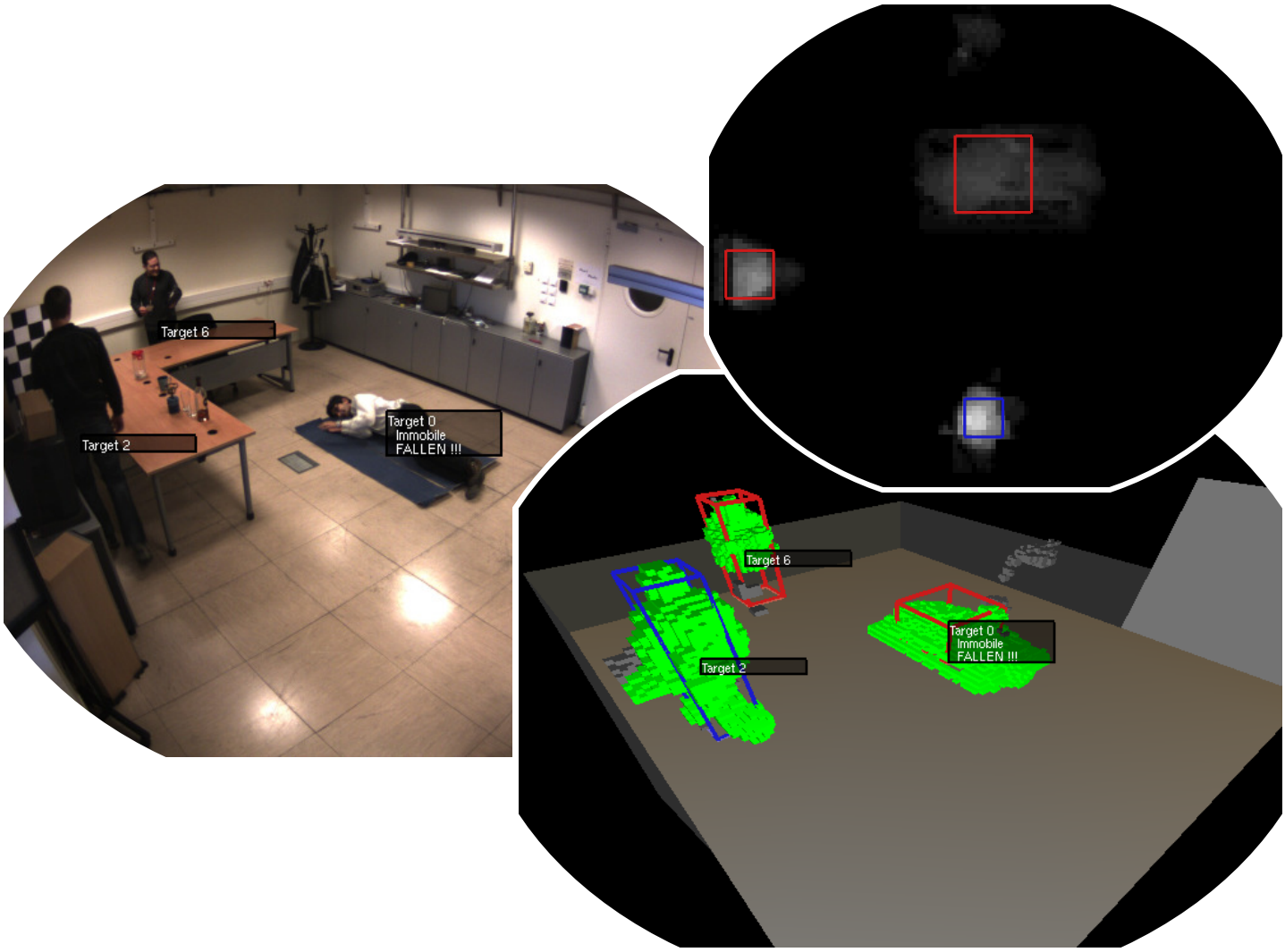
Multi-Camera Person Tracking using Particle Filters

based on Foreground Estimation and Feature Points



AALBORG UNIVERSITY

Department of Electronic Systems
Vision, Graphics, and Interactive Systems

Master Thesis - 2009/2010

Martin Andersen
Rasmus Skovgaard Andersen

AALBORG UNIVERSITY

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7
9220 Aalborg Ø
Telephone 99 40 86 00
http://es.aau.dk/

Title:
Multi-Camera Person Tracking using Particle Filters based on Foreground Estimation and Feature Points

Project term:
1. september 2009 - 3. June 2010

Project group:
10gr1022

Group members:
Martin Andersen
Rasmus Skovgaard Andersen

Supervisor:
Zheng-Hua Tan

Co-supervisor:
Aristodemos Pnevmatikakis

Copies: 5

Pages: 109

Appendix pages: 20

Attachments: 1 CD

Finished: June 3, 2010

Synopsis:

