive information about the location of entrances and interesting objects and how these objects are used, and can then use this contextual information in a state machine to recognize the actions of any human in the room, assuming the initial scene description covers that action. Noting that skin-like colors are not typically present in office environments, the authors use skin detection, tracking and scene change detection to extract the low-level information, and feed this into the state machine. The output is a textual description of the event that occurred along with the frame that best represents the event (the key frame). The contextual information is used to recognize events, but also to reduce computation. This is achieved by running specific algorithms only at a time and place that makes sense according to the context, such as performing skin detection only in the entrance areas, and scene change detection only when a person is close enough to an interesting object to have an effect on it, and so on.

The approaches from [11, 12] (the ADVISOR) and [13] are quite complex, and recognize complex behavior, or *high-level behavior*. Other approaches are simpler, and recognize less complex behavior. Some of these use a single camera to get the input frames, others use stereo vision, which is basically two cameras at different positions aimed towards the same object or scene (just like the eyes of a human). Some use additional, contextual, information, others do not.

The relatively simple approaches that use a single camera are often intended to be used in environments where CCTV is already being used with a human operator, such as

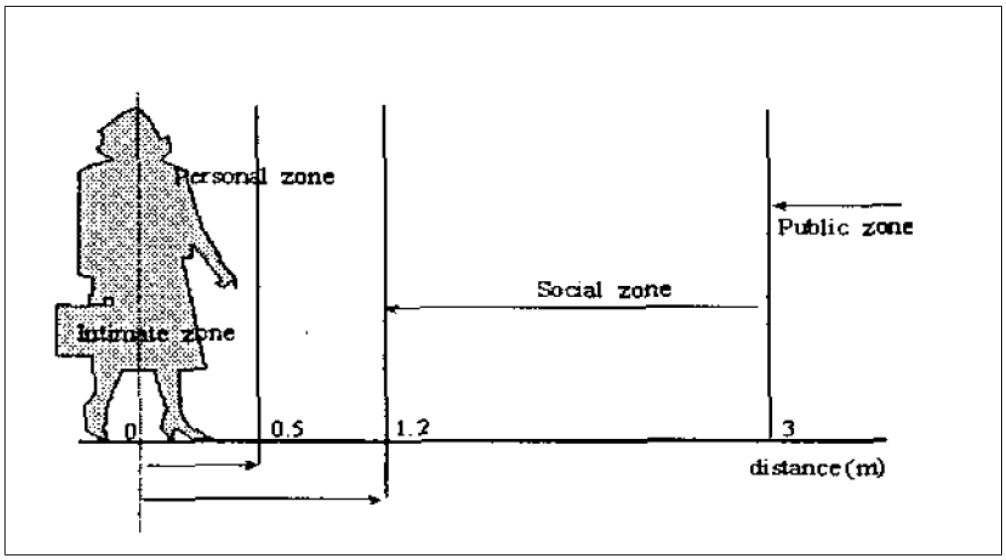


Figure 2.9: *Social zones versus distance [18].*

the London Metro system [18, 28]. These approaches have more limited information than the more complex ones. Consequently they can only recognize relatively *low-level behavior*. The following are examples of features that have been used in these kind of systems [18, 20] to recognize different types of low-level behavior.

Position The 2D blobs can be used to get an approximate positioning of people. More precise information is possible using stereo cameras. The position alone is enough to detect some position-based events such as unattended luggage, intrusion into forbidden areas, falls on tracks, etc. The approximate position can be used to rule out a fight, since no fight is possible if no two blobs are close enough to each other. The distance between blobs should be based on the social zones, and not absolute distance. Figure 2.9 shows the relation between social and absolute distance as used in [18].

Motion Much more information is available if the blobs are tracked. The *velocity* of the centroid or median coordinates could then be used, as a fight is likely to include running at one point or another (fleeing/chasing). One could also use the centroids of groups or individuals *moving to coincide* as a sign of a fight about to happen or already in progress, as noted in [18]. The same article distinguishes between a fight and an attack, where the attack could lead to a fight. Indications of an attack is a newcomer getting *too close* (fig. 2.9) to another blob that was initially static, but is "provoked" into action. It could also be that the "victim" is initially moving, but then stops or changes direction (i.e. tries to move apart) after the newcomer comes close. The *trajectory* can be used to recognize unusual motion patterns [15], such as someone running against the normal flow of motion. The author designed an algorithm to analyze the trajectory in order to recognize chasing/fleeing scenarios [30].

Blob characteristics If tracking is used, fast changes in a *blob's characteristics* can be an indication that something dramatic is happening to this person/group. Several blobs *merging and splitting rapidly* [18] is another obvious sign of something dramatic happening.

People density People density can be used to look for possibly dangerous situations. If the environment is not crowded with people and yet someone is standing inside someone else's personal zone, it could be an indication of an attack. People standing inside each other's intimate zone are either embracing or there is certainly something suspect going on (unless it is an extremely crowded environment, such as an overfilled elevator). Due to occlusion this situation will result in one big blob of abnormal shape.

Body parts The W4 system described in [20] can segment body parts, and this could be very useful in recognizing fights. If two individuals have their arms extended towards each other it could indicate someone is boxing or wrestling. If an individual's extremities are changing shape and position rapidly, it could indicate someone hitting or kicking each other. The relative velocity of body parts is an important factor in how a human interprets behavior [40], and it is certainly likely that some of the involved individuals' extremities have a high relative velocity when fighting. W4 can also in some cases estimate the body posture, which could be used to find people lying on the ground, perhaps passed out.

Carried objects W4 can also recognize whether an individual is carrying an object. This could be used to recognize theft and muggings, and could be combined with several of the other features for interesting results (body parts, for instance).

Combinations of the above Most actual work involves the use of several of the above mentioned features in some scheme. [31] defines the concept of *global agitation* of a group in order to recognize violent behavior. Global agitation is defined in terms of *internal* and *external* agitation. The visual features used to infer global agitation are:

- *Internal agitation*:
  - *Evolution of the density* of the group. The density is the ratio of the number of pixels that compose the group and the size of the group.
  - *Movement of the centroid*.
- *External agitation*:
  - *Evolution of the size (3D height * 3D width)* of the group.
  - *Acceleration* of the group.
  - *Evolution of the trajectory (orientation)* of the group.

Observing that the internal and external agitation are time dependent behaviors, they are inferred using a Recurrent Bayesian Network.

With the limited amount of information available when not using contextual information, one should expect a relatively large amount of false alarms, especially when trying to recognize complex behavior such as fights. In many public areas people run to get to the train before it leaves the platform, wave their arms to get the attention of a friend, etc, without being involved in a fight. The purpose of this kind of surveillance system is therefore only to attract the attention of an operator, and aid in the interpretation of the scene by annotating the video frames.

Chapter 3

Recognizing Unwanted Behavior

This chapter discusses what behavior is unwanted in the context of this thesis, and reduces the abstract behavior to a set of concrete behavior descriptions that can be recognized. Then algorithms for recognizing chasing/fleeing scenarios and people lying down are presented.

The chapter contains the following sections:

- Abstract Behavior Definitions
- Behavior Reduction: Concrete Behavior Descriptions
- Unrecognized Behaviors
- Recognized Behaviors
- Alternative Approaches to Behavior Recognition
- Recognizing <Chasing>
- Recognizing <Not Standing>

3.1 Abstract Behavior Definitions

There are many types of behavior that are unwanted in public areas. Some examples are stealing, vandalism, fighting, drug distribution and abuse, not paying the subway fare, and so on.

The focus of this thesis is on recognizing behavior that could cause physical or psychological injury or discomfort to anyone present. This should include recognizing behavior resulting from accidents, and not be restricted to criminal and antisocial behavior. The focus is not on solving crimes, but on preventing injuries.

The following categories can then be defined:

(I) **Behavior resulting from injuries**
Accidents, violence and use of drugs and alchohol lead to injuries, in the form of temporary or permanent damage to motor skills. Examples: intoxicated person, falling down a stair, victim of random violence, etc. The actual event that caused the injury may have happened elsewhere, or it may not have been picked up by any surveillance camera in that area.

(V) **Violent behavior: physical discomfort**
Violent behavior is characterized by sudden motion with great force, or the use of weapons. Examples: fighting, chasing, firing a gun, etc.

(T) **Threatening behavior: psychological discomfort**
Threatening behavior is a very broad term, but is typically characterized by people avoiding, or withdrawing from, a certain invidual. Examples: shouting, pointing at bystanders, carrying a weapon, following someone (without running), etc.

3.2 Behavior Reduction: Concrete Behavior Descriptions

A set of concrete behaviors which together cover the abstract definitions must be described. The following is a set of descriptions that cover a large subset of the behaviors in the categories defined above. Note that the descriptions are not meant to be definitions, merely low-level descriptions.

3.2.1 <Not Standing>

Description

The behavior <Not Standing> is said to be exhibited by an individual who is sitting, lying or crawling on the floor or ground.

Rationale

Arguably the most obvious unwanted behavior in public areas is someone sitting, lying or crawling directly on the floor. People usually sit on a chair, bench, or some suitable, elevated structure, and not directly on the floor. A person sitting, lying or crawling on the floor is likely a victim of an accident or violence.

Other possibilities exist, of course, such as a homeless person sitting on the floor, or someone crawling or lying on the floor while searching for a lost item. Regardless of the cause, this behavior should be monitored closely, as the person is likely to need assistance of some sort.

Recognizing <Not Standing> ensures that many of the behaviors under category (I) in section 3.1 are covered. Since the behaviors under category (V) often results in someone

falling down at some point in time, <Not Standing> will also cover some of the behaviors in that category (although it would be better to recognize the behavior causing someone to fall down). Those that remain are the injuries that are light enough for the person to remain standing more or less upright. There might still be visible symptoms, such as an abnormal posture, staggering, limping, bleeding, etc, but the person is not lying or sitting on the floor.

3.2.2 <Fighting>

Description

The behavior <Fighting> is said to be exhibited by a group of individuals who are in close proximity to each other, and are visually agitated over some period of time. This corresponds to the definition of *violent behavior* in [31].

Rationale

The behavior <Fighting> covers some situations resulting from behavior in category (V) in section 3.1. Specifically, those that involve two or more people violently quarreling or wrestling over some period of time. A short but brutal attack will therefore go undetected using this concrete behavior description.

3.2.3 <Attacking>

Description

The behavior <Attacking> is said to be exhibited by some individual A, if, for some other individual B, the following sequence of events takes place:

1. A and B are initially standing, walking or running at some distance to each other
2. A then changes direction, and makes a sudden motion towards B, hitting or coming very close to hitting B
3. B immediately recoils or falls down

Rationale

Recognizing <Attacking> ensures that not only wrestling, but also sudden violence of short duration is covered. An example of this concrete behavior is a random act of violence, such as the recent trend called ”happy slapping” [48]. This form of violence is often of short duration, and leaves the victim dazed and confused, or perhaps lying on the floor. A person lying on the floor will be covered by <Not Standing>, but a sudden punch or push that causes the victim to recoil, but not fall down, will only be covered by <Attacking>.

3.2.4 <Chasing>

Description

The behavior <Chasing> is said to be exhibited by some individual A, if, for some other individual B, all the following conditions are simultaneously true:

- both A and B are running
- A is heading roughly in the direction of B for some period of time T
- A never catches up to B during T

This corresponds to the definition in [30, page 23].

Rationale

As noted in [30], these conditions do not take obstacles into consideration. Information about the locations of obstacles could be used to create a more general definition. Other contextual information could be used to reduce the number of false alarms, such as the locations of train platforms, airport gates, schedules, etc.

The concrete behavior <Fighting> covers violent behavior in the form of wrestling, <Attacking> covers sudden violence of short duration, and <Chasing> covers violence where there is no contact at all. Together these should cover a large portion of the behaviors under the category (V) in section 3.1. The ones remaining are the ones that involve the use of guns and ranged weapons.

3.2.5 <Pointing>

Description

<Pointing> is said to be exhibited by an individual with one or both arms outstretched towards some other individual for a some period of time.

Rationale

Recognizing <Pointing> ensures that someone wielding a gun or other ranged weapon will be covered. <Pointing> is often also an indication of threatening behavior, even when no weapon is wielded. <Pointing> will thus cover behavior involving ranged weapons, belonging to category (V) in section 3.1, and some threatening behaviors belonging to category (T).

3.2.6 <Repulsing>

Description

<Repulsing> is said to be exhibited by an individual who seem to have a repulsive force on other people in the area.

Rationale

A <Repulsing> behavior is indicated by people changing direction when coming to close to the repulsive individual. See figure 2.1 (a) and (b) for two examples of this. Recognizing <Repulsing> ensures that a large portion of the behaviors in the category (T) in section 3.1 is covered.

3.2.7 <Intruding>

Description

<Intruding> is said to be exhibited by an individual who is located in a forbidden area.

Rationale

In the context of this thesis, ”forbidden” is actually ”dangerous”. <Intruding> is indicated by someone being in an area that has been marked as ”forbidden”, perhaps with some conditions, such as the time of day or the position of some other object. The train tracks in a subway station is a good example of an area that should be marked ”forbidden”. <Intruding> is actually not a behavior, but rather a *state*: the state of being in the wrong place.

3.3 Unrecognized Behaviors

The above set of concrete behavior descriptions is an attempt at covering all the ”interesting” behaviors, that is: unwanted behavior as defined in section 3.1. Obviously, this attempt must fail, because there are unwanted behaviors that are simply not possible to recognize using only vision.

Some of these behaviors are impossible to recognize due to technical reasons, such as the low quality of the images from the surveillance camera, or the distance of the subjects from the camera. This could for instance be facial expressions, use of fingers, and other behaviors that produce only small visual clues.

Other behaviors produce no visual clues at all. Talking is an example of this. It is perfectly acceptable for people to be talking, but only if what is said is socially acceptable. Constant or loud cursing, threatening, and screaming are examples of inappropriate and unwanted behavior that under some circumstances are impossible to recognize using only vision. To recognize such behavior robustly, other sensors such as microphones would have to be used.

Antisocial behavior and some scenarios that could be dangerous have intentionally been left out. *Loitering* and *vandalism* are examples of the former, *unattended luggage* of the latter. This is due to the focus of this thesis (section 3.1).

3.4 Recognized Behaviors

Due to the scope and time constraints of this work, only a few of the behaviors can be thoroughly discussed in this thesis. The other behaviors described in section 3.2 are candidates for future research.

Firstly, an algorithm for recognizing <Chasing>, originally developed by the author in [30], will be extended to cope with arbitrary camera positions and rotations. This is interesting because the camera configurations varies between different scenes and environments. Some scenes have ceiling mounted cameras, others have cameras mounted at a more oblique angle. Secondly, methods for recognizing <Not Standing> will be investigated, and an algorithm that makes use of cast shadows is designed. <Not Standing> is arguably the most important of the unwanted behaviors in the context of preventing injury and discomfort, as many dangerous situations at some point in time includes someone not standing.

3.5 Alternative Approaches to Behavior Recognition

After having extracted the interesting features from the frames of a video, the question becomes: how can these features be used to recognize the behavior of interest? This depends of course on the behavior. Some can be recognized simply by looking at a single feature in a single frame. An example is what Velastin [18] calls *position-based events*, for instance <Intruding>. Other behaviors are much more complex, and finding an explicit algorithm for recognizing such a behavior may be very difficult. An example of such a behavior is <Fighting>.

Behavior recognition is basically pattern recognition. Often when dealing with the problem of recognizing a complex pattern, *machine learning methods* are applied. Examples of machine learning methods are Artificial Neural Networks (ANN), Hidden Markov Models (HMM), Bayesian Networks (BN) and Case-Based Reasoning (CBR). An interesting discussion of the various issues when using BN, HMM and ANN can be found in [31]. CBR is described in [3].

The advantage of learning methods are of course the ability to adapt and learn very complex behaviors, and also to make use of prior knowledge. There are a few disadvantages, however. Firstly, they need lots of examples. Secondly, they suffer from the problem of the *temporal dimension* [31]. In other words, the time-scale of the behavior is fixed, unlike in the real world (move slow or fast, a human will still recognize the behavior). The algorithm developed in [31] is designed to be independent of the time-scale, and shows great potential for recognizing complex behaviors such as <Fighting>.

The method chosen here is similar to that of Velastin [18], that is a *structural approach*. This means the algorithms will be reasoning directly on structural information, such as positions and distances, instead of giving these as inputs to some probabilistic or neural network. This is chiefly because <Chasing> and <Not Standing> are relatively simple behaviors; the use of learning methods does not seem necessary. Also, the scope of this thesis,

which is the *whole* process of behavior recognition, does not allow for the time consuming process of gathering examples and teaching a network. The use of learning methods is an interesting direction for future research, particularly for recognizing <Fighting> and <Attacking>.

3.6 Recognizing <Chasing>

This section describes the algorithm developed in [30] for recognizing <Chasing> under certain assumptions regarding the camera angle. The algorithm is then modified to remove the need for any assumptions regarding the camera angle. The final algorithm presented is independent of the extrinsic camera parameters.

3.6.1 Feature Selection

The first thing to do is to find the characteristic features of the behavior <Chasing>. There may be several people involved, but there must be at least one *offender* and at least one *victim*. Thus it should be sufficient to compare each person to every other person in the scene, because if there is a chase in progress we must at some point compare one of the offenders with one of the victims.

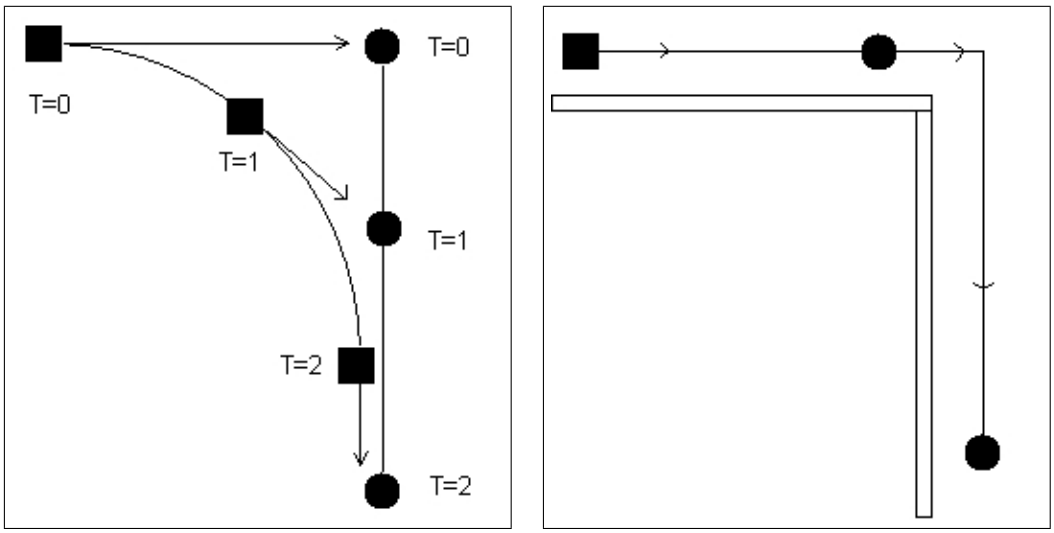
The most obvious characteristic of a chase is that both the offender and victim will be *running*. Another characteristic is that they are not likely to be running very close to each other. Put another way: if the offender has already caught up with the victim, why are they still running? There could be many reasons, including an incidental passing, a group of people running towards the same destination, and so on, but a real chase would seem unlikely.

Figure 3.1(a) is an attempt at illustrating a chase in an open space. At some point in time the offender and victim are at the positions marked T=0. Then as the chase progresses they move to the positions marked T=1 and T=2. The arrow starting at the offender illustrates the direction the offender is heading at. Notice that the offender, as is natural when trying to catch someone, is always heading towards the position of the victim at the time.

This is assuming the offender has a clear path towards the victim. If there are obstructions along the path, such as walls, furniture, and such, both offender and victim will have to avoid these, making the analysis of the trajectory less straightforward. Say, for instance, that the offender is chasing the victim along a straight path for a while, and then the victim turns sharply around a corner (figure 3.1(b)). The offender will continue towards the corner, and until it's reached will be heading in a completely different direction than would be natural if there was no obstacle. *It will continue to be an assumption that there are no large obstacles in the scene under surveillance.*

Based on these observations, some basic requirements for a situation to be recognized as a chase can be listed:

1. Both offender and victim are running.



(a) A chase in an open space (b) A chase around a corner

Figure 3.1: *The progression of a chase as seen from above [30]. The offender is the square, the victim is the circle.*

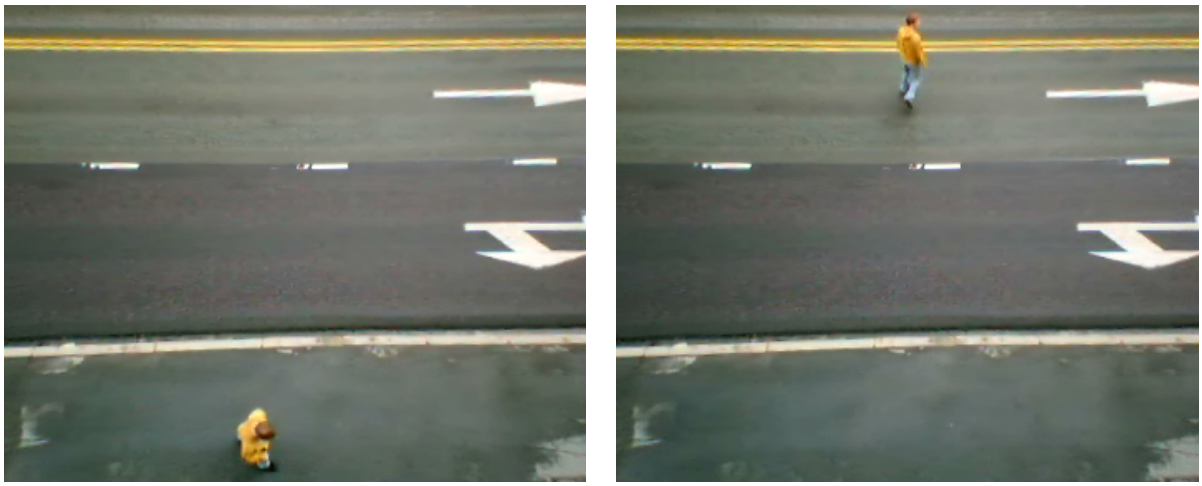
2. The offender is heading roughly in the direction of the victim.

3. The offender never catches up with the victim. That is; the minimum distance between them must be longer than a certain threshold value (arm's length).

Given the above requirements, the only feature needed to recognize <Chasing> is actually the *position*. The trajectory is formed by tracking the positions of individuals over time, and from the trajectory the speed and direction can easily be calculated. The distance between two individuals at any point in time can of course also be calculated from the trajectory. Assuming the floor or ground in the area under surveillance is approximately *planar*, as is often the case, then positions of people in the scene can be approximated by positions in the two dimensional floor plane.

However, a person is not a single point. How then should the position be defined? In [30], the author used the *centroid* of a 2D blob in the image plane as the position of the corresponding individual. This was an acceptable approximation, since the camera was assumed to be looking directly down at the scene from some height (and perspective distortion was ignored). With no assumptions about camera parameters, this is not a useful approximation. See figure 3.2 for an example of a scene where this would give unacceptable results. One possibility is to relate the position of an individual with the world position of a specific body part, for instance the head or the feet, instead of some image feature like the centroid. This would obviously remove any dependence on camera parameters.

Considering the above observations, a natural definition for the position of an individual would be the point of contact between the individual and the floor. It is not only natural, but as will be seen in section 3.6.2 it also simplifies the process of going from image plane coordinates to world (floor plane) coordinates. This is quite important, as the image plane



(a) Projection rays perpendicular to ground plane (b) Projection rays at oblique angle to ground plane

Figure 3.2: *The effect of extrinsic and intrinsic camera parameters on visual blob features. In image 3.2(a), the projection rays are perpendicular to the ground plane where the individual is standing, and so the person appears as if seen directly from above. In the next image the projection rays are at an oblique angle to the ground plane where the individual is standing, and so the person appears as if seen from the side.*

coordinates are not useful in themselves (again, see section 3.6.2). Assuming the individual is standing, the contact point would be the position of its feet.

In conclusion, the feature selected for recognizing <Chasing> is the point of contact between each individual and the floor, or the world *feet position*.

3.6.2 Feature Extraction

How can the position of the feet of an individual relative to the floor plane be computed? First, the feet must be located in the image. Then, the 2D image coordinates can be mapped onto the floor plane using a mapping called a homography.

How to find the feet in the image

Consider figure 3.2 and 3.3. As can be seen, it is not always the case that the bottom part of the foreground blob corresponds to the feet of a person. This is highly dependent on the camera parameters, and most importantly the camera angle. Many surveillance cameras are ceiling mounted, and the view is similar to that of figure 3.3. In this configuration, the position of the feet of an individual could be anywhere inside the corresponding foreground blob, depending on where the individual is in the field of view. The feet position of person B is at the bottom of the blob, but the feet of person C is at the top of the blob. The feet of person A is at the blob centroid.

Observe, however, that the feet of a person standing upright are close to a line going

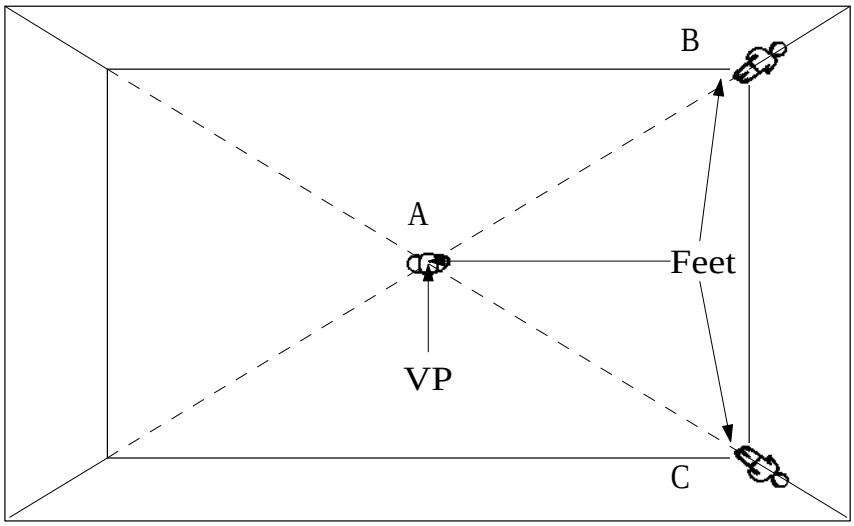


Figure 3.3: *Estimating the feet positions in the image plane. A, B and C are three individuals* standing *at different positions in a square room. A is standing in the center of the room, and B and C in the top right and bottom right corner respectively. The camera is looking down at the center of the room. VP is the* vertical vanishing point*, and the feet positions are marked.*

through the centroid or median coordinates of the blob and the vertical vanishing point (assuming the floor is the reference plane). See figure 3.4 for an example from a real scene. This is generally not the case for a person lying on the floor, or standing but leaning in some direction (not standing at a 90 degree angle to the floor plane). People usually lean somewhat forward when running straight ahead, and perhaps additionally leaning into a turn. How much a person is leaning, depends on how fast the person is running. The algorithm that will be developed in this section will disregard these sources of error, for the following reasons:

1. Individuals who are not running (but for instance lying on the floor), will not be analyzed by the <Chasing> recognition algorithm, and so the estimated feet positions will not be used.

2. It is assumed that leaning while running gives only a small error in the estimate of feet positions. An analysis of this error is beyond the scope and time constraints of this thesis.

Figure 3.5 shows a pseudo code of the feet estimation algorithm. It works as follows: if the vertical vanishing point is inside the blob, than that is used for the feet position estimate. If not, then the point on the contour with the minimum distance from the vertical vanishing point is found. The feet position estimate is then the projection of this point onto the line through the median coordinates of the blob and the vertical vanishing point.

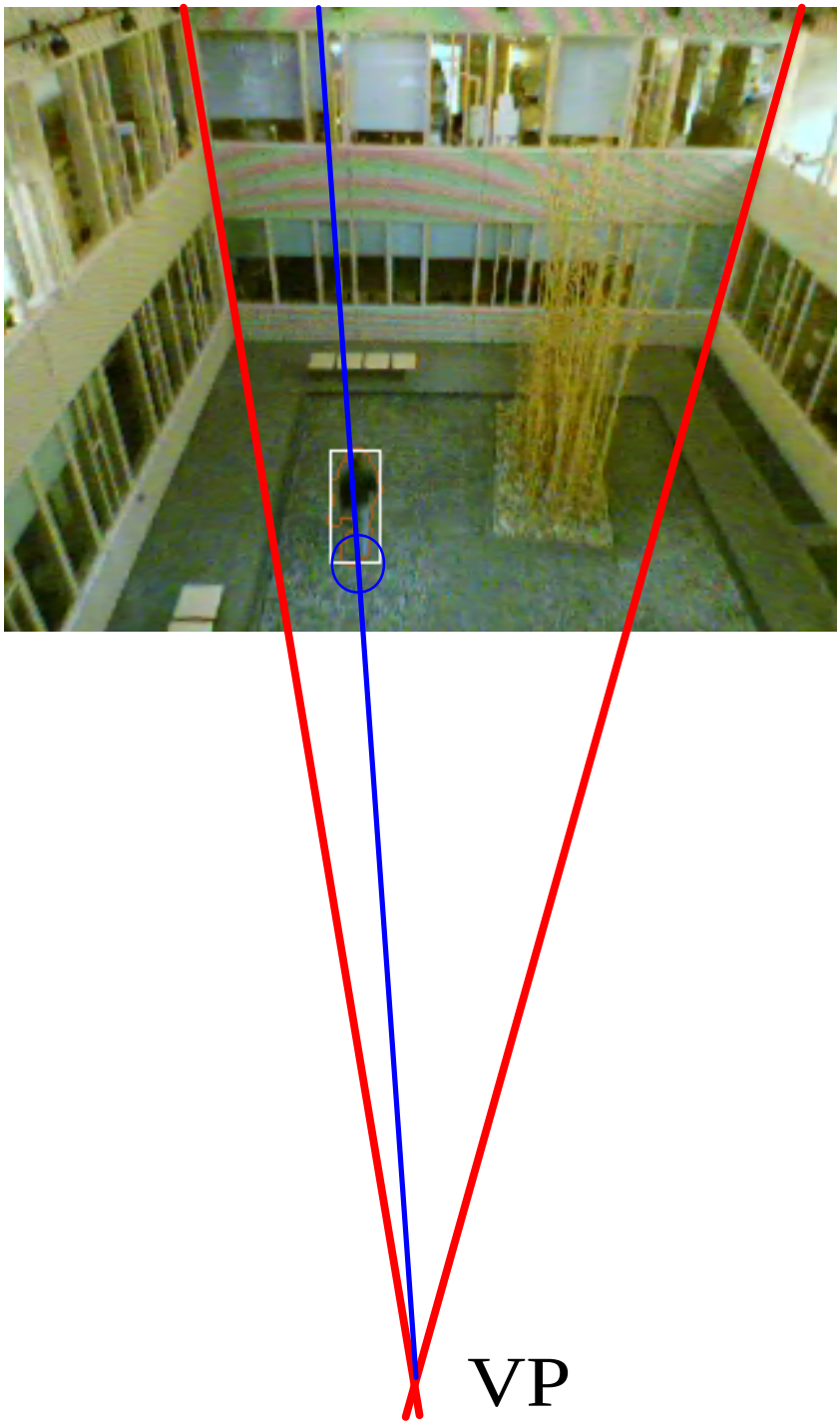


Figure 3.4: *An example of how to find the vertical vanshing point (VP). The red lines are images of lines that are both perpendicular to the reference plane(the floor), and are thus parallel in the scene. Such lines can be used to find the vertical vanishing point. Note that the vanishing points (vertical and horizontal) often fall outside the physical image. The blue line goes through the centroid of the blob and the VP. The estimated feet position is at the center of the blue circle.*

```
vp = vertical vanishing point

if vp inside blob:
    feetpos = vp
else
    mindist = inf

    For each point p on the contour of a blob:

        If dist(p,vp) < mindist:
            mindist = dist(p,vp)
            minpos = p

    m = median coordinates of blob
    m_vp = vp - m
    m_p = minpos - m

    feetpos = m + m_vp((m_vp * m_p)/|m_vp|)
```

Figure 3.5: *Algorithm: feet position estimation.*

The median coordinates are used because they are less affected by the movements of the extremities (hands and feet) compared to the centroid. An example of the result of this algorithm on an image from a real scene can be seen in figure 3.4. If it were to be used on the image in figure 3.3, the results would be quite accurate. The feet of A would be estimated to be the median coordinates. B and C would have estimates very close to the actual feet positions. In the case of someone's legs being widely separated, which happens periodically when running if seen from the side, the estimated feet position would be somewhere in between the legs, of course lying on the line going through the median coordinates and the vertical vanishing point.

This algorithm is independent of camera angle and rotations, which is an important property given the large number of surveillance camera configurations in use. It would also be possible to use this algorithm with a moving camera, for instance a surveillance camera that rotates using a servo. This is assuming the vertical vanishing point is updated as it rotates (and also the homography described in next section).

How to find the vertical vanishing point

A vanishing point is where images of parallel edges of objects in the scene intersect. The vertical vanishing point is where images of edges that are perpendicular to a scene reference plane intersect. In man made environments it is usually easy to find straight edges that are

(a) Perspective distortion (b) Plane-to-plane correspondence

Figure 3.6: *Figure 3.6(a) illustrates the effect of perspective distortion on the image of a typical indoor scene. The corners of the four-sided polygon marked in red in figure 3.6(b) is an example of image plane to world plane correspondance points that could be used to calculate a plane-to-plane homography [6]. The coordinates in the image plane are readily available, and the coordinates in the floor plane can be measured manually. Note that it is always best to cover as large an area as possible, to avoid unnecessary extrapolation when applying the homography matrix to get coordinates on the floor plane.*

perpendicular to the reference plane. In scenes where no such edges can be found, some calibration object could be used. A minimum of two such lines are needed. If more are available, a *maximum likelihood estimate* algorithm is employed to estimate the point [25].

How to find the feet in the scene

It is not enough to know the position of the feet of an individual in the image. Consider figure 3.6. The blob corresponding to a person running towards or away from the camera will have a relatively low image speed when the person is far from the camera, even if the actual speed of the individual relative to the floor is high. If the camera was aimed almost horizontally, the person would appear to be standing still and shrinking. Thus the speed of blobs in the image plane is usually not a good approximation of the speed of the corresponding individuals relative to the floor. To get the speed of an individual relative to the floor, the estimated feet positions must of course also be relative to the floor plane.