Much research have in resent years been directed at sensing the presence and state of people and many possible applications exist. One application whose importance is continuously increasing with the average age of the western societies increases is assistive living environments for elderly.
In this project a person tracking system for multi-camera environments is developed with this application in mind. Tracking is based on two different cues or modalities: Foreground and feature points. These compliment each other well since foreground is present whenever a person is moving, while feature points initialised on a person can be tracked indefinetly if he is stationary.
Foreground is first found in 2D for each camera by estimating a background model using Gaussian Mixture Models. Then foreground from all cameras are combined yielding one single voxel-based 3D foreground. Feature points are initialised for each camera on persons and tracked individually in 2D using the KLT-tracker. Finally, layered sampling is applied to fuse information from foreground estimation and feature points into one particle filter for each present person.
The system is implemented to run on one computer using video recordings, and a part of the system is implemented to run live distributed between multiple computers. Tests on the CLEAR 2007 data set prove that the combination of the two modalities provide better performance than a tracking system based solely on one of them.

Preface

This report documents the Master Thesis project entitled *Multi-Camera Person Tracking using Particle Filters based on Foreground Estimation and Feature Points*. The project was carried out during the 9'th and 10'th semester of the specialisation *Vision, Graphics, and Interactive Systems* under the Department of Electronic Systems at Aalborg University in 2009/2010. Besides this report, a paper has also been written in connection with the project. This has recently been accepted for the EUSIPCO 2010 conference [28], and a draft of it is attached here as Appendix E.

The report is divided in to five parts plus appendices: *Introduction*, *Modalities*, *Tracking System*, *Implementation*, and *Evaluation*. The first part motivates the project and concludes in a problem formulation and a hypothesis that point out the direction of the project. Analysis of possible solutions and design of our system are contained in the following two parts, and the fourth part describes our C++ implementation. The last part evaluates the performance of our system though a number of different tests and concludes on the project as a whole.

References to secondary literature sources are made using the syntax [number]. The number refers to the alphabetically sorted bibliography found at the end of the report, just before the appendices.

We would like to thank our supervisor at Aalborg University Zheng-Hua Tan for proposing this intriguing project and for establishing contact with leading scientists within the area at the Athens Information Technology (AIT) [1]. We would also like to thank our co-supervisor from AIT Aristodemos Pnevmatikakis and phd. stud. Nikos Katsarakis both for inspiring and assisting us in the project and for showing great hospitality during our stay in Athens, as well as for making it possible to test our system on the CLEAR data set [19].

A CD is attached to this report which includes:

- Source code of the developed program. Build instructions and dependencies are listed in Appendix C.
- Video files illustrating the performance of the developed system.
- PDF file of this report.

Aalborg - June 3, 2010

Martin Andersen

Rasmus Skovgaard Andersen

Contents

Part I

Introduction

Contents

The project is in this first part motivated by analysing the need for and relevant applications of a person tracking system. Previously developed algorithms and systems for person tracking are also examined to provide the basis for developing an efficient approach that has not previously been proposed. This leads to a problem formulation for the project in Chapter 3.

Chapter 1

Motivation

Video surveillance in general has a large variety of potential applications including traffic control, surveillance of automated production lines and surveillance of people. The latter covers many different area and situations where the common denominator is to allow an automatic system to sense the presence and state of one or multiple people, and much interest has in resent years been directed at this area. Applications that have received extensive research include surveillance [61], assistive living environments [35, 72], and human-machine interfaces [66, 76].

One widely used type of video surveillance systems for monitoring people are security systems based on single cameras. These typically record video either continuously in a time-limited circular buffer or based on events, where events can be detected using simple motion detection algorithms. Such systems have proven to be very useful, when security officers after an event such as a robbery are investigating what happened and who were responsible. In many situations it will, however, be an advantage if an automated system more precisely can tell what is happening, and only sound off an alarm if specific events or events that are out of the ordinary occur. This is the case for both security systems, assistive living environments etc. Common for all applications is that the more precisely an automated system are able to detect what is happening, the less personnel will be required.

In this project we focus on the application of assistive living environments for elderly. In the western societies in general, people live longer. This means that the average age of people and the amount of retired elderly are increasing. In Denmark, the number of citizens above 65 is expected to increase by 60% from 2010 to 2042 while the population in total only increases by 9% [20]. This means that the share of the population above 65 will increase 16% in 2010 to 25% in 2042. Thus, systems that can increase the living standard of elderly while reducing the need for assisting human service personnel is of major interest, and the demand will only increase in the years to come.

To increase the capability of elderly to live in their own homes with minimal assistance, both emergency and cognitive care are of significant importance. Emergency situations can easily arise for weak elderly that e.g. might not be able to reach a telephone and call for help after an accident. Cognitive care is important to become aware of behavioural changes that might be caused by conditions such as depression or dementia, that are not easily discovered during short consultations at a doctor. Therefore, we focus on fall detection and activity monitoring for the elderly. This is done by tracking the position and determining the mobility of persons present in the scene under surveillance in real-time and to reason about their body posture.

The mobility of a person can be classified as either *mobile* or *stationary*, and it can be used to measure the activity of a person and, when combined with the location estimate, detecting abnormal behaviour. The body posture can be classified as either *standing*, *sitting* or *fallen* and can be used both to detect abnormal behaviour as well as detecting accidents when combined with the location estimate.

While tracking systems based on one single camera suffice for some situations, they also have severe limitations. For instance, persons can very easily become temporarily occluded behind objects or other persons, and any position can only be determined relative to that camera - that is, in two dimensions. For automated tracking systems it is therefore a major advantage to have more information than a single camera. This information can besides additional cameras be provided by other passive (i.e. non-intruding) or active sensors. Passive sensors can for instance be microphones or infrared sensors while active sensors can be laser range finders or ultrasound sensors [30].

Active sensors and to a certain degree microphones are well-suited to provide information about the location of people. Cameras do, however, provide more information about what is happening to the people besides their location. Therefore, we base our system on multiple cameras that in combination are used to track present people and reason about their mobility and body posture.

1.1 Setup and Data Sets

When developing new algorithms, it is always essential to have the ability to produce test results that are comparable with existing algorithms. Within vision based multi-camera tracking for indoor use, the de-facto standard setup is illustrated in Figure 1.1. A total of 5 cameras are used, and 4 of these placed in top corners while the last is placed in the middle of the ceiling pointing down.

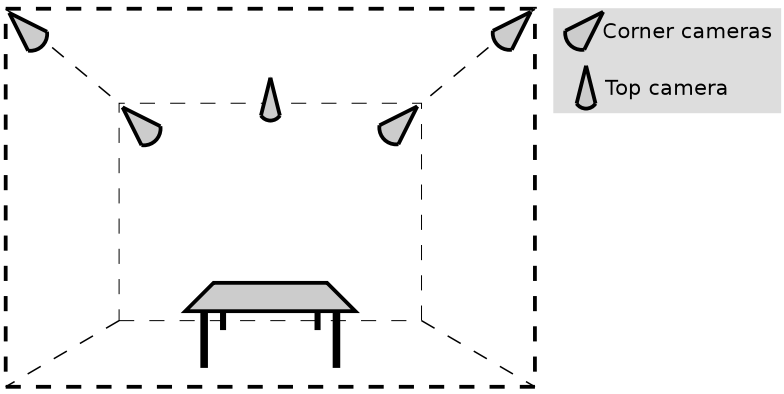


Figure 1.1: *Camera setup.*

1.1.1 Development Data Set and Setup

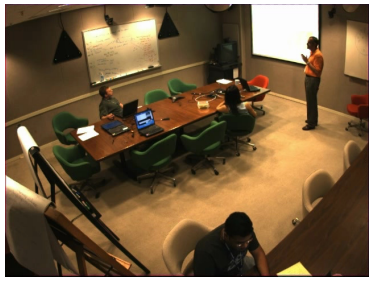
To assist in the development process, a test setup was established in a laboratory at Athens Information Technology (AIT) [1], that matches the setup shown in Figure 1.1. The resolution of the cameras are 1600×1200 for the four corner cameras and 1024×768 for the top camera

and all cameras can record at 15 fps. An image from one of the corner cameras is shown in Figure 1.2a. Two sequences were recorded to be used for development of the system; one of 1:40 minutes and one of 6:23 minutes. These are henceforth denoted the short/standard development data set, respectively. The sets show up to 3 (for the short) and 4 (for the standard) persons simultaneously moving around and/or remaining immobile for some time. To test the posture reasoning, a person is falling at 3 occasions and sitting at 6 occasions in the standard development data set. In the remainder of this report, the standard development data set is always used unless otherwise specified.

When recording from the development setup, each camera is connected to separate, networked computers, and this setup can also be used to test the live performance of the system.

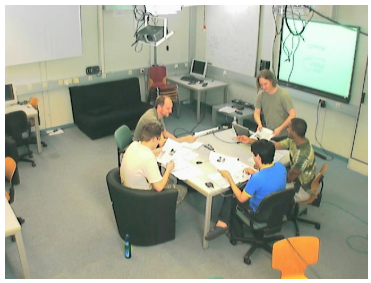
(a) AIT (for development)

(b) AIT



(c) IBM

(d) ITC



(e) UKA

(f) UPC

Figure 1.2: *Figure (a) is a frame from the data set used for development of the system, and the remaining figures are frames from each of the locations included in the CLEAR 2007 data set, which is used for testing.*

1.1.2 CLEAR Test Data Set

While the development setup allows both the recording of an appropriate development data set as well as live system tests, it is not suited to produce results that can be compared to performance of previously developed systems. For this purpose, the CLEAR 2007 data set is chosen. This data set originates from an international workshop, known as the *2007 CLEAR evaluation and workshop* [19]. It contains 40 video recordings of meeting rooms equipped with 5 cameras as illustrated in Figure 1.1, and a number of background frames without any persons present are available for each recording. The recordings are from five locations, so that 8 recordings exist from each location, and the length of a recording is approximately 5 minutes. In each recording, people are moving around and/or sitting at a table. An image from each location is shown in Figure 1.2.