To go from 2D image plane coordinates to 2D (floor plane) or 3D world coordinates, some form of camera calibration is needed. Traditionally, camera calibration involves using multiple views of the same scene, often using a custom calibration object of known geometry [16, 50]. However, this is often not practical in surveillance tasks, since it would take some manual work on site. Since there are often *very* many sites (just think of how many surveillance cameras that exist in London alone, more than 120,000!), this would be impractical.

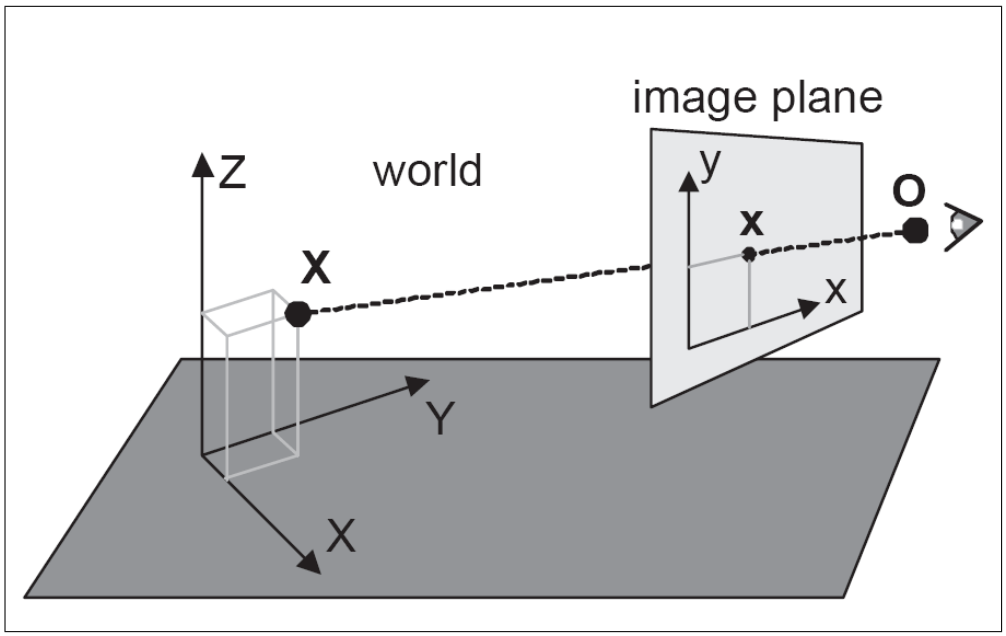


Figure 3.7: *The central projection, or pinhole, camera model [8].*

Criminisi, Reid and Zisserman of the University of Oxford, UK, have developed a method for *minimal camera calibration*, which requires only a single view of the scene [6, 7, 8]. They call this method *Single View Metrology*. Single view metrology allows for two types of measurements: (*i*) lengths of segments on planar surfaces, and (*ii*) distances of points from planes. The first is obviously the type of measurement needed here, in order to get the relative positions of people on the floor plane. The second type of measurement can be used to measure the height of people, which can be used in the recognition of <Not Standing> (see section 3.7).

As the only type of measurement needed to recognize <Chasing> is (*i*), that will be explained here. The reader is referred to the PhD dissertation of Criminisi [7] for a comprehensive description of the algorithms and possible applications. The following explanation will follow section 2.1 and the appendix of [8].

Single View Metrology

The camera model employed is central projection (see figure 3.7). As noted in [8], effects such as radial distortion, which corrupt the central projection model, can generally be removed [14].

The problem that must be solved is how to go from known image plane coordinates to unknown floor plane coordinates. Of course, it does not have to be the floor, it can be any planar surface in the scene. What is needed is a mapping from one 2D coordinate system to another 2D coordinate system. Given an image of a planar surface, points on the image plane can be mapped into corresponding points in the world plane by means of a projective transformation called *homography* [21] (see figure 3.8).

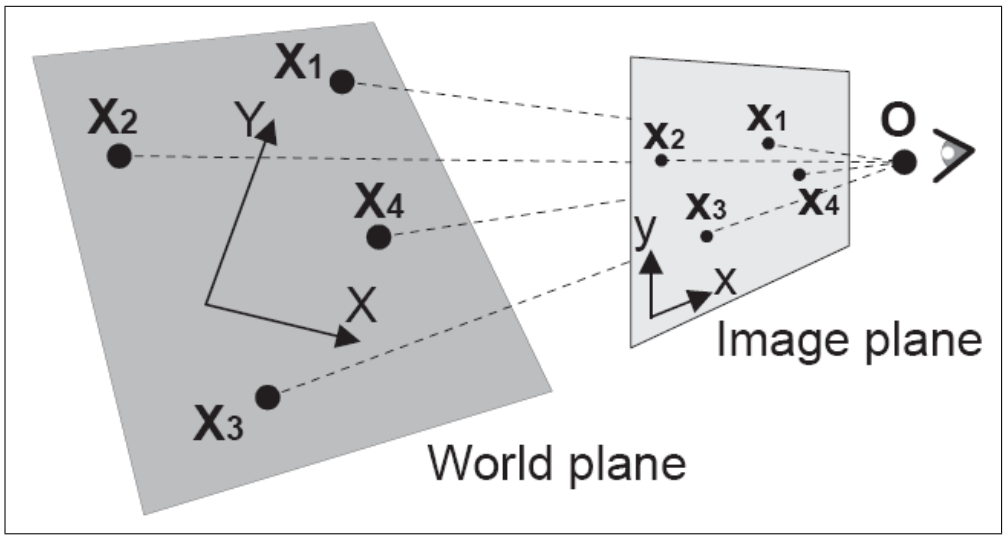


Figure 3.8: *Mapping from image plane to world plane [8].*

Points in one plane are mapped into the corresponding points in the other plane as follows:

$$X = Hx \tag{3.1}$$

where x is an image point, X is the corresponding point on the world plane (both expressed in homogeneous coordinates) and H is the 3×3 matrix representing the homography transformation. x is the feet position in the image plane, and is thus known. H is not known, however, and must be found.

The homography can be estimated using a set of known image-world correspondences. From 3.1 each image-to-world point correspondence provides two equations which are linear in the elements of the matrix H. They are:

$$h_{11}x + h_{12}y + h_{13} = h_{31}xX + h_{32}yX + h_{33}X$$
$$h_{21}x + h_{22}y + h_{23} = h_{31}xY + h_{32}yY + h_{33}Y$$

For n correspondences we obtain a system of $2n$ equations in eight unknowns. If $n = 4$ (as in fig. 3.6(b)) then an exact solution is obtained. Otherwise, if $n > 4$, the matrix is over-determined, and H is estimated by a suitable minimization scheme.

The solution is obtained using Singular Value Decomposition (SVD). Writing H as a 9-vector $h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^\top$ the homogeneous equation 3.1 for n points become $Ah = 0$, with A the $2n\times9$ matrix:

$$A = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 & -X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 & -Y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nX_n & -y_nX_n & -X_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_nY_n & -y_nY_n & -Y_n \end{pmatrix}$$

The solution h is a unit eigenvector of the matrix $A^\top A$ and $\lambda = h^\top A^\top A h$ is the corresponding eigenvalue. If $A = USV^\top$, then the columns of the matrix V contains the eigenvectors of $A^\top A$, and the elements on the diagonal of the matrix S are the corresponding eigenvalues. Only the eigenvector corresponding to the minimum eigenvalue should be considered. Appendix C is a listing of some Matlab code for calculating H. The singular value decomposition is calculated using the Matlab function svd: `[U,S,V] = svd(A)`.

As mentioned, at least four sets of correspondences must be found, giving at least eight rows in the matrix A. These points should be chosen so that as much as possible of the world plane is inside the area created by connecting the points (see figure 3.6(b)). Using equation 3.1 on a point outside this area will result in extrapolation, which is less accurate than interpolation.

After the homography matrix H has been found, the floor plane coordinates corresponding to the image plane coordinates of feet positions can be found by equation 3.1. Note that usually the image plane coordinate system is a right-handed system (y = 0 at the top of the image), whereas the world plane is often a left-handed system. If that is the case, then the image coordinates should be converted to a left-handed system (for instance by changing the sign of the y-coordinates) before applying equation 3.1.

3.6.3 Algorithm

Figure 3.9 is a pseudo code of an algorithm that is able to recognize <Chasing>. This is the same one used in [30], but using floor plane coordinates instead of image plane coordinates. This algorithm no longer depends on the camera being ceiling mounted. In fact, the camera can also be rotated around any axis, since the vertical vanishing point is used to find the feet positions. The following assumptions are necessary:

- There are no large obstacles in the scene. This would cause problems in the trajectory analysis, as explained in section 3.6.1.
- The people-density in the scene is relatively low. Occlusions will obviously cause problems in estimating the feet positions, as well as the motion detection and tracking (as in any single-sensor surveillance system). It is still advantegous to use ceiling mounted cameras, as this will minimize the frequency of occlusions.
- Leaning while running causes only a small error in the estimate of the feet position in the image. The size of this error should be investigated in future work.

3.7 Recognizing <Not Standing>

Recognizing <Not Standing> would be a simple task if certain assumptions on camera angle could be made. If the camera angle could be assumed to be within some range, a simple check on the width/height ratio of the bounding box of an individual would probably perform well. When no assumptions are made about camera parameters, recognizing <Not Standing> becomes less straightforward.

```
alarm = false, offender = nil, victim = nil

For each Person p1, p2, such that p1 != p2:

    If mspeed(p1) < chaseMinSpeed or mspeed(p2) < chaseMinSpeed:
        continue

    minDist = Inf, p1diff = 0, p2diff = 0

    For t = lastFrame; t >= (lastFrame-chaseFrames); t--:

        If |position(p1,t) - position(p2,t)| < minDist:
            minDist = |position(p1,t) - position(p2,t)|

        p1diff += |angle(heading(p1,t)) -
                   angle(position(p2,t) - position(p1,t))|

        p2diff += |angle(heading(p2,t)) -
                   angle(position(p1,t) - position(p2,t))|

    p1diffnorm = (p1diff / chaseFrames) * distFactor(p1,p2)
    p2diffnorm = (p2diff / chaseFrames) * distFactor(p1,p2)

    If (p1diffnorm < chaseMaxDiff or p2diffnorm < chaseMaxDiff)
    and (minDist > chaseMinDist):

        chase = true

        If p1diffnorm < p2diffnorm:
            offender = p1
            victim = p2
        Else:
            offender = p2
            victim = p1

        break
```

Figure 3.9: *Algorithm: recognizing <Chasing> [30]. The variable and function names are hopefully self-explanatory.* ***position*** *and* ***heading*** *are assumed to return a 3D vector.* ***angle*** *returns the angle of a vector in the range [0, 2pi).* ***mspeed*** *returns the mean speed of a person during the last* ***chaseFrames*** *frames.*

3.7.1 Feature Selection

Selecting a set of visual features from which a specific behavior, in this case <Not Standing>, can be inferred, is not as simple as it first might seem. Firstly, the features must be characteristic for that behavior. Secondly, as many characteristic features as possible should be used, in order to increase the accuracy of the inference process. Thirdly, all these features should be conditionally independent of each other, since a feature that is conditionally dependent on another feature in the set will not bring any new information to the table. Additionally, the complexity and computational cost of the feature extraction process must also be considered.

Height

Arguably the most obvious characteristic feature of an individual who is not standing, is that *the distance from the ground and up to the highest point on the individual will be smaller than when standing*. More specific, the distance from the ground and up to the top of the head will be smaller when lying or sitting than when standing.

Body Parts

Another characteristic is that *other body parts apart from the feet are touching the ground*. When sitting, the bottom would obviously touch the ground, when crawling the knees and hands, and when lying the back or chest. However, detecting which body parts touch the ground obviously requires first detecting the locations of the body parts in the image. This is extremely difficult in the images typically produced by surveillance cameras, due to limited resolution, occlusions and other problems discussed in section 2.2.2.

Speed

Another very obvious characteristic is that *the speed of a lying or sitting individual is likely to be very low*, regardless of direction. Speed here refers to world space. Speed in the image plane (the speed of blobs), as mentioned in section 3.6.2, is generally not an accurate approximation to the world speed.

Blob Features

The above described features require knowledge of 3D (world) positions and dimensions of people and their body parts. There are more easily accessible features that are affected by the behavior <Not Standing>, however. The following 2D image features will be affected in some way:

- area: the number of pixels in the blob
- size: the size of the bounding rectangle
- width and height

- ratio of width to height
- shape, contour, silhouette
- blob speed

Exactly how each feature is affected depends on the extrinsic and intrinsic parameters of the camera, and also the location of the specific individual in the field of view. An example is the effect of camera angle on the imaged height of someone falling down. If the camera is mounted horizontally or at an oblique angle, the height will decrease when the person is falling down. However, if the camera is mounted directly overhead, the imaged height could either increase or decrease, depending on the direction the person is falling relative to the orientation of the camera, and also the location of the individual in the field of view (see figure 3.3). Thus the effect on a visual feature is different depending on the camera angle and on the location of the individual in the field of view.

The image speed of a blob corresponding to an individual lying or sitting on the floor will be low. However, the converse is not true: a blob with a low image speed does not necessarily correspond to a lying or sitting individual. Consider figure 3.6. The blob corresponding to a person running towards or away from the camera will have a relatively low image speed when the person is far from the camera, even if the actual speed of the individual relative to the floor is high. This is due to the perspective distortion. Thus the speed of blobs in the image plane is usually not a good approximation of the speed of the corresponding individuals relative to the floor. Instead, camera calibration techniques such as those in [6, 7, 16, 21] must be used to get 3D (or 2D floor plane) positions, as described in section 3.6.2.

Cast Shadows

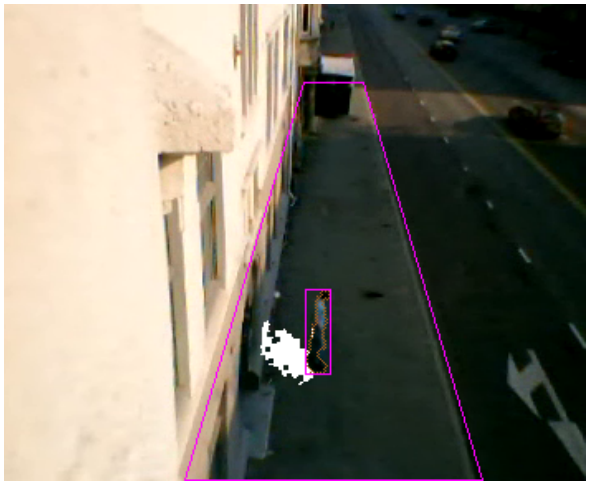
Cast shadows, as mentioned in section 2.2.2, are usually considered to be merely noise, and steps are taken to detect and remove them from the segmented foreground (section 2.2.3). However, cast shadows can also be a valuable source of information, assuming they can be detected. See for instance [36].

Consider figure 3.10. If the area is illuminated by a light source of sufficient intensity, *a standing person will cast a relatively large shadow* when compared to the area of the blob itself. Conversely: a person lying on the floor will cast a relatively small shadow, due to the decreased height. If relative positions of individuals and light sources and extrinsic and intrinsic camera parameters are known, it is also possible to use cast shadows for getting 3D positions and dimensions.

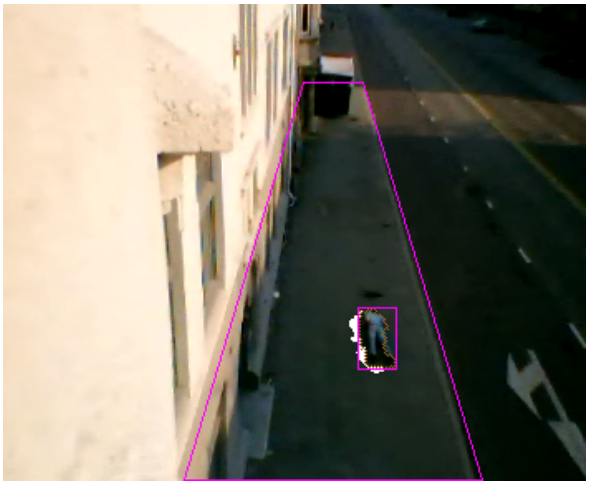
The relative area of cast shadows has a few advantages over the other features extracted directly from the image. One is that it is not affected by camera angle, orientation or focal length. The absolute size is affected of course, but the area of the cast shadow relative to the area of the corresponding foreground blob is not. Another advantage is that a person's extremities will not disturb the result. Whether a person is lying on his back with his arms outstretched, or in a fetal presentation, the result will be the same: a very small cast

(a) Standing upright in direct sunlight

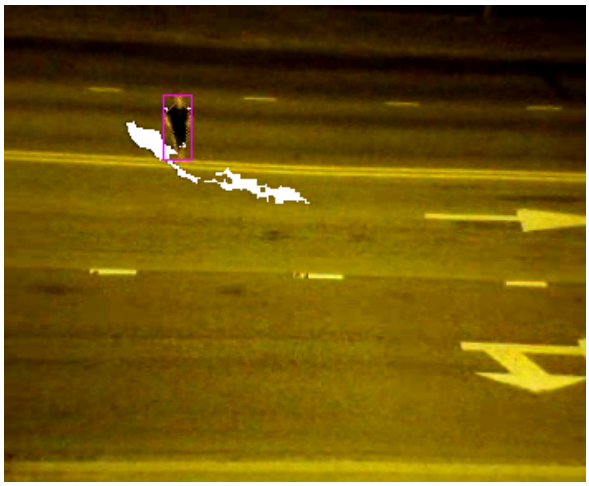
(b) Lying on the ground in direct sunlight



(c) Standing upright in indirect sunlight



(d) Lying on the ground in indirect sunlight



(e) Standing near a street light

(f) Lying on the ground near a street light

Figure 3.10: *Using cast shadows to recognize <Not Standing>. In the bottom four images, the shadow pixels detected by the author's implementation of the algorithm in section 2.2.3 are marked white. Obviously, a much larger shadow is detected when the person is standing upright, compared to when lying down. The large square marked in figures 3.10(c) and 3.10(d) is the Region Of Interest.*

shadow area relative to blob area. In contrast, extremities will significantly affect the size of the bounding rectangle and blob area.

How much shadow is cast when standing/lying, depends of course on the locations of the light sources relative to the object and camera, the number of light sources, and their intensities. If the location of the only light source coincides with the location of the camera, no shadow will be visible in the image. If there is more than one light source, one shadow will be cast for each source, and thus there is a much higher probability of the camera seeing some of the cast shadow. The reliability of cast shadows as a feature of interest therefore depends on how easy it is to control the lighting in the area.

Selected Features

The world height of the individual is a very important feature when considering <Not Standing>, as it relates directly to whether the person is standing upright or not, with no underlying assumptions on camera and light source parameters. It is not necessarily the case that a person appearing shorter than normal is lying or sitting on the ground (could be bending over to pick something up, for instance), but that person is certainly not standing upright. Using the expression "shorter than normal" means of course that the height of an individual must be tracked, so that a comparison can be made. The world height could be found using techniques developed in [6, 7, 8, 9]. This should definitely be investigated in future work.

Cast shadow is also very interesting as a feature of interest, even if it is often regarded as merely noise. Shadows can reveal a lot of information about the environment and subjects under surveillance. Shadows could be a particularly important source of information for single-sensor systems, such as the vast majority of existing surveillance systems. Even so, there seems to have been done little research on using cast shadow as a source of information in behavior recognition and automated security applications. Therefore this will be the feature selected for recognizing <Not Standing> in this thesis, along with the world speed of an individual.

3.7.2 Feature Extraction

The extraction of the speed of an individual is described in section 3.6.2. If the individual is not standing, the position found will generally not be that of the feet. This is because an individual lying on the floor invalidates the assumption that the feet will be close to the line through the centroid and the vertical vanishing point. However, if the individual is indeed not standing, then there will be very little change between the estimates of consecutive frames, and so the speed should still be calculated correctly (even if the estimated position is not that of the feet).

The extraction of cast shadow size consists of first finding the shadow pixels, and then finding the area of the shadow cast by each individual. The former has already been done by the motion detector (section 2.2.3). To do the latter accurately would be quite difficult, as it would require a 3D reconstruction of the scene (and if that could be done,

```
For each tracked individual p:

    shadowarea  = area of shadow cast by p
    blobarea    = area of the blob corresponding to p
    shadowratio = shadowarea / blobarea

    if shadowratio < notstandingmaximumratio

        notstandingframes++

        if notstandingframes >= notstandingminimumframes
        && mean speed of p over notstandingframes < notstandingmaximumspeed:

           notstanding = true

    else:

        notstandingframes = 0
```

Figure 3.11: *Algorithm: recognizing <Not Standing> using cast shadows.*

recognizing <Not Standing> would not require the use of shadows anyway). The area can be approximated by counting the shadow pixels in shadow regions that are touching the blob in the segmented foreground image. This is only a rough approximation, as overlapping shadows cast by several individuals will cause serious errors in the estimated area. Better algorithms should be investigated in the future.

3.7.3 Algorithm

Figure 3.11 shows a pseudo code for an algorithm capable of recognizing <Not Standing> using cast shadows. The following assumptions are necessary:

- The scene must be well-lit by at least one light source, preferably more. For this algorithm to be usable, the light sources should be distributed so that wherever an individual might stand, a shadow is detected by the camera.

- As with the <Chasing> algorithm, the people-density of the area should be relatively low. Occlusions remain a severe problem in all intelligent surveillance systems, and in particular in single-sensor systems. As mentioned in section 3.6.3, using a ceiling mounted camera helps a lot. Not only are there less occlusions between people, but the cast shadow is less likely to be occluded by obstacles, other people, or the individual itself.

Cast shadow will not be a good feature to use in environments where controlling the lighting is difficult, and in areas where the shadow is not in view of the camera (i.e. it is occluded by some static structure or moving object, or it is cast somewhere outside the camera view). An example of the former is most outdoor environments, where lighting is unpredictable and very difficult to control. An example of the latter is in areas where the people-density is relatively high, so occlusions occur often. Someone standing very close to a wall might also cause a too small shadow to be cast, depending on the number and locations of light sources. The more light sources are used, assuming they are positioned well, the more robust cast shadows become as a feature of interest.

Chapter 4

Implementation

This chapter describes an implementation of the algorithms from chapter 2.2, 2.3 and 3.

The chapter contains the following sections:

- Overview
- Modules
- Parameters
- Configuring
- Source Code
- Executables

4.1 Overview

The algorithms described in chapter 2.4 and 3 have been implemented in C++. This implementation will be denoted `Bully`, in order to simplify the text.

Figure 4.1 shows the overall structure of `Bully` as a UML class diagram. The control flow is shown in the UML sequence diagram in figure 4.2. This is the classical approach described in section 2.4, with three successive stages: *foreground segmentation* (or *preprocessing*), *tracking* and *analysis* of tracking data (in this case: *behavior analysis*).

The implementation of a stage is referred to in this text as a module, and contained in a class derived from the abstract class `Module`. Figure 4.3 shows the module hierarchy as a UML class diagram.

`Module` contains pointers to two important classes: `Params` and `BufferManager`. These are supplied by `System` as constructor parameters to concrete decendants. `Params` is simply a container (hash map) for the parameters given by the user in the configuration file, command line or directly in the source code. Table 4.1 and 4.2 lists all available parameters. `BufferManager` is simply a hash map of the buffers allocated. A buffer is a memory area

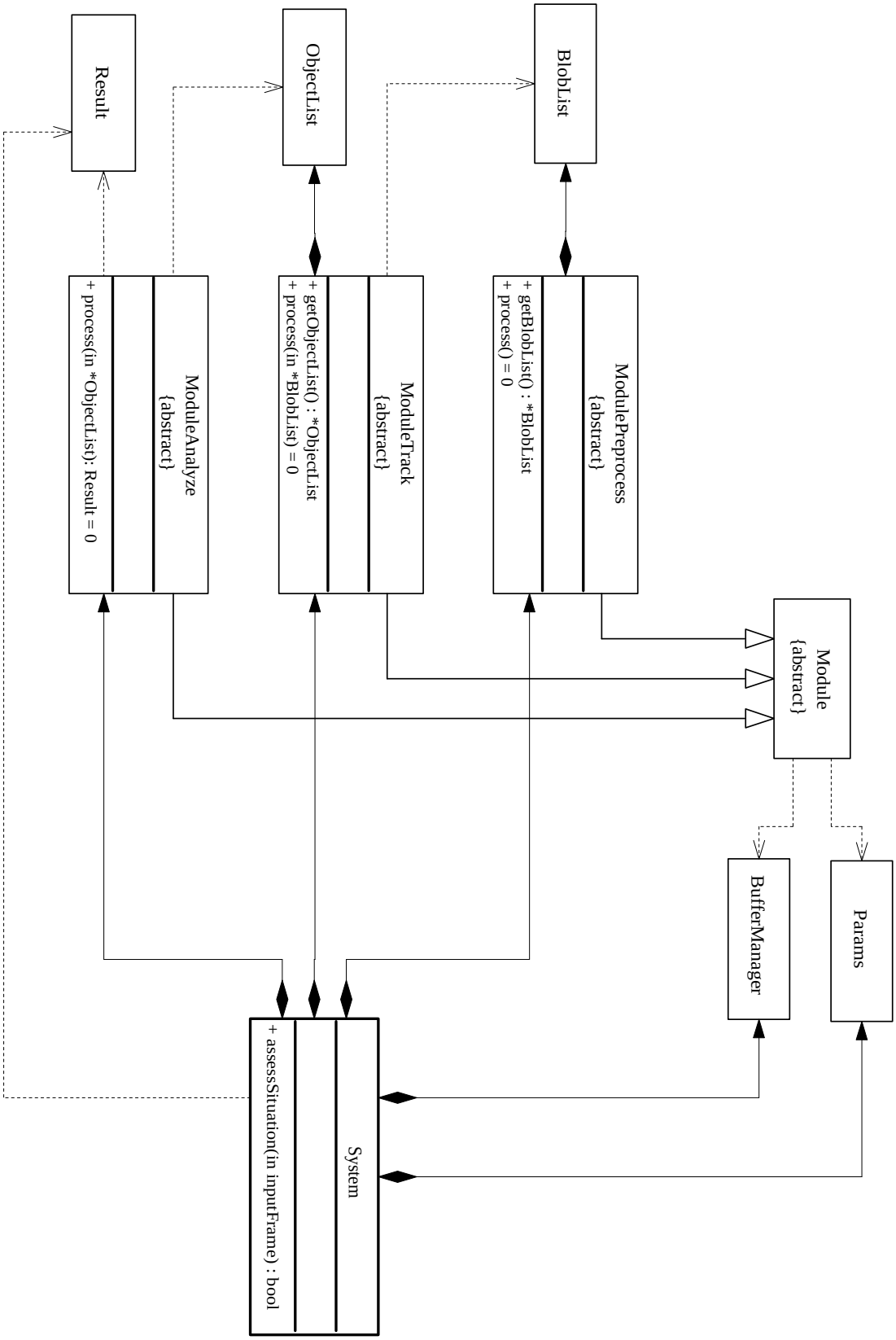


Figure 4.1: *UML class diagram showing the overall structure of* `Bully`.

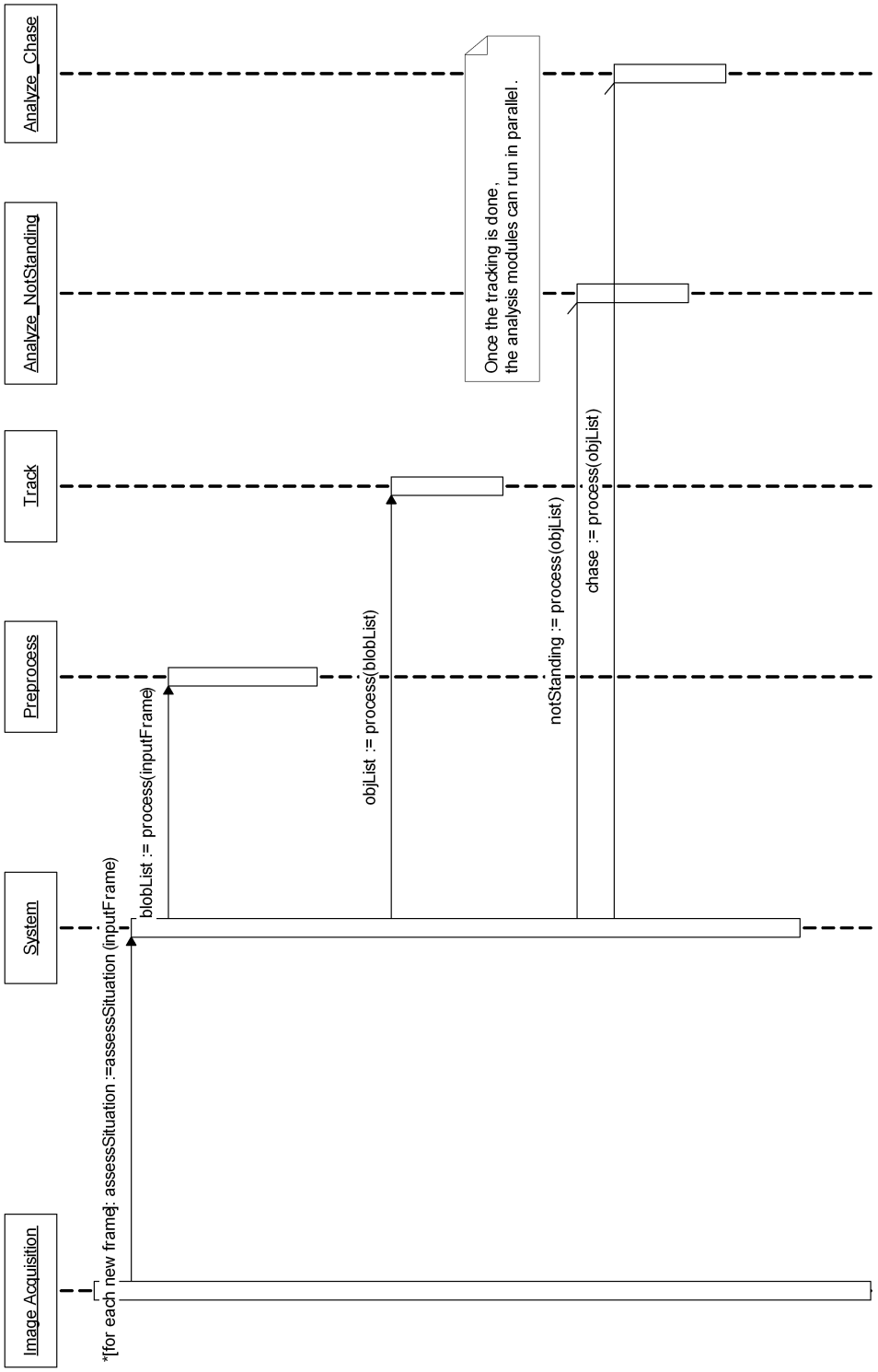


Figure 4.2: *UML sequence diagram showing the control flow of* `Bully`*. Note that the analysis modules do not run in parallel in the current implementation, but this would be simple to change to multithreaded execution.*

Figure 4.3: *UML class diagram showing the module hierarchy.*

used to store an image, for instance the input frame, intermediate results and the output frame. These two pointers in `Module` means every concrete module implementation will have access to all the parameters and all the buffers.

As can be seen from figure 4.3, there are three classes which inherit from `Module`: `ModulePreprocess`, `ModuleTrack` and `ModuleAnalyze`. Each of these are themselves abstract classes, and are in fact interfaces that must be satisfied in concrete module implementations. The `System` class actually holds pointers to these abstract classes, which enables selection of concrete module classes at run-time.

This is beneficial because it makes it simple to implement more than one concrete module class, each using different algorithms to accomplish the task (motion detection, tracking or analysis). The three different preprocessing modules is an example. The module `ModulePreprocess_Default` makes no attempt at detecting and removing shadow pixels, whereas the other two preprocessing modules implements different algorithms for removing shadows.

More importantly, it is possible to select at run-time which behaviors should be recognized, since each concrete class deriving from `ModuleAnalyze` attempts to recognize one specific behavior. Unlike the other modules, however, more than one analysis module can be selected at any time, meaning as many behaviors can be recognized as there are concrete analysis module implementations.

Each module contains a method called `process(...)`. This is the method that is called by `System` once each frame, to make the module perform its specific task. The preprocessing modules perform foreground segmentation, the tracking modules track the blobs in the foreground, and the analysis modules analyze the tracked objects, attempting to recognize a specific behavior exhibited by any human in the scene. Each preprocessing module maintains a list of foreground blobs, and each tracking module maintains a list of tracked objects. Each analysis module simply returns a list of alarms from `process(ObjectList*)`, which is empty if no alarms were triggered. An alarm is triggered if the specific behavior the module is looking for is recognized, and it includes a short description of the type (such as "Not Standing", or "Chase"), along with the area where the alarm was triggered.

4.2 Modules

4.2.1 `ModulePreprocess`

As mentioned in section 4.1, there are three different preprocessing modules:

- `ModulePreprocess_Default`: simple background subtraction. Does not attempt to detect and remove cast shadows.
- `ModulePreprocess_HSV`: implementation of the shadow detection algorithm described in [10]. See appendix A.
- `ModulePreprocess_CNCC`: implementation of the shadow detection algorithm described in [19]. See appendix B.

See table 4.1 for available parameters, along with [10, 19] and appendix A and B for appropriate usage.

4.2.2 `ModuleTrack`

- `ModuleTrack_Default`: simple region tracker. Does not attempt to handle occlusions or splitting/merging.
- `ModuleTrack_Tdt4725`: the region tracker described in section 2.3.3 and [30].
- `ModuleTrack_Reading`: the region tracker from the Reading People Tracker [42], described in section 2.3.3.

4.2.3 `ModuleAnalyze`

- `ModuleAnalyze_Default` is a dummy module that exists only to make it possible to run `Bully` without actually selecting any analysis modules. It does nothing.
- `ModuleAnalyze_Debug` is module that can be used to get various useful information. Examples are easy selection of ROIs, getting information about a tracked object by clicking on the corresponding blob, and getting information about floor plane positions given the mouse cursor position in the image plane.
- `ModuleAnalyze_NotStanding`: using cast shadows and world speed (relative to the floor plane) to recognize <Not Standing>. A straightforward C++ implementation of the pseudo code in figure 3.11.
- `ModuleAnalyze_Chase`: using world positions to recognize <Chasing>. A straightforward C++ implementation of the pseudo code in figure 3.9.

4.3 Parameters

Table 4.1 and 4.2 lists the available parameters, with a short description of each. A couple of notes about the notation in the tables:

- MULTI: this means the parameter holds a list of values. The ”default value” is ”empty”, which means that the list is empty. See figure 4.4 and 4.5 for how to set parameter values.
- bool: an integer, either 0 (false) or 1 (true).
- Matrix/Vector/Point: these are given as strings, to simplify the implementation
  - 2D Point: a point in 2D space, ”(x y)”.
  - 3D Point: a point in 3D space, ”(x y z)”.