The CLEAR data set allows comparison of different tracking systems because the location of all persons in the scene are hand annotated with a precision of 5-8 cm. For the CLEAR 2007

workshop, metrics defining how to measure the performance of a system was also introduced, and a total 7 different systems participated in the original workshop. The performance of these are now available for comparison, and the CLEAR 2007 data set has since the workshop to a large degree become the de-facto standard for comparison of tracking systems.

The CLEAR evaluation metrics is described in the following section. The performance of the system developed in this project are measured and compared to the participating systems in the CLEAR workshop in Chapter 14.

A limitation with the CLEAR data set is that all recordings are from meeting rooms and no persons are falling. For this reason, the development data set described in Section 1.1.1 must be used to test posture reasoning. Also, the time consumption of the tracking systems are not reflected in the CLEAR metrics. The development data set is therefore also used to measure the time consumption of the developed system, with the prospect of real-time execution in mind. These results are documented in Chapter 13.

1.2 The CLEAR Evaluation Metrics

The metrics used for evaluation of the system are from the CLEAR 2007 Evaluation Workshop [19], and combines four different measurements [9]:

- **Misses** (m)**:** For the CLEAR evaluation an object is missed if it was not tracked within 50 cm accuracy.
- **False positives** (fp)**:** Note that a track that is more that 50 cm wrong is also marked as a false positive.
- **Mismatches** (mme)**:** When a track belonging to an object switches to another object this is counted as one mismatches.
- **Position error** (d^i)**:** The position error d^i of the matched object i. Note that because an object have to be tracked within 50 cm to be declared "matched", a random guess will have mean(d^i) =25 cm. Additionally it is worth noticing that the ground truth of the CLEAR data set is hand annotated with a precision of 5-8 cm.

These four measurements are combined in to two metrics; the Multiple Object Tracking Precision (MOTP) and the Multiple Object Tracking Accuracy (MOTA):

$$\text{MOTP} = \frac{\sum_{i,n} d_n^i}{\sum_t c_n} \tag{1.1}$$

$$\text{MOTA} = 1 - \frac{\sum_n (m_n + fp_n + mme_n)}{\sum_t g_n} \tag{1.2}$$

where:
n is the frame number,
c_n is the number of matches in frame n, and
g_n is the correct number of persons present in frame n.

The MOTP shows the precision of the tracker system when it is able to track a person correctly. In contrast, the MOTA value gives a measure of the trackers configuration errors, consisting of

false positives, misses and mismatches. It is worth noting that the number of mismatches are only counted once per error, while the number of false positives and misses are calculated in each frame. The number of mismatches will therefore contribute very little to the MOTA score.

Chapter 2

Existing Systems

Before developing the person tracker it is important to investigate existing state-of-the-art tracking systems. In this chapter will give an overview of existing tracking methods before the tracking methods for this project will be discussed and decided up on. This will be done by in Section 2.1 having a look at different modalities that are used for tracking and in Section 2.2 the existing combination of different modalities are discussed and in Section 2.3 the modalities used in this project will be chosen.

2.1 Modalities for Tracking

The measurement data to be used for tracking can be constituted by various different kinds of information. We define a modality in the widest sense as any kind of cue that can be used to identify and/or track people. Figure 2.1 illustrate tracking modalities that have been used in literature and their relationships, and each of them are described here:

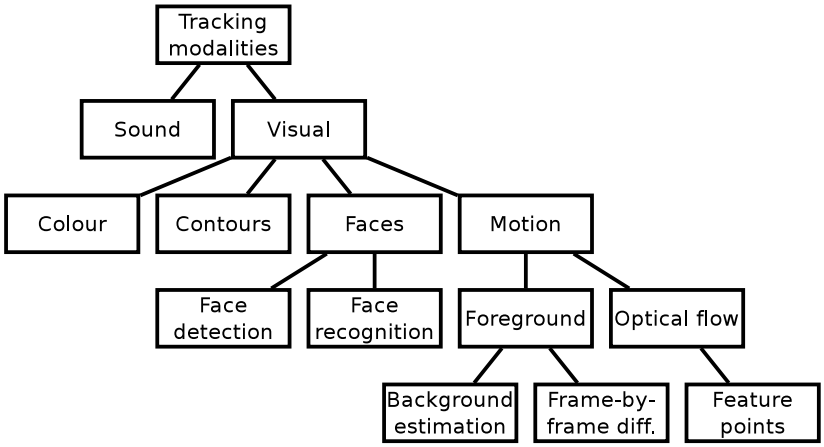


Figure 2.1: *Modalities that can be used for tracking organised in a tree structure.*

- **Sound:** People moving around and interacting with each other and/or the environment make noise. By using multiple microphones, this can be used for tracking. This is e.g. done in [64] by using stereo microphones to help tracking a head during tele-conferences.

Also, within the CLEAR evaluation and workshop [19], a number of papers use sound as one of the modalities in multimodal 3D person tracking systems [8, 46].

- **Visual:**
  - **Colour:** A very intuitive approach is to track persons using their colours. Different approaches to colour tracking exist. In [67], a colour mixture model is used based on Gaussian distributions that are fitted to the data using the Expectation Maximisation algorithm [21]. In [59], the algorithm is further developed by making the mixture model adaptive. While colour cues can in general not be used to detect, it is a common modality in the tracking stage itself [8, 17].
  - **Contours:** Contours are another intuitive approach to tracking. Using a standard edge detector it will in most cases be possible to find the outline (or contour) of persons, and these can then be tracked. Contour tracking in general is described in [2]. One problem when tracking people is that they can easily become partly occluded when passing behind objects such as chairs etc. Therefore, instead of tracking the contour of entire bodies, only the upper body can be detected [60] and tracked [8]. This also makes it possible to use contours to detect new persons, since upper bodies seen from the front or back are very similar for different people.
  - **Face detection and recognition:** Face is a modality that, disregarding false positives, is only present at real persons. Thus, it is well suited to distinguish people from moving objects, and in multimodal tracking systems face detection is also suitable for initialisation. A well-known face detection algorithm that is able to run in real-time has been developed by Viola and Jones in [75] based on a boosted cascade of simple classifiers. It is used for 2D tracking in e.g. [45, 63] and for 3D tracking in [44]. Face recognition can increase the ability to distinguish between different present persons.
  - **Motion:** In indoor environments, motion is a very strong cue indicating human presence. It has in literature been treated in many distinct ways, including:
    - **Foreground:** The simplest form of foreground detection is to take the frame-by-frame difference. While this is a valid approach that have been used e.g. in multimodal tracking systems [64], it has the limitation that it does not provide foreground evidence when the persons are stationary. Instead, a background can be estimated and compared with each frame to yield a 2D foreground [71]. By combining foreground evidence from multiple cameras, people can be tracked in 3D [17, 50].
    - **Optical flow and feature points:** Optical flow is defined as the motion of points in the scene relative to the camera projected onto the 2D image plane. By determining the optical flow of the entire image, moving objects can be separated from the background. One way to use optical flow for tracking is by identifying few particularly trackable feature points and then track these [69]. Such points is e.g. used for tracking of larger objects in 2D in [23].

2.2 Combining Modalities

To improve the performance of tracking systems, different modalities can be combined. In general, the more information that is used the better performance will be possible. Of course, more modalities also mean more complexity and a slower system. Therefore, a limited number of

modalities are preferable, especially in real-time systems. When choosing modalities, it is desired to have them complement each other without giving much redundant information. Sound does e.g. not hold information that is redundant with any visual modality.

In literature, many combinations of modalities have been attempted. A well-known paper from 2004 by Blake et. al. investigates how modalities can be efficiently combined and builds a system based on colour and motion [64]. They also design another system specifically for head tracking during tele-conferences based on colour and sound. The CLEAR 2007 evaluation and workshop compares several systems that uses multiple modalities [19, 72]. Two of the systems that provide good performances, proposed by Stiefelhagen et. al. [8] and Canton-Ferrer et. al. [17], are based on foreground and colour. The system by Stiefelhagen et. al. also utilise an upper body detector, and this system performs best of all systems the in CLEAR 2007 workshop (measured in the Multiple Object Tracking Accuracy metric, described in Section 1.2).

2.3 Choice of Modalities

As we wish to make a pure vision system, the sound modality will not be used. For visual tracking systems the foreground modality is widely used. The foreground modality in most cases gives a good indication of the location of people as stationary scene objects are efficient sorted out by this approach. This modality will be used in our tracker.

The foreground modality depend on a foreground detection algorithms, this can work either by estimating a background and comparing this to each frame, or comparing consecutive frames to one another without storing a particular model of the background. The most common approaches include:

Offline background estimation: In offline background estimation, the a model of the background is build by analysing one or more frames without foreground present. This model is then used unchanged for every frame during the foreground detection. In the simplest case the background model can be one image. This approach will work well if and only if the scene does not change significantly after the background model has been build. Also, decided background frame(s) are required.

Frame by frame subtraction: Comparing consecutive frames on a frame-by-frame basis is an efficient, fast and simple way to find motion, that is unaffected by permanent changes in the scene. However, persons that are stationary will disappear immediately. Thus, a tracker based on this approach will need to be able to handle frequent absence of foreground.