Name	**Possible Values**	**Default Value**	**Description**
inputPath	string	""	path to input (video)
backgroundPath	string	""	path to background/reference frame (output from bgtool)
modulePreprocess	"default", "hsv", "cncc"	"default"	which preprocessing module to use
moduleTrack	"default", "tdt4725", "reading"	"default"	which tracking module to use
moduleAnalyze	"default" "debug" "notstandings" "chase"	"default"	which analysis module(s) to use (MULTI) - provides various information useful for debugging - using shadow to infer <Not Standing> - using floor plane positions to infer <Chasing>
roi	2D Vector	empty	Region Of Interest (MULTI)
roni	2D Vector	empty	Region Of No Interest (MULTI)
prepForegroundDiffLower	[0,255]	8	lower bound on FG/BG diff. of a foreground pixel
prepShadowCNCCLower	[0,1]	0.80	lower bound on FG/BG NCC of a shadow pixel
prepShadowCNCCWidth	$2n+1,\ n \geq 1$	7	width of NCC mask
prepShadowHSVLumLower	[0,1]	0.05	lower bound on FG/BG luminance ratio of a shadow pixel
prepShadowHSVLumUpper	[0,1]	0.98	upper bound on FG/BG luminance ratio of a shadow pixel
prepShadowHSVHueUpper	[0,255]	50	upper bound on FG/BG hue diff. of a shadow pixel
prepShadowHSVSatUpper	[0,255]	50	upper bound on FG/BG sat. diff. of a shadow pixel
prepRegionAreaLower	[0,-]	100	lower bound on region areas
prepStructElementSize	$2n+1,\ n \geq 1$	3	size of morphological structuring element
prepPosition	"median", "centroid", "centerbbox"	"median"	definition of the position of a blob

Table 4.1: `Bully` *parameters available to the user. Continues in table 4.2.*

Name	**Possible Values**	**Default Value**	**Description**
trackHomography	Matrix	Identity	3×3 homography matrix:
trackVerticalVP	2D Point	(0 0)	Vertical vanishing point in the image plane
analChaseFrames	[0,-]	20	number of frames to analyze
analChaseSpeedLower	[0,-]	5.0	lower bound on the speed of a running person
analChaseDirDiffUpper	[0,-]	1.0	upper bound on the diff. of dir. of two people
analChaseDistDiffLower	[0,-]	1.0	lower bound on the min. dist. between two people
analNotStandingFrames	[0,-]	15	number of frames to analyze
analNotStandingShadowUpper	[0,1]	0.01	upper bound on the relative shadow area of a person
analNotStandingSpeedUpper	[0,-]	3.0	upper bound on the mean speed of a person
drawInput	bool	1	copy input frame to output
drawBlobRect	bool	0	draw a rectangle around detected blobs
drawBlobContour	bool	0	draw the contour of detected blobs
drawObjectRect	bool	1	draw a rectangle around tracked objects
drawTrajectory	bool	1	draw the trajectory of tracked objects
drawShadow	bool	0	mark shadow pixels (white)
drawROI	bool	1	draw a rectangle around the Region Of Interest
drawAlarm	bool	1	draw a rectangle and short desc. at the area of an alarm
printTrackedObjects	bool	0	print the number of tracked objects
printShadowArea	bool	0	outputs the area of each cast shadow in contact with a blob

Table 4.2: *Continuation of table 4.1.*

```
# This is the file bullyconfig.txt (this line is a comment):
drawROI = 1
drawAlarm = 1
modulePreprocess = cncc
moduleTrack = default
moduleAnalyze = notstandings
roi = (181 47) (217 47) (289 287) (109 287)
roi = (0 47) (100 47) (120 200) (30 220)
trackHomography = 1 2 3; 4 5 6; 7 8 9
```

Figure 4.4: *Configuring Bully using a configuration file*

- 2D Vector: a list of 2D points, "$(x_1\ y_1)(x_2\ y_2)...(x_n\ y_n)$".
- 3D Vector: a list of 3D points, "$(x_1\ y_1\ z_1)(x_2\ y_2\ z_2)...(x_n\ y_n\ z_n)$".
- Matrix: row major, and the rows are separated by semicolons. An $m \times n$ matrix is written as follows: "$M_{11}\ M_{12}\ \ldots M_{1n};\ M_{21}\ M_{22}\ \ldots M_{2n}; \ldots;\ M_{m1}\ M_{m2}\ \ldots M_{mn}$"

RONI is a Region Of No Interest. It is useful for masking out particularly noisy areas. An examples is a building facade, where windows and doors can open and close.

All parameters concerning distances are given in meters.

The unit for time used in setting the parameters is *frames*. It would of course be much better to use some unit independent of the image acquisition characteristics, such as *seconds*. However, the author had problems getting reliable values for the FPS of a video from the decoding software. This error probably originated in the camera drivers which were used for the video capture. As a result, the author found it best to use frames as the time unit.

Speed is relative to the floor plane (*world speed*) unless otherwise specified. Speed is always given in meters per frame, and not meters per second. It would of course be much better to have speeds given in meters per second, and this would be easy to implement in a production system, as the frame rate would be known (fixed, or found by querying the image acquistion system). The reason for not using meters per second is the problem of incorrect FPS explained above.

4.4 Configuring

The parameters that control the algorithms can be set via a configuration file or directly through code.

4.4.1 Configuration File

Figure 4.4 shows an example of setting parameters in a configuration file. Note that whitespace is not significant, and that each line must be on the format:

```
Bully b;
b.params.set("drawROI", 1);
b.params.set("drawAlarm", 1);
b.params.set("modulePreprocess", "cncc");
b.params.set("moduleTrack", "default");
b.params.set("moduleAnalyze", "notstandings");
b.params.set("roi", "(181 47) (217 47) (289 287) (109 287)");
b.params.set("roi", "(0 47) (100 47) (120 200) (30 220)");
b.params.set("trackHomography",
                " 0.0066 -0.0009 -0.6558;"\
                "-0.0007  0.0143 -0.7320;"\
                "-0.0000 -0.0008  0.1840");

// more C++ here...
bool drawROI = (int)b.params.getValue("drawROI") != 0;

std::vector<Bully::Vector2> roi1, roi2;
b.params.getVector(b.params.getString("roi",0), roi1);
b.params.getVector(b.params.getString("roi",1), roi2);

Bully::Matrix3<double> m;
b.params.getMatrix(b.params.getString("trackHomography",0) , m);
std::cout << m; // Print the homography to the console
```

Figure 4.5: *Configuring Bully using the parameter API. The homography in the figure is an actual example used with the video* `glassgard_running.avi` *in the* `testdata` *directory.*

```
paramName = paramValue
```

Also note that multiple ROIs and RONIs can be set. Refer to table 4.1 and 4.2. The executable `bullytest.exe` is very convenient to use when setting the ROI: simply use the mouse and click at four different locations on the screen, and the coordinates will be printed to the console.

4.4.2 Parameter API

Figure 4.5 shows an example of setting and reading parameters using the parameter API.

4.4.3 Profiling

The source code of `Bully` has a built-in profiler made by the author. It can be enabled by uncommenting the line

```
//#define ERIKSUL_PROFILE
```

Class/file/executable	Ca. lines of code
`ModulePreprocess`	370
`ModulePreprocess_Default`	60
`ModulePreprocess_HSV`	120
`ModulePreprocess_CNCC`	140
`ModuleTrack`	30
`ModuleTrack_Default`	115
`ModuleTrack_Tdt4725`	340
`ModuleTrack_Reading` (not including RPT source)	170
`ModuleAnalyze`	70
`ModuleAnalyze_Default`	25
`ModuleAnalyze_Debug`	115
`ModuleAnalyze_Chase`	120
`ModuleAnalyze_NotStanding_Shadow`	85
...	
`Bully` total	5625
bullytest	300
bgtool	263
Video playback / image acqusition	490
Timing / profiling	235
Total LOC written by the author	6900

Table 4.3: *C++ Source Code Statistics.*

in the file `<top of source tree>/timing/profiling.h`. The profiler currently only works under Microsoft Windows XP and newer, but it would be a simple task to make it work on any posix system. When run in profiling mode, a file called "bullyprofileXX.txt" will be created in the working directory upon program exit. XX is a string of digits representing the time at which the file was created. This file will contain useful profiling information, such as how many times a function was called, and how many percent of the calling functions executing time (wall clock) was spent in the called function. It is a rough approximation, but the data is still useful.

4.5 Source Code

See table 4.3 for some statistics of the source code. In addition to the C++ source code, some Matlab code was written to test the shadow detection algorithms described in [10, 19]. This code can be seen in appendix A and B. All source code is available on the included DVD, and also on the author's homepage at NTNU: `http://www.idi.ntnu.no/~eriksul`.

The source code compiles on Windows XP, and with only minor modifications on Linux and any posix system. Solution and project files are included for Visual Studio 2005.

4.5.1 External Libraries

- OpenCV: used for low-level image processing, such as morphological operations, etc.
- ffmpeg: used for video decoding.
- SDL: used for platform independent keyboard and mouse input, as well as graphical output.
- Reading People Tracker: the region tracker from RPT was used in `ModuleTrack_Reading`.

4.6 Executables

4.6.1 `bullytest.exe`

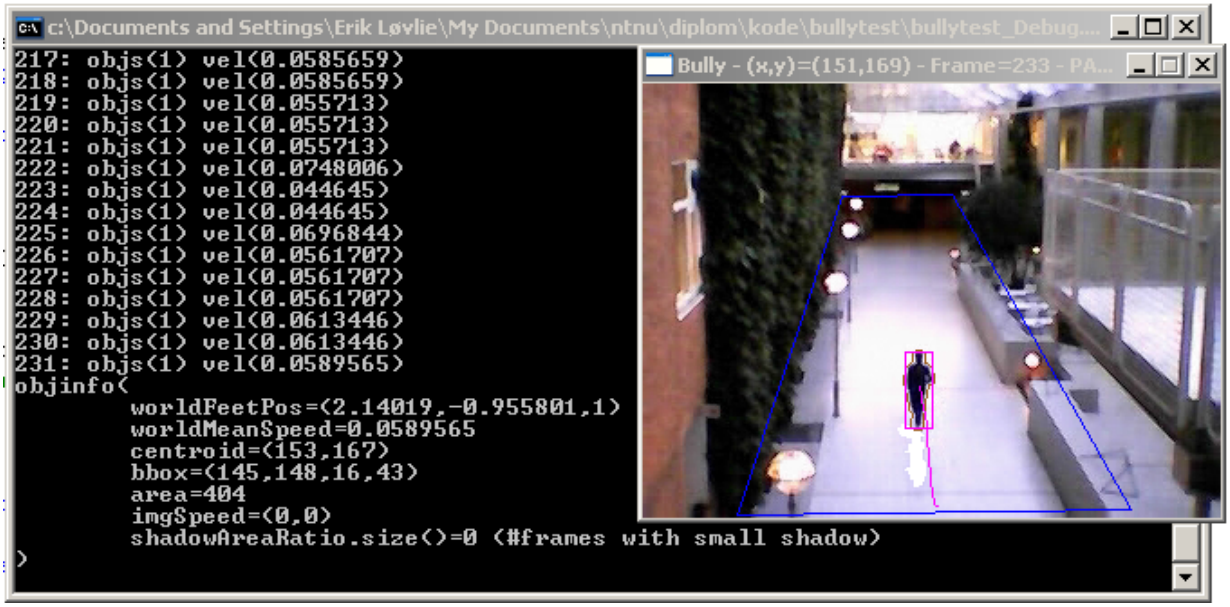
Test program for bully, using videos stored on a harddisk as source. Figure 4.6(a) shows `bullytest.exe` in action. The user has just clicked on the running person, and the information about the object has been printed to the console. Note that it is possible to turn off all output by setting the appropriate parameters (drawXX in table 4.2) to zero. This should of course be done before profiling or benchmarking.

User Interface

Some actions are triggered by keys on the keyboard:

- **Escape (Esc):** exit bullytest.
- **Space:** pause video playback and analysis.
- **'f':** "fast", no frame rate limitation.
- **Up arrow:** increase frame rate.
- **Down arrow:** decrease frame rate.
- **Print Screen (Prt Sc):** take screen shot, and store as `screenXX.png` and `screenXX.params` in the current working directory. `XX` is a string of digits representing the time at which the file was created. The file with the file name extension `.params` is a text file containing the parameters used when taking the screen shot.
- **'b'**: show all buffers used in motion detection (input, various intermediate results, and segmented foreground). This works only when `ModuleAnalyze_Debug` is selected.

Some actions require pressing a mouse button while holding down a key on the keyboard. The following actions apply only if the module `ModuleAnalyze_Debug` is selected (which it is by default in `bullytest.exe`). While pushing the key



(a) `bullytest.exe`

(b) `bgtool.exe`: extracting background

(c) `bgtool.exe`: showing stored background

Figure 4.6: *Executables*

- **'o'**, press any mouse button on a blob, and some information about the corresponding object will be printed to the console
- **'p'**, press any mouse button somewhere on the screen, and the position of the mouse cursor in the image along with the corresponding position on the floor plane will be printed to the console
- **'d'**, press the left mouse button somewhere on the screen, and the position of the mouse cursor becomes the start point used to calculate the distance between two points on the floor plane
- **'d'**, press the right mouse button somewhere on the screen, and the distance in the floor plane between the start point (selected with the left mouse button) and the current position of the mouse cursor is printed to the console
- **'r'**, press the left mouse button somewhere on the screen to add a point to the ROI
- **'r'**, press the right mouse button somewhere on the screen to reset the ROI

4.6.2 `bgtool.exe`

Tool to extract the background from videos. The color of a background pixel is calculated as the color that occured most frequently at that pixel. The calculation is done per color channel. This is a very simple background model, but it was adequate for the author's needs.

The user interface is extremely simple: drag and drop, or use the command line with no arguments for usage information. The input file is either a video file, or a file containing the output of a previous background extraction.

If the input file is a video file, the process of extracting the background will begin immediately. The output is then a file containing the extracted background, with the same file name as the video file, but with `.bg` appended. The user can press escape at any time to interrupt the extraction process, which will output the extracted background at that time to the output file. Figure 4.6(b) is a screenshot of the tool while extracting a background.

If the input file has a `.bg` extension on its filename, the contained background is shown. This can be seen in figure 4.6(c).

Chapter 5

Demonstration

This chapter presents some results that demonstrates the algorithms and implementation described and developed earlier in this thesis.

The chapter contains the following sections:

- Configuration
- Results

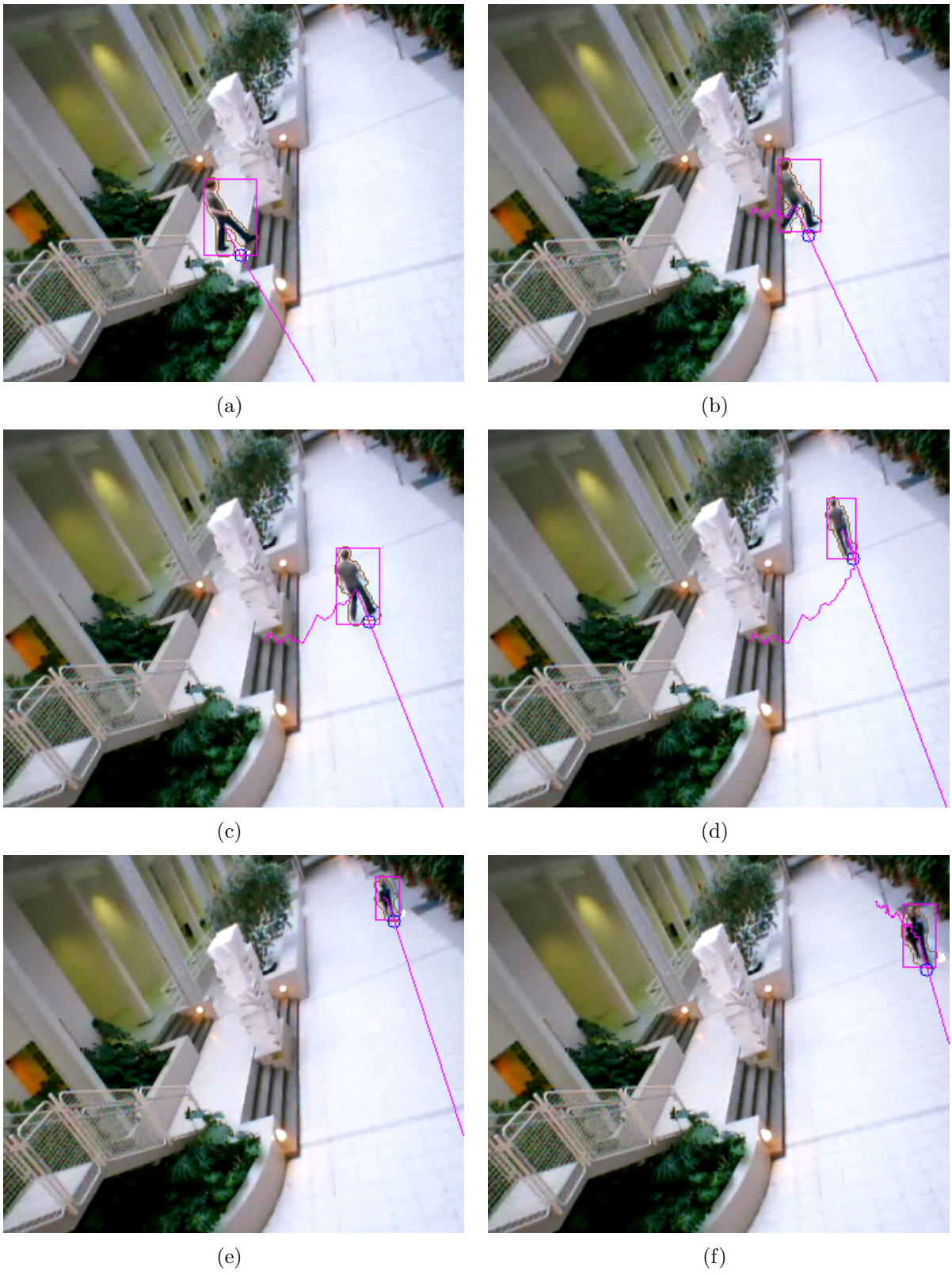
5.1 Configuration

The camera used to capture the videos was a Creative NX Ultra WebCam. The analysis was performed on pre-recorded videos. On a 1.4 GHz Intel Centrino laptop with 768 MB of RAM, the framerate varied between 10 and 50 FPS, depending on which modules (algorithms) were selected. The bottleneck of the implementation is the motion detection module, in particular the shadow removal algorithms.