Adaptive background estimation: Instead of estimating a background model before foreground is to be detected, it can be done simultaneously by using an adaptive background model. Such an approach is able to detect stationary persons for a while, without relying on decided background frames. Also, changes in the scene such as moved objects or changed lightning conditions can be incorporated into an adaptive background model. However, a trade-off is introduced between the ability to detect persons that have been stationary for a long while, and the ability to incorporate various changes into the background.

In our scenario, objects can be moved around and sunlight can enter the scene. Since neither lightning changed nor the movement of objects can be predicted, the offline background esti-

mation are not suitable. In order not to complicate the tracker design too much, the adaptive background estimation is chosen instead because of its robustness to immobile persons.

The adaptive foreground modality can both be used for tracking and detection of new persons, but have some limitations e.g.:

Stationary persons: The foreground estimation needs to be adaptive in order to handle variations in lightning conditions and to incorporate moved scene objects. This also means that stationary persons will tend to fade into the background as time passes by.

Separating persons standing close together: When two persons are close together and their foreground merges, the foreground modality has no way to separate them.

To counter this a second modality with ability of tracking stationary people must be introduced. Faces and contours cannot be guaranteed to be present if a person is partly occluded or located in a unfortunate angle. Both colour and feature points should, however, be able to handle these situations. Both of these has the potential to keep the track when two persons pass close together. As the combination of foreground and colour has been used widely in the litterateur [8, 17, 50, 52, 64] we would like to design a combined foreground and feature point tracker.

Feature points cannot be used for detecting new persons. However, the foreground modality can handle this task on its own as persons are moving when they enters the scene, thus this is no problem. On a known person on the other hand feature points can be initialised and tracked, in this way they are associated with a specific person. With this modality two persons passing each other can be separated. The track of each feature point can be kept for as long as the viewpoint of the feature point is not changed too much. This means that stationary persons that are hard to track by the foreground modality are easily tracked using feature points. Thus feature points and foreground should be a good combination.

An intuitive way to use feature points for 3D tracking is to find the same feature point in different cameras and then triangulate. While this is definitely possible in theory [22], it will be difficult to do robustly when viewpoints of the cameras are very different. Therefore, the feature points in each camera are treated separately instead.

To our knowledge such a system has not previously been proposed in the literature. One approach that uses both foreground and feature points is proposed in “Robust Detection and Tracking of Moving Objects in Traffic Video Surveillance” by Borislav Antić et. al. [3]. They use a motion detector to guide their feature point tracker. Our idea is different from these in two ways: We wish to combine foreground and feature points so that both are used for tracking, and the tracker is to be used in a multi-camera environment.

Chapter 3

Problem Formulation

In Chapter 1 the need for systems that are able to track moving persons was discussed. Particularly the application of such a system on surveillance of elderly living alone in assisted living environments was analysed. It was concluded that reasoning about the mobility as well as detection of fallen persons would be advantageous to help the elderly live alone.

In the same chapter it was emphasised that proposed algorithms must be comparable to previously developed algorithms. Therefore the CLEAR data set was introduced [19]. Many papers published in the last couple of years proposing algorithms for tracking multiple persons have used these data sets for evaluation. The performance of the system developed here must therefore also be evaluated using the entire CLEAR data set and compared to other systems.

With the choice of the CLEAR data set for evaluation, it follows naturally that the system must be able to handle certain, typical situations occurring in these for a good performance to be possible. The CLEAR metrics define what a "good performance" means. These situations constitute in addition to the purpose of reasoning and real-time execution guide lines in the design process:

- **Entering and leaving:** People are entering and leaving the scene throughout the CLEAR videos. Additionally, in some of the videos people are present in the first frame (disregarding the background frames). All of these situations must be supported.
- **Mobility:** The CLEAR data set has both persons moving around as well as persons remaining stationary for several minutes. Thus, both cases must be supported.
- **Multiple persons:** In the CLEAR data sets, up to 5 persons are present simultaneously.
- **Precision:** The ground truth for the CLEAR data set is given with a precision of 5-8 cm. This means that a larger precision is not required for our system.
- **Real-time execution:** For a real-life implementation to be possible, the system must be able to run in real-time.
- **Reasoning:** Reasoning about the mobility and body posture of the tracked persons must be able to run simultaneausly with the person tracking.

In Chapter 2, previously developed tracking modalities and algorithms was presented, and it was

chosen to base our system on both adaptive foreground estimation and feature points initialised on each person. This leads to the following problem formulation:

Problem formulation:
How can a system be developed that based on combined information from adaptive foreground estimation and feature points, is able to track multiple persons in a multi-camera environment and reason about their mobility and body posture?

This problem formulation implicitly constructs the following hypothesis: A tracking system based on a combination of adaptive foreground estimation and feature points must provide better performance than a similar tracking system based solely on one of these modalities. Thus, this hypothesis must also be verified.

Part II

Modalities

Contents

In the problem formulation is was chosen to base the tracking system on the two modalities foreground and feature points. In this part each of these is examined closely, and algorithms are designed to make it possible to use them for person tracking in a multi-camera environment. An adaptive algorithm for foreground detection in 2D is developed in Chapter 4 which must be run for each camera, and an algorithm for combining information from all cameras into a single 3D representation of foreground is designed in Chapter 5. Existing feature point algorithms are examined in Chapter 6, and based on the well-known KLT feature point tracker, an algorithm for initialisation and tracking of feature points on persons is designed.

Chapter 4

Foreground Detection in 2D

The term *foreground* means non-stationary persons or objects in the scene. For the human visual system it is usually a trivial task to distinguish foreground from background in a video sequence. This is, however, not the case for a computer vision system, and much research has been carried out in this area. Most methods for foreground detection are based per pixel background models [31, 41, 48, 53, 65]. This approach is also chosen for our system. A pixel based method does not have any idea of general things in the scene and therefore noise and shadows must be handled afterwards. The framework for our foreground detection is shown in Figure 4.1. In Section 2.3 it was chosen to base foreground detection on adaptive background estimation, and this algorithm is designed in Section 4.1. The efficiency and performance of the algorithm are discussed in Section 4.3, and shadow and noise removal are described in Section 4.4 and 4.5, respectively. Table 4.1 at the end of the chapter shows the parameter values used as default in our system.

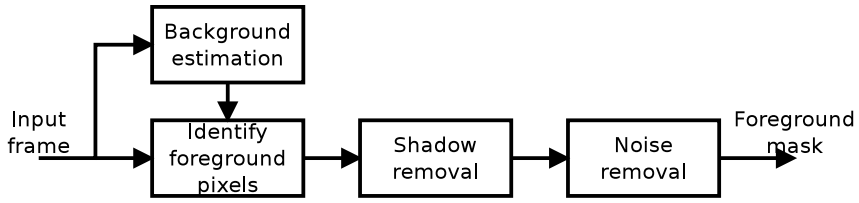


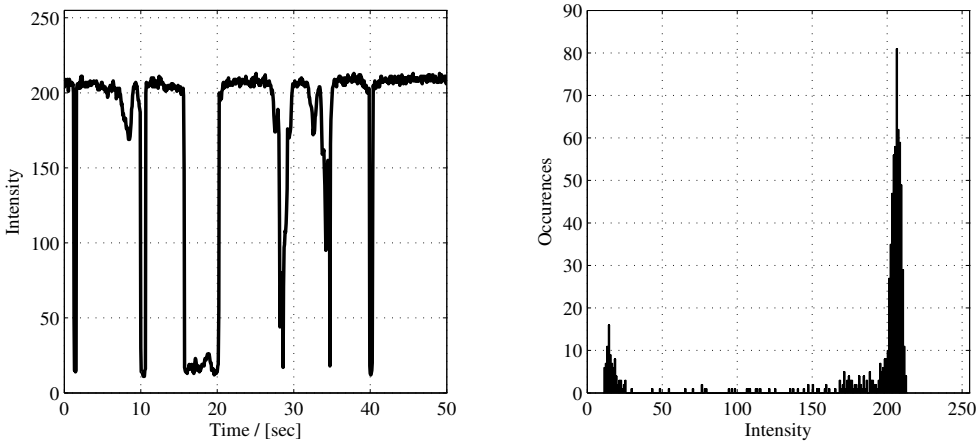
Figure 4.1: *Conceptual framework of a 2D foreground detector based on background estimation.*

4.1 Adaptive Background Estimation

Adaptive background estimation per-pixel is a commonly used technique in computer vision, and in general two approaches exist: Parametric and non-parametric. A parametric background estimator assumes that the background is distributed as a predefined distribution, whereas a non-parametric estimator can handle arbitrary distributions. On the other hand, non-parametric approaches require a huge amount of memory to store the background [54] while parametric approaches only require few parameters per pixel. Since our system must be able to run using frames from five cameras of potentially high resolutions on a single computer, the parametric approach is chosen.

4.1.1 Background Estimation using Mixture Models

The idea of a mixture model is to model each RGB pixel by several simple distributions such as Gaussian distributions. In theory, every distribution can be modelled as a Gaussian Mixture Model (GMM). However, it is only an advantage to use a parametric description if a relative small number of simple distributions suffice to give a good approximation. The idea of applying mixture models to model pixels is that only one distribution shall be used to model the background colour, while one or more additional distribution shall model the various foreground objects or persons passing in front of the background. The idea is illustrated in Figure 4.2, where red intensity values of a particular pixel from the development data set are plotted as a function of time and in a histogram. Note that the values are divided in groups, which should make it possible to model the distribution without using a large number of simple distributions. In the figure, only the red intensity is shown, but the other colour channels are similar with a high degree of correlation.



(a) Red intensity as a function of time.

(b) Histogram of red intensity values.