Each figure will be marked by the location, which also identifies a table of parameters.

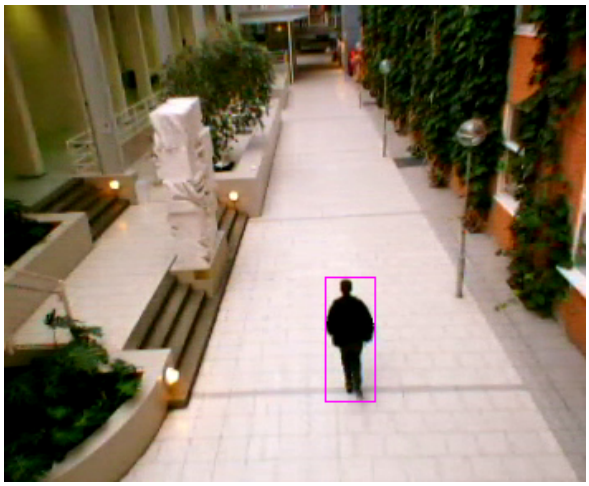
5.2 Results

The images in this section that show an alarm, will have a rectangle around the area where the alarm was detected, along with an abbreviation of the behavior recognized. A red rectangle signifies the area of an alarm, or someone involved in the behavior that triggered the alarm. A blue polygon signifies a ROI.

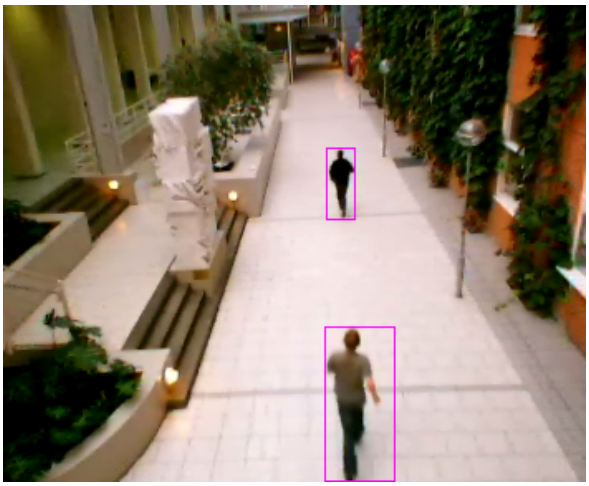


(a) (b) (c) (d) (e) (f)

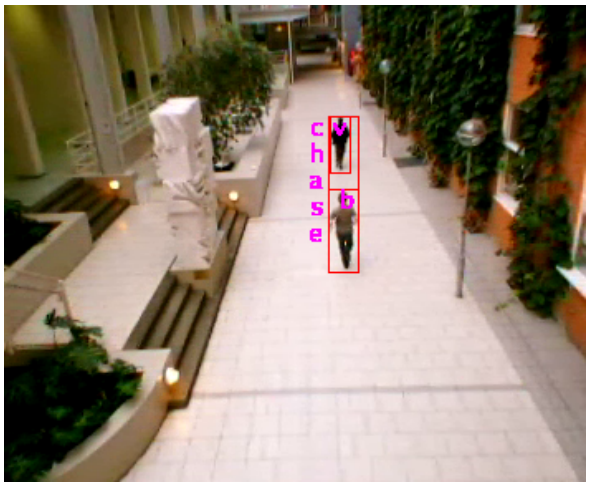
Figure 5.1: *Estimating the position of the feet. The camera has been rotated slightly. The straight line shown goes through the median coordinates of the blob and the vertical vanishing point. The blue circle is the estimated feet position. Figure 5.1(a) to 5.1(d) shows an individual walking down the stairs and away from the camera. Figure 5.1(e) and 5.1(f) shows another individual walking towards the camera. Location: GLASSGAARD1.*



(a) Individual running away from the camera

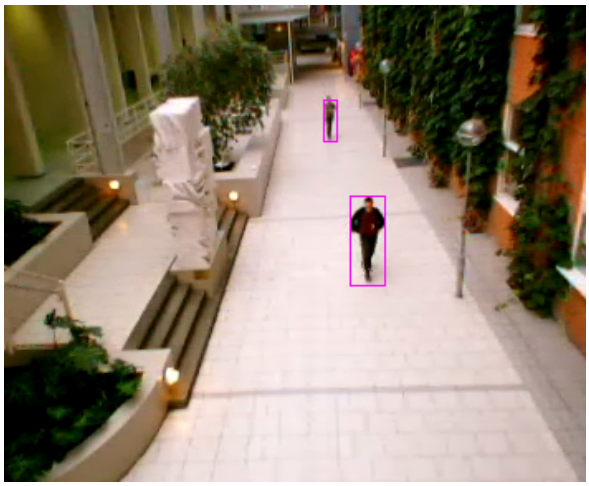


(b) Second individual entering the scene

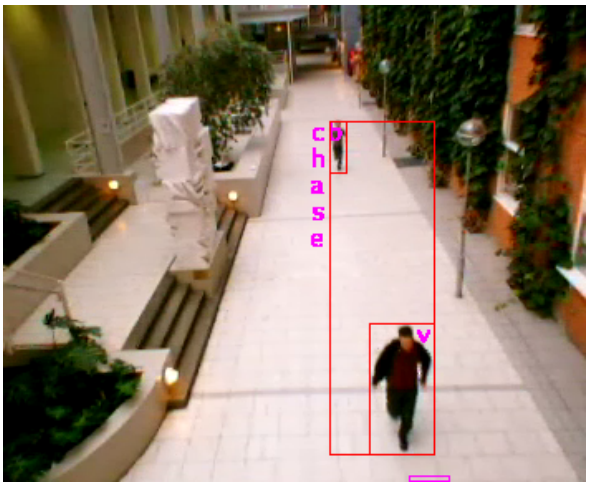


(c) A <Chase> has been recognized

(d) Two people running towards the camera



(e) Same as figure 5.2(d), but closer

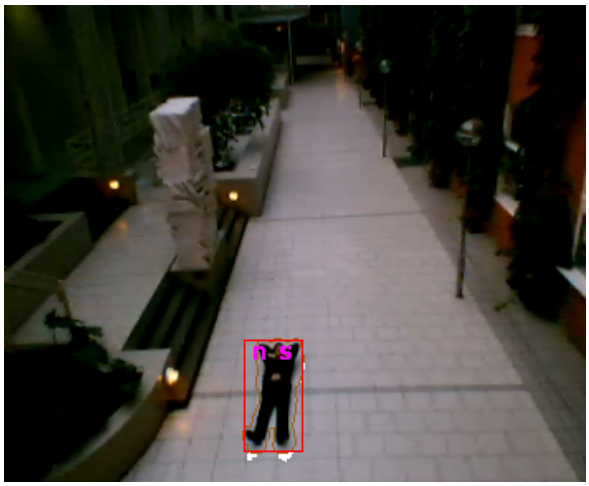


(f) A <Chase> has been recognized

Figure 5.2: *Recognizing <Chasing>. "Bully" and "victim" is marked with a* `b` *and* `v` *respectively. The correspondence points used to compute the homography is shown in figure 5.6(a). Location: GLASSGAARD2.*

(a) An individual standing on the floor

(b) Sitting down; <Not Standing> recognized



(c) Lying down; <Not Standing> recognized

(d) Sitting down and legs wide apart; <Not Standing> recognized

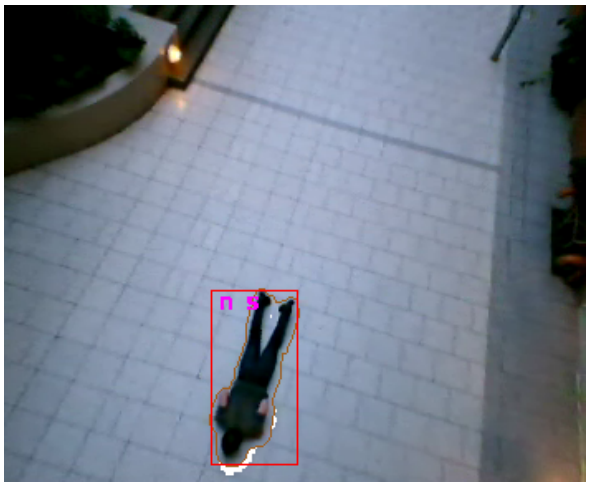
(e) Lying down; <Not Standing> recognized

(f) Standing up again

Figure 5.3: *Recognizing <Not Standing> using cast shadows. The alarm area has been marked* `ns`*, for "not standing". Detected shadows pixels have been marked white. The correspondence points used to compute the homography is shown in figure 5.6(a). Location: GLASSGAARD3.*

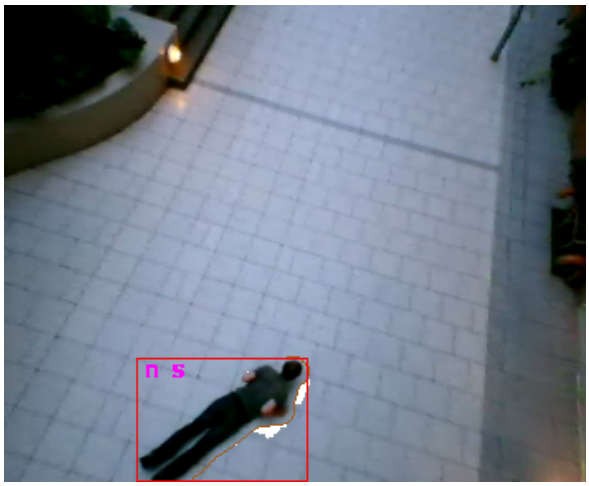
(a) Individual walking

(b) About to lie down

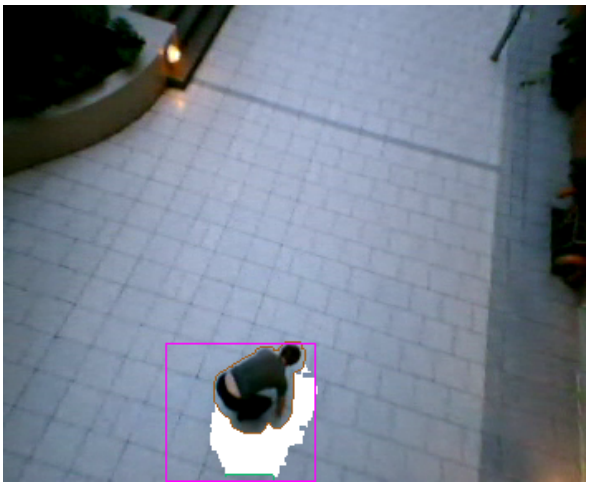


(c) Lying down; <Not Standing> recognized

(d) Lying down; <Not Standing> recognized

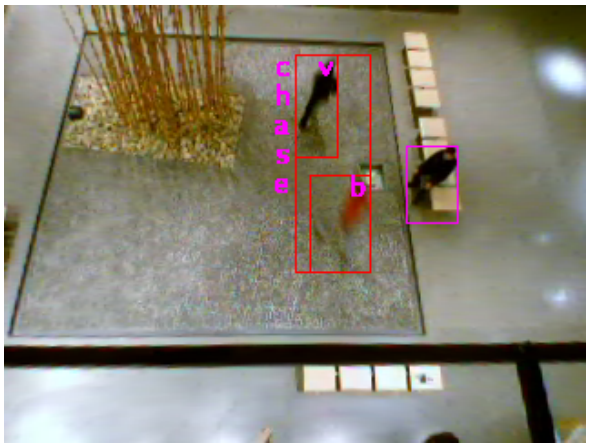


(e) Lying down; <Not Standing> recognized

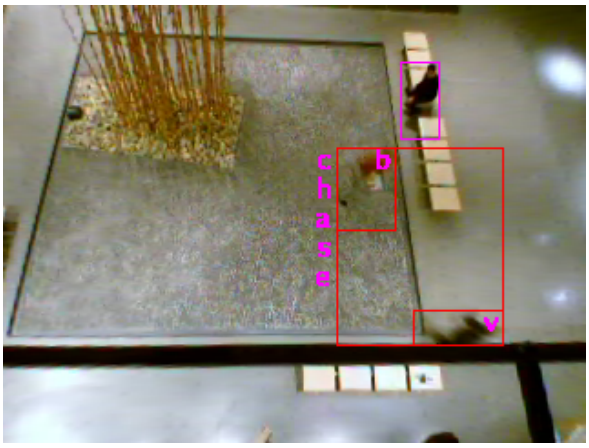


(f) Standing up again

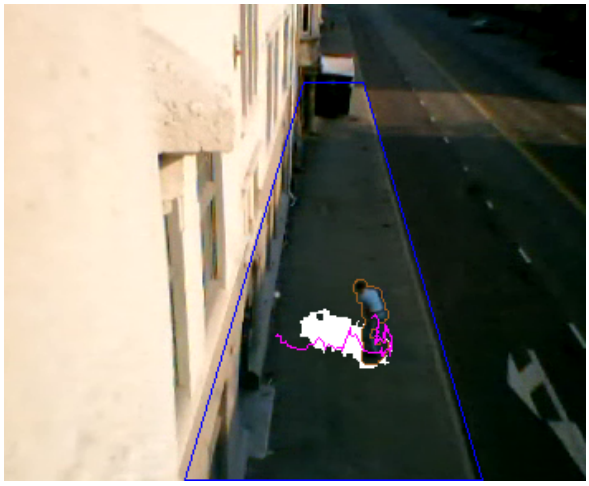
Figure 5.4: *Recognizing <Not Standing> using cast shadows. The camera has been mounted on the ceiling. The correspondence points used to compute the homography is shown in figure 5.6(b). Location: GLASSGAARD4.*



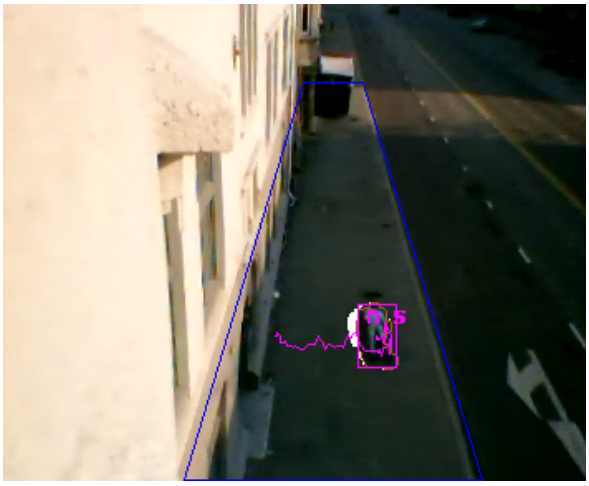
(a) Chase away from camera; <Chase> recognized



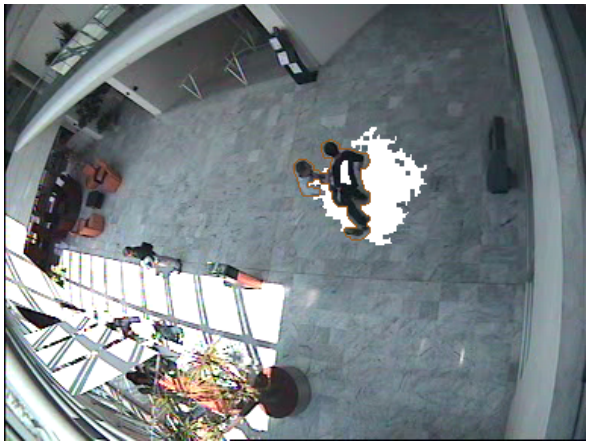
(b) Chase towards camera; <Chase> recognized



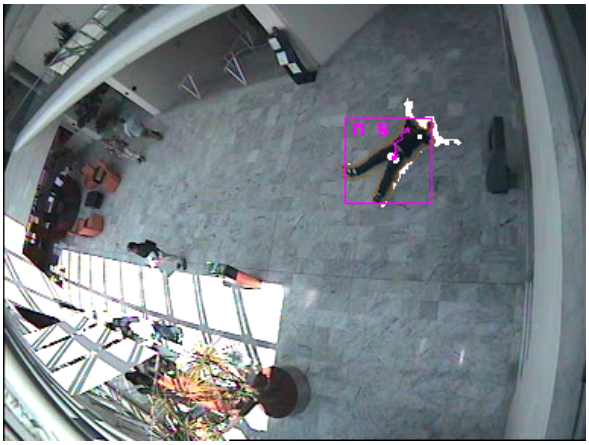
(c) About to lie down



(d) Lying down; <Not Standing> recognized

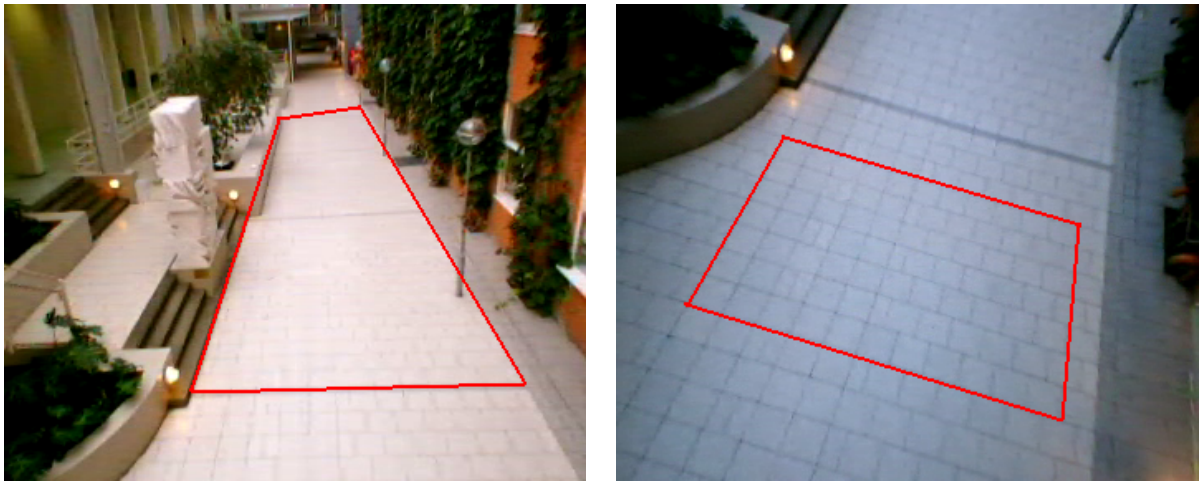


(e) Standing and fighting



(f) Passed out; <Not Standing> recognized

Figure 5.5: *Various scenarios. The top four images show that one can recognize the behavior <Not Standing> by measuring the area of cast shadows relative to the area of the corresponding foreground blob. Figures 5.5(e) and 5.5(f) are from videos in the CAVIAR dataset [35].*



(a) GLASSGAARD2 & GLASSGAARD3 (b) GLASSGAARD4

Figure 5.6: *Homography correspondence points.*

Parameter	**Value**
modulePreprocess	cncc
prepForegroundDiffLower	8
prepShadowCNCCLower	0.95
prepShadowCNCCWidth	7
prepRegionAreaLower	200
prepStructElementSize	3
prepPosition	median
trackHomography	0.0066 -0.0039 0.1326; 0.0023 0.0073 -0.8744; -0.0002 -0.0007 0.4665
trackVerticalVP	(543 814)

Table 5.1: *Parameter values: GLASSGAARD1.*

Parameter	**Value**
modulePreprocess	cncc
moduleTrack	tdt4725
moduleAnalyze	default
moduleAnalyze	debug
moduleAnalyze	chase
prepForegroundDiffLower	6
prepShadowCNCCLower	0.9
prepShadowCNCCWidth	7
prepRegionAreaLower	50
prepStructElementSize	3
prepPosition	median
trackHomography	0.0071 -0.0023 -0.6801; -0.0003 0.0127 -0.6003; 0.0002 -0.0016 0.4205
trackVerticalVP	(228 852)
postGroupFramesLower	30
analChaseFrames	24
analChaseSpeedLower	0.1
analChaseDirDiffUpper	0.6
analChaseDistDiffLower	1

Table 5.2: *Parameter values: GLASSGAARD2.*

Parameter	Value
modulePreprocess	cncc
moduleTrack	default
moduleAnalyze	default
moduleAnalyze	debug
moduleAnalyze	notstandings
prepForegroundDiffLower	10
prepShadowCNCCLower	0.9
prepShadowCNCCWidth	7
prepRegionAreaLower	200
prepStructElementSize	3
prepPosition	median
trackHomography	0.0071 -0.0023 -0.6801; -0.0003 0.0127 -0.6003; 0.0002 -0.0016 0.4205
trackVerticalVP	(228 852)
analNotStandingShadowUpper	0.5
analNotStandingFrames	15
analNotStandingSpeedUpper	3

Table 5.3: *Parameter values: GLASSGAARD3.*

Parameter	Value
modulePreprocess	cncc
moduleTrack	default
moduleAnalyze	default
moduleAnalyze	debug
moduleAnalyze	notstandings
prepForegroundDiffLower	8
prepShadowCNCCLower	0.9
prepShadowCNCCWidth	7
prepRegionAreaLower	200
prepStructElementSize	3
prepPosition	median
trackHomography	0.0071 -0.0023 -0.6801; -0.0003 0.0127 -0.6003; 0.0002 -0.0016 0.4205
trackVerticalVP	(228 852)
analNotStandingShadowUpper	0.5
analNotStandingFrames	15
analNotStandingSpeedUpper	3

Table 5.4: *Parameter values: GLASSGAARD4.*

Chapter 6

Conclusion

This chapter includes a summary of the thesis, followed by a discussion of the advantages and disadvantages of the algorithms and implementation described and developed earlier in this thesis. The thesis concludes with a section on interesting areas for future work.

The chapter contains the following sections:

- Summary
- Discussion
- Future Work

6.1 Summary

This thesis has examined *the whole process* of recognizing unwanted human behaviors from videos taken by surveillance cameras. Algorithms for motion detection and tracking, which are the basis for individual behavior recognition, have been described and implemented, along with new algorithms for the recognition of chase/fleeing scenarios and people lying on the floor.

First, an overview of the state of the art in automated security and behavior recognition has been presented. The difference between crowd and individual behavior recognition has been described, and individual behavior recognition has been divided into three processes: *motion detection*, *tracking* and *behavior recognition*.

The basic algorithms for motion detection have been described, and problems identified. The problems include random noise, low FG/BG contrast, cast shadows and reflections. Then algorithms that solve these problems have been described, focusing primarily on random noise and cast shadows due to the impact these problems have on the tracking process. Images from the individual steps in the motion detection process were also presented.

The basic algorithms for tracking have been described, and problems identified. The problems include errors in foreground segmentation and occlusions. Then algorithms that deal with these problems are presented: (*i*) a simple region tracker that handles merging/splitting and low FG/BG contrast, and (*ii*) an advanced people tracker, the Reading

People Tracker. The latter can use multiple sensors, and uses a region tracker and an active shape tracker in combination for better accuracy and robustness.

Then the behaviors of interest in the context of this thesis have been defined, and reduced to a set of concrete behavior descriptions. The different approaches to pattern recognition were discussed, followed by the design of an algorithm for recognizing the concrete behavior description <Chasing>. This algorithm is independent of camera angle and rotation, due to the use of single view metrology to find the positions of the feet of an individual. Next the characteristic features of <Not Standing> were discussed, and an algorithm that makes use of cast shadows presented.

Finally, some results from using the previously described algorithms on real video sequences are presented.

6.2 Discussion

This section will discuss the advantages and disadvantages of the algorithms presented in section 2.2, 2.3 and 3.

6.2.1 Motion Detection

The simple static background model employed in this thesis works quite well in most indoor scenarios. If the surveillance system is to be used in environments where illumination changes, or objects in the background are moving (i.e. leaves, grass), for instance most outdoor environments, then more complex background models must be used [45].

Cast shadows cause serious errors in foreground segmentation unless correctly detected and removed. Also, cast shadows (and reflections) constitute a very important source of information about the geometry of the scene and subjects under surveillance. It is therefore very important to detect cast shadows.

The algorithm in [19] does a good job of detecting shadow pixels, and only two parameters need to be decided: the size of the cross-correlation neighborhood, and the cross-correlation threshold for shadow pixels. This makes it very attractive. It is, however, also quite computationally expensive.

The algorithm in [10] produces some errors, due to not considering texture similarity. It also requires setting more parameters: five different thresholds relating to luminosity and color. This algorithm should only be used if the algorithm in [19] is too expensive.

6.2.2 Tracking

People tracking is a very active area of research, and so much more powerful trackers exist then what has been described in this thesis. A couple of examples are W^4 [20] and the Reading People Tracker [42].

The tracker implemented in this thesis does a fairly good job of tracking individuals in areas of relatively low people-density, and retrieving the trajectory after an occlusion (or

splitting from a group) usually succeeds in such areas.

One distinct disadvantage is that no attempt is made at tracking an individual while grouped with other individuals. The Reading People Tracker tries to solve this by using a combination of bottom up and top down approach; a simple region tracker combined with an active shape tracker. The active shape tracker attempts to fit active shapes to the blobs using a head detector for initialization.

While this provides somewhat more accurate tracking compared to the tracker implemented in this thesis, occlusions are still a severe problem. As a consequence, much research is directed at multi-sensor tracking.

6.2.3 Estimating Feet Positions

The advantage of using the proposed algorithm is that it is independent of camera angles and rotations, as shown in image 5.1. This is important in automated surveillance due to the large number of different camera configurations that are in use. It is particularly interesting in configurations with a ceiling mounted camera.

The main disadvantage is of course that this method only works on people that are standing. It works well for standing and running individuals, but does not give good estimates for lying and sitting individuals. There will also be a small error caused by leaning while running, but the size of this error has not been investigated in this thesis. Another disadvantage is that the measured speed of an individual tends to be a little uneven, because the estimated position of the feet oscillates between the two feet of the individual. Using the mean speed over some time interval is usually enough to get good estimates, however.

Single view metrology is very attractive because it allows for a minimal camera calibration with no knowledge of the camera used. This makes it possible to get 3D information from a single view of the scene. However, single view metrology assumes the pinhole camera model, which does not model radial distortion. Radial distortion often occurs when using cameras with a wide-angle lens, as can be seen in figure 5.5(e) and 5.5(f). [7] describes a method for warping a radial distorted image into a perspective image. This method should be used if wide-angle lenses are used for image acquisition.

6.2.4 Recognizing <Chasing>

The algorithm presented for recognizing <Chasing> is simple and computationally inexpensive. As shown in the results (figure 5.2), it is independent of camera angles and rotations, and it identifies the offender and the victim.

It has a few disadvantages. It relies on accurate information about the positions of individuals. The positions are extracted using the estimated position of the feet in the image and a planar homography. The feet of individuals are quite often occluded, however, particularly when the camera angle is fairly oblique (figure 5.2). Therefore this algorithm works best with ceiling mounted cameras, as that minimizes the frequency of occlusions.

However, ceiling mounted cameras bring another disadvantage: a small area in the field of view. This means that if there is a chase occuring, the subjects will only be in the field of view for a very short time. To make sure the chase is still recognized, the minimum number of frames required for a chase to be occuring must be lowered, which again will increase the number of false alarms.

To avoid having to lower the minimum number of frames a chase must be occuring, the area covered by the camera must be increased. There are two ways of doing this: (*i*) mounting the camera high above the scene, and (*ii*) using a wide-angle lens. (*i*) means a very high resolution CCD must be used to not miss any important details, and this means expensive cameras. It will also be difficult to find high enough ceilings or mounting points in many areas of interest. (*ii*) is attractive and often used in indoor surveillance. However, occlusions will be frequent along the edges of the captured images, due to the short focal length.

Using a wide-angle lens will usually be the most practical solution. The issue of the occlusions of people and their feet is an important area for future research.

6.2.5 Recognizing <Not Standing>

The algorithm presented for recognizing <Not Standing> is simple and computationally inexpensive, and as the above algorithm it is independent of camera angles and rotations.

Figure 5.3(c) is an example of what makes recognition of <Not Standing> difficult. The size, height, and aspect ratio of the bounding box is not very different from that of a standing person leaning slightly to the right. A person standing can be seen in figure 5.3(a). Even the contour looks like that of a standing person (straight body and slightly separated legs). An accurate posture analysis could determine whether the person is standing or not, but such an algorithm requires high resolution images with little noise. That is not the normal case with surveillance cameras.

It will normally not be suitable for use outdoors. It can work, as seen in figure 5.5(c) and 5.5(d), but it is not in general suited for outdoor environments. This is because in most outdoor areas controlling the lighting is difficult, and so the cast shadow will often not be visible to the camera.

As with the <Chasing> algorithm, this algorithm will not work well in crowded areas. This is because a shadow cast by some individual will often be occluded by other individuals. Even if the shadow is not occluded, another shadow cast by another individual may overlap with the first, causing problems with the feature extraction.

It is quite difficult to accurately find which shadows were cast by which individual, so a rough approximation is used instead. This approximation will not work in a crowded environment, due to overlapping shadows cast by different individuals. This can be worked around by detecting if a blob corresponds to more than one individual, and if so then avoid attempting to recognize <Not Standing>.

Considering how much information cast shadows convey about the geometry of the scene from a single view, much effort should be put into researching how to accurately extract cast shadows, and how to use this information to recognize human behavior.

6.3 Future Work

The scope of this thesis has been very wide, but hopefully the algorithms and implementation presented here can provide a basis for further research into the area of automated security and behavior recognition. The following sections will discuss the directions that seem particularly interesting.

6.3.1 Use of Vanishing Points

The use of vanishing points for finding the feet and head of standing individuals should be further investigated. In particular, it seems interesting to use the vertical vanishing point to find an estimate for the head position in the image, and then use an active shape tracker based on that position (as in the Reading People Tracker). This would make it possible to track partly occluded individuals. The error in the estimated feet position caused by leaning while running should also be investigated.

6.3.2 Use of Single View Metrology

Single view metrology seems like a very appealing way of getting 3D information with a minimal knowledge of the camera configuration. As has been shown in this thesis, this can be used to measure world positions and distances accurately. This information could be used to recognize new behaviors, such as <Attacking> and <Repulsing>.

6.3.3 Use of Cast Shadows, Reflections, Symmetries and Repetitions

Cast shadows convey a lot of geometrical information about the scene and subjects under surveillance, along with reflections, symmetries and repetitions as mentioned in [7]. Cast shadows could be used to get the height of an individual, and therefore to recognize <Not Standing>. Reflections could be used to get information about an object that would otherwise be occluded. Reflections are effectively an alternate view of the scene.

6.3.4 Behaviors

Recognizing the behaviors listed in section 3.2 should be researched, as these cover most of the behavior that is usually regarded as unwanted in public areas. The algorithm in [31] for recognizing <Fighting> should be implemented and tested. Using Case-Based Reasoning to recognize human behavior does not seem to have been attempted much, so this would be very interesting for the more complex behaviors.

It is interesting to attempt to use contextual information in the behavior recognition process. This will decrease the number of false alarms, such as falsely detecting <Chasing> when people are running to catch the train, and so on.

6.3.5 Implementation

The implementation presented in this thesis have some shortcomings. The background model is very simple; a more complex model should be used to make outdoors surveillance possible. The tracker is also very simple. The author did attempt to make use of the Reading People Tracker, but getting it to compile and run on the author's development platform was not possible in the time frame available. Using the Reading People Tracker should be further investigated. Using multiple cameras in the tracking process seems to be the best bet for avoiding frequent occlusions, and therefore this should be implemented.

The simple user interface used in the current implementation is not scalable. A framework such as TrollEye (TrollHetta AS) could be used to improve this. TrollEye provides the user with standard surveillance functionality, such as camera selection and the setting of ROIs. A better GUI would be particularly useful for calibrating using single view metrology. The current way of calculating the homography and vanishing points is somewhat awkward.

Appendix A

Shadow Removal Using Color

Note: the author is not an expert in the Matlab language, so the following code could be sub-optimal. This algorithm has also been implemented in optimized C++, but the Matlab code is shown here as it is more compact and easier to read. This algorithm is described in [10].

Code

```
data1 = [   'reference.bmp       ';
            'screen1138121354.bmp';
%           ... more files here...
            'screen1138121370.bmp'];

files = data1;
diffthreshold = 8;
lumlowthreshold = 0.01; % >= 0
lumhighthreshold = 0.99; % must be < 1
huethreshold = 0.2;
satthreshold = 0.5;
minregionsize = 20;

iref = imread(deblank(files(1,:)));
irefhsv = rgb2hsv(iref);
irefg = double(rgb2gray(iref));
iref = double(iref)/255;

for f = 2:size(files,1)
    i = imread(deblank(files(f,:)));
    ihsv = rgb2hsv(i);
    ig = double(rgb2gray(i));
    i = double(i)/255;
```

```
    % Subtract background and remove one-pixel thick noise
    idifft = abs(ig - irefg) >= diffthreshold;
    ieroded = bwmorph(idifft, 'erode');

    % Removing shadows
    SP =    ieroded(:,:) ...
            & (abs(ihsv(:,:,1)-irefhsv(:,:,1)) <= huethreshold) ...
            & ((ihsv(:,:,2)-irefhsv(:,:,2)) <= satthreshold) ...
            & ((ihsv(:,:,3)./irefhsv(:,:,3)) <= lumhighthreshold);
    iseg = ieroded & ~SP;

    % Remove small blobs
    ilab = bwlabel(iseg);
    for l = 1:size(ilab,1)
        [r,c] = find(ilab==l);
        if length(r) > 0 && length(r) < minregionsize,
            iseg = iseg & ~bwselect(iseg,c,r);
        end
    end

    % "Reconstruct" by applying 2 dilations and 1 erosion
    iseg = bwmorph(iseg, 'dilate', 2);
    iseg = bwmorph(iseg, 'erode');

    if menu('','Next', 'Exit') == 2, break, end
end
```

Appendix B

Shadow Removal Using Texture

Note: the author is not an expert in the Matlab language, so the following code could be sub-optimal. This algorithm has also been implemented in optimized C++, but the Matlab code is shown here as it is more compact and easier to read. This algorithm is described in [19].

Code: function converting from rgb to hsL

```
% This is the file rgb2hsL.m
function [hsL] = rgb2hsL(rgb)

    maxrgb = max(rgb,[],3);
    minrgb = min(rgb,[],3);
    HSV = rgb2hsv(rgb);
    H = HSV(:,:,1) .* 360;
    S = (maxrgb - minrgb);% ./ maxrgb;

    hsL(:,:,1) = cosd(H) .* S;
    hsL(:,:,2) = sind(H) .* S;
    hsL(:,:,3) = (maxrgb + minrgb) .* 0.5;
```

Code: main program

```
data1 = [   'reference.bmp       ';
            'screen1138121354.bmp';
%           ... more files here...
            'screen1138121370.bmp'];

files = data1;
diffthreshold = 8 ./ 255.0;
shadowthreshold = 0.8;
```

```
minregionsize = 30;
N = 3;           % (2N+1)^2 ncc window
MN = (N*2+1).^2; % area of window

iref = double(imread(deblank(files(1,:)))) ./ 255;
irefg = rgb2gray(iref);
irefhsl = rgb2hsL(iref);

for f = 2:size(files,1)
    i = double(imread(deblank(files(f,:)))) ./ 255;
    ig = rgb2gray(i);
    ihsl = rgb2hsL(i);

    % Subtract background and remove one-pixel thick noise
    idifft = abs(ig - irefg) >= diffthreshold;
    ieroded = bwmorph(idifft, 'erode');

    % Find shadow pixels
    cncc = zeros(size(ig), 'double');
    hs_a = sum(irefhsl(:,:,1:2).*irefhsl(:,:,1:2), 3);
    dot_a = max(hs_a, 0) + irefhsl(:,:,3).^2;
    hs_b = sum(ihsl(:,:,1:2).*ihsl(:,:,1:2), 3);
    dot_b = max(hs_b, 0) + ihsl(:,:,3).^2;
    hs_ab = sum(irefhsl(:,:,1:2).*ihsl(:,:,1:2), 3);
    dot_ab = max(hs_ab, 0) + irefhsl(:,:,3).*ihsl(:,:,3);

    for x = N+1:size(ig,1)-N
        for y = N+1:size(ig,2)-N
            if ihsl(x,y,3) < irefhsl(x,y,3)
                luma=0; lumb=0; suma=0; sumb=0; sumab=0;
                for i = x-N:x+N
                    for j = y-N:y+N
                        luma = luma + irefhsl(i,j,3);
                        lumb = lumb + ihsl(i,j,3);
                        suma = suma + dot_a(i,j);
                        sumb = sumb + dot_b(i,j);
                        sumab = sumab + dot_ab(i,j);
                    end
                end
                avga = luma ./ MN;
                avgb = lumb ./ MN;
                vara = suma - MN.*avga.*avga;
                varb = sumb - MN.*avgb.*avgb;
                cncc(x,y) = (sumab - MN.*avga.*avgb) ./ realsqrt(vara.*varb);
            else
```

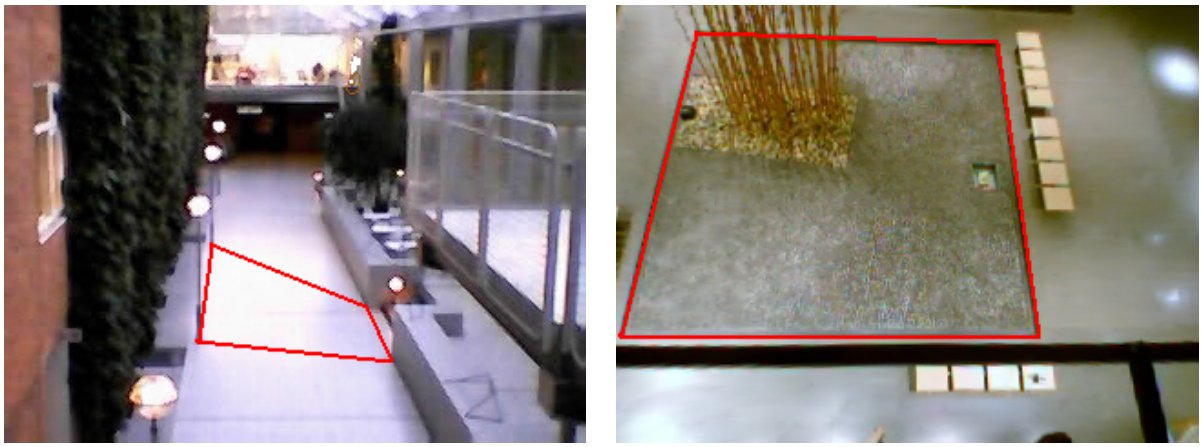
```
                cncc(x,y) = 0;
            end
        end
    end

    iforeground = (cncc < shadowthreshold) & ieroded;

    % Remove small blobs
    iseg = iforeground;
    ilab = bwlabel(iseg);
    for l = 1:size(ilab,1)
        [r,c] = find(ilab==l);
        if length(r) > 0 && length(r) < minregionsize,
            iseg = iseg & ~bwselect(iseg,c,r);
        end
    end

    % "Reconstruct" by applying 2 dilations and 1 erosion
    iseg = bwmorph(iseg, 'dilate', 2);
    iseg = bwmorph(iseg, 'erode');

    if menu('','Next', 'Exit') == 2, break, end
end
```

Appendix C

Computation of Homography

The following Matlab code shows how to compute the image plane to reference plane homography using at least four sets of point correspondences, as described in [6, 7]. (x_i, y_i) are the image plane coordinates, and (X_i, Y_i) are the reference plane coordinates.

Figure C.1 shows images of the two scenes mentioned in the Matlab code. The image plane coordinates are the corners of the red polygon. The reference plane coordinates used here are not accurately measured, but only used for testing/demonstration. Also note that the selection of points is not ideal in both of the images. The selection in figure C.1(a) is particularly poor, as it covers a very small portion of the reference plane. This means that some extrapolation will occur when applying the homography to the 2D image plane coordinates of people in the scene. As already mentioned, these are just examples.



(a) Case 1: glassgaarden

(b) Case 2: realfagbygget

Figure C.1: *Two different scenes used in testing/demonstrating the computation of the image plane to reference plane homography.*

Code

```
w = 320; h = 240; % image width and height
%%% -------------------------
% Scene 1: glassgaarden
% x1 = 213; y1 = h-194;
% x2 = 200; y2 = h-167;
% x3 = 106; y3 = h-184;
% x4 = 113; y4 = h-132;
% x5 = 181; y5 = h-130;
% X1 = 5; Y1 = -1.5;
% X2 = 5; Y2 = 1.5;
% X3 = 0; Y3 = 0;
% X4 = 0; Y4 = 8;
% X5 = 5; Y5 = 8;
%%% -------------------------
% Scene 2: Realfagbygget
x1 = 44; y1 = h-16;
x2 = 207; y2 = h-22;
x3 = 230; y3 = h-181;
x4 = 2; y4 = h-181;
X1 = 0; Y1 = 9;
X2 = 9; Y2 = 9;
X3 = 9; Y3 = 0;
X4 = 0; Y4 = 0;
%%% -------------------------

A = [
    x1, y1, 1,  0,  0,  0,  -x1*X1, -y1*X1, -X1;
    0,  0,  0,  x1, y1, 1,  -x1*Y1, -y1*Y1, -Y1;
    x2, y2, 1,  0,  0,  0,  -x2*X2, -y2*X2, -X2;
    0,  0,  0,  x2, y2, 1,  -x2*Y2, -y2*Y2, -Y2;
    x3, y3, 1,  0,  0,  0,  -x3*X3, -y3*X3, -X3;
    0,  0,  0,  x3, y3, 1,  -x3*Y3, -y3*Y3, -Y3;
    x4, y4, 1,  0,  0,  0,  -x4*X4, -y4*X4, -X4;
    0,  0,  0,  x4, y4, 1,  -x4*Y4, -y4*Y4, -Y4;
%    x5, y5, 1,  0,  0,  0,  -x5*X5, -y5*X5, -X5;
%    0,  0,  0,  x5, y5, 1,  -x5*Y5, -y5*Y5, -Y5;
    ];

% Using singular value decomposition to find eigenvalues and -vectors.
[U, S, V] = svd(A);

% Pick the eigenvector corresponding to the smallest eigenvalue.
H(1,1:3) = V(1:3,9);
```

```
H(2,1:3) = V(4:6,9);
H(3,1:3) = V(7:9,9)

% Testing by projecting the image points with known world locations.
P1=H * [x1,y1,1]';
P2=H * [x2,y2,1]';
P3=H * [x3,y3,1]';
P4=H * [x4,y4,1]';

% Divide by scaling factor (P1-P4 are in homogenous coords).
PP1=P1./P1(3);
PP2=P2./P2(3);
PP3=P3./P3(3);
PP4=P4./P4(3);

disp('Reference plane coordinates:');
disp([PP1 PP2 PP3 PP4])
```

Bibliography

[1] S. Intille F. Baird L. Cambell Y. Irinov C. Pinhanez A. Bobick, J. Davis and A. Wilson. Kidsroom: Action recognition in an interactive story environment. *Technical Report 398, M.I.T. Perceptual Computing*, 1996.

[2] B Brumitt B. Meyers M. Czerwinski A. Shafer, J. Krumm and D.Robbins. The new easyliving project at microsoft. In *Proc. DARPA/NIST Smart Spaces Workshop*, 1998.

[3] Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59, 1994.

[4] J. K. Aggarwal and Q. Cai. Human motion analysis: a review. *Comput. Vis. Image Underst.*, 73(3):428–440, 1999.

[5] D. Beymer and K. Konolige. Real-time tracking of multiple people using stereo. In *Proc. IEEE Frame Rate Workshop, 1999*, 1999.

[6] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *Int. J. Comput. Vision*, 40(2):123–148, 2000.

[7] Antonio Criminisi. *Accurate visual metrology from single and multiple uncalibrated images*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[8] Antonio Criminisi. Single-view metrology: Algorithms and applications. In *Proceedings of the 24th DAGM Symposium on Pattern Recognition*, pages 224–239, London, UK, 2002. Springer-Verlag.

[9] Antonio Criminisi, Andrew Zisserman, Luc Van Gool, Simon Bramble, and David Compton. A new approach to obtain height measurements from video.

[10] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti. Improving shadow suppression in moving object detection with hsv color information.

[11] Bremond F. Thonnat M. Cupillard, F. Group behavior recognition with multiple cameras. In *Applications of Computer Vision, 2002. (WACV 2002). Proceedings. Sixth IEEE Workshop on*, pages 177–183, 2002.

[12] Bremond F. Thonnat M. Avanzi A. Cupillard, F. Video understanding for metro surveillance. In *Proceedings of the 2004 IEEE Internation Conference on Networking, Sensing and Control*, 2004.

[13] M. Shah D. Ayers. Monitoring human behavior from video taken in an office environment. *Image and Vision Computing, Vol. 19*, 2001.

[14] F. Devernay and O. Faugeras. Automatic calibration and removal of distortion from scenes of structured environments. In *Proceedings of SPIE*, volume 2567, San Diego, CA, July 1995.

[15] R. Romano L. Lee E. Grimson, C. Stauffer. In *Using Adaptive Tracking to Classify and Monitoring Activities in a Site*, pages 22–29, 1998.

[16] Olivier Faugeras. *Three-dimensional computer vision: a geometric viewpoint.* MIT Press, Cambridge, MA, USA, 1993.

[17] L. M. Fuentes and S. A. Velastin. People tracking in surveillance applications. In *2nd IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2001)*, Hawaii, 2001.

[18] S.A. Fuentes, L.M. Velastin. From tracking to advanced surveillance. In *Image Processing, 2003. ICIP 2003. Proceedings*, Departamento de Opt. y Fisica Aplicada, Valladolid Univ., Spain, 2003.

[19] Daniel Grest, Jan-Michael Frahm, and Reinhard Koch. A color similarity measure for robust shadow removal in real time. In *VMV*, pages 253–260, 2003.

[20] Ismail Haritaoglu, Davis Harwood, and Larry S. David. W4: Real-time surveillance of people and their activities. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):809–830, 2000.

[21] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge University Press, New York, NY, USA, 2000.

[22] M. Loughlin J. Rehg and K. Waters. Vision for a smart kiosk. *Computer Vision and Pattern Recognition*, 1997.

[23] Julio Cezar Silveira Jacques Jr, Cl´audio Rosito Jung, and Soraia Raupp Musse. Background subtraction and shadow detection in grayscale video sequences.

[24] Y. Kuno, T. Watanabe, Y. Shimosakoda, and S. Nakagawa. Automated detection of human for visual surveillance system.

[25] D. Liebowitz and A. Zisserman. Metric rectification for perspective images of planes. In *CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 482, Washington, DC, USA, 1998. IEEE Computer Society.

[26] Alan J. Lipton, Hironobu Fujiyoshi, and Raju S. Patil. Moving target classification and tracking from real-time video. In *WACV '98: Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV'98)*, page 8, Washington, DC, USA, 1998. IEEE Computer Society.

[27] Dr. Alan J. Lipton. Intelligent video surveillance in crowds, 2004. [Online white paper; `http://www.objectvideo.com/objects/pdf/whitepapers/FlowControl.pdf`; accessed 02-May-2006].

[28] S.A. Velastin L.M. Fuentes. Assessment of image processing as a means of improving personal security in public transport. In Nikos Paragios Paolo Remagnino, Graeme A. Jones and Carlo S. Regazzoni, editors, *Video-Based Surveillance Systems, Computer Vision and Distributed Processing*, pages 159–166. Kluwer Academic Publishers, 2001.

[29] S.A. Velastin L.M. Fuentes. Foreground segmentation using luminance contrast. In *WSES International Conference on Speech, Signal and Image processing, SSIP 01*, Malta, 2001.

[30] Erik Sundnes Løvlie. Tdt4725 image processing, depth study: Recognizing violent behavior. Department of computer and information science, the Norwegian University of Science and Technology, 2005.

[31] F. Bremond N. Moenne-Loccoz and M. Thonnat. Recurrent bayesian network for the recognition of human behaviours from video. In *ICVS2003*, 2003.

[32] Monique Thonnat Nathanael Rota. Video sequence interpretation for visual surveillance. In *Third IEEE International Workshop on Visual Surveillance*, page 59, 2000.

[33] T. Olson and F. Brill. Moving object detection and event recognition algorithms for smart cameras. In *Proc. DARPA Image Understanding Workshop 1997*, pages 159–175, 1997.

[34] G.R. Patterson, J.B. Reid, and J.M. Eddy. A brief history of the oregon model. In G.R. Patterson, J.B. Reid, and J. Snyder, editors, *Antisocial behavior in children and adolescents: A developmental analysis and model for intervention.*, pages 3–21, Washington DC, US, 2002. American Psychological Association.

[35] CAVIAR Project. Caviar test case scenarios, 2004. [`http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1/`; accessed 17:17, 30-May-2006].

[36] Ian D. Reid and A. North. 3d trajectories from a single viewpoint using shadows. In John N. Carter and Mark S. Nixon, editors, *BMVC*. British Machine Vision Association, 1998.

[37] Arnstein Ressem. Segmentering, følging og klassifisering i videoanalyse av trafikk. *Hovedoppgave, Institutt for matematiske fag, NTNU*, 2000.

[38] Elena Salvador, Andrea Cavallaro, and Touradj Ebrahimi. Shadow indentification and classification using invariant color models. 2001.

[39] T. Kato H. Inokuchi S. Sato, K. Maeda. Cad-based object tracking with distributed monocular camera for security monitoring. In *Proceedings of the Second CAD-Based Vision Workshop, 1994*, pages 291–297, 1994.

[40] Claudette Cédras; Mubarak Shah. Motion-based recognition a survey. *Image Vision Computing*, 13, 1995.

[41] N. T. Siebel and S. J. Maybank. Real-time tracking of pedestrians and vehicles. In *2nd IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2001)*, Hawaii, 2001.

[42] Nils T. Siebel. *Design and Implementation of People Tracking Algorithms for Visual Surveillance Applications*. PhD thesis, Department of Computer Science, The University of Reading, 2003.

[43] Maybank S.J. Siebel N.T. The advisor visual surveillance system. In *Proceedings of the workshop "Applications of Computer Vision" (ACV'04)*, pages 103–111, May 2004.

[44] Micheloni C. Chiavedale C. Snidaro, L. Video security for ambient intelligence. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 35(1), 2005.

[45] W.E.L. Stauffer, C.; Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Iss 1999, vol. 2*, pages –252 Vol. 2. IEEE Computer Society, 1999.

[46] M. Harville J. Woodfill T. Darell, G. Gordon. Integrated person tracking using stereo, color, and pattern detection. *Computer Vision and Pattern Recognition*, 1998.

[47] A. Erkan P. Lewis C.Powers C. Qian Terry Boult, R. Micheals and W. Yin. Frame-rate multi-body tracking for surveillance. Vision and Software Technology Lab, EECS Department Lehigh University, 1998.

[48] Wikipedia. Happy slapping — Wikipedia, the free encyclopedia, 2006. [`http://en.wikipedia.org/w/index.php?title=Happy_slapping&oldid=50882555`; revision 12:44, 30-April-2006].

[49] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):780–785, 1997.

[50] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.