Figure 4.2: *Red values of a pixel in a duration of the development data set where dark persons occasionally pass a bright background. Most of the values are located in two groups in the histogram; one for the bright background and one for the dark passing foreground.*

4.1.2 Parametric Distributions

The most frequently used mixture model in the literature is the GMM [15]. This method was originally introduced for image processing by Stauffer et. al. [71]. In 2008 over 150 different papers were published, investigating different kinds of this method [15]. Some researchers have, however, pointed out that there typically exists very little noise in indoor environments when using quality cameras, and that the background is often better modelled as a Laplacian distribution rather than a Gaussian distribution [48].

To determine which parametric distribution is suitable, tests have been carried out to determine if the background fit into a simple, predefined distribution. In Figure 4.3 the distribution of one pixel from the development data set that is background all the time is shown. The graphs indicate that a Gaussian distribution models the pixel values quite well. This example is consistent with the general situation, and Gaussian distributions are therefore chosen as the basis distribution in the mixture model.

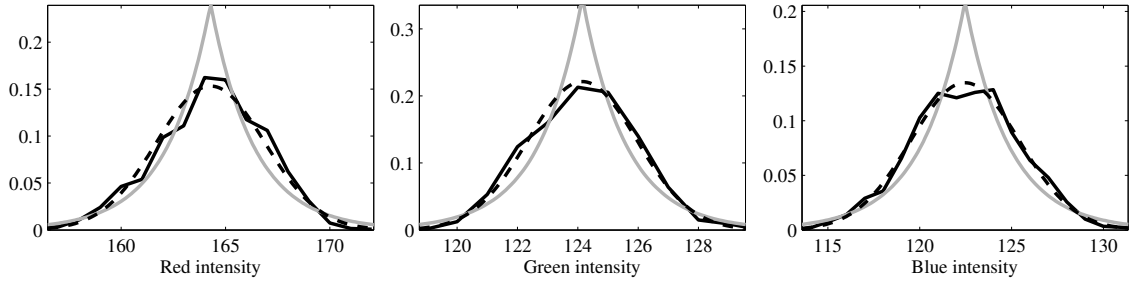


Figure 4.3: *The black line is the distribution of the pixels value for one pixel (in the development data set described in Section 1.1). The dashed and the grey line are the best fitting Gaussian and Laplacian distribution, respectively.*

4.1.3 Multivariate Gaussian Distributions

The probability density function (pdf) of a multivariate Gaussian distribution is given as:

$$f_{\text{Gaussian}}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{4.1}$$

where:
$\mathbf{x}$ is a random variable vector,
$\boldsymbol{\mu}$ is the mean value vector, and
$\boldsymbol{\Sigma}$ is the covariance matrix.

Thus, for each Gaussian distribution in the GMM, a 3 dimensional mean and a 3×3 covariance matrix must be stored. Additionally, a weight also needs to be stored for each Gaussian distribution. These means that a total of 13 variables must be stored per Gaussian. As each pixel is usually modelled using 3-5 Gaussians, a total of 39 to 65 variables need to be stored per pixel [71]. To save both memory and computation time, many papers using GMM do not use a full covariance matrix. Stauffer et. al. e.g. only uses one standard deviation, averaged over all the colour channels [71]. This corresponds to adding variations in all colour channels and modelling using one univariate GMM. This approach both minimizes computation time, memory consumption, and reduces the risk that the variance of a distribution collapse to almost zero if no (quantized) noise exist for some period of time. Therefore we also apply this simplification:

$$\boldsymbol{\Sigma} = \sigma^2 I \tag{4.2}$$

4.2 The GMM Background Estimation Algorithm

A flowchart of the necessary steps in the GMM algorithm is shown in Figure 4.4. In every new frame the colour of each pixel is compared against the distributions of the pixel to test if this colour belongs to one of the existing distributions. The distributions are updated and sorted according to the likelihood of being background. The pixel is afterwards marked as foreground or background depending on the matching distribution. In the following the algorithm is explained in depth.

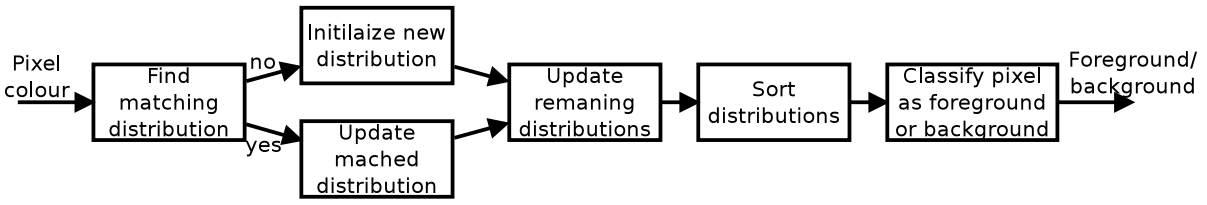


Figure 4.4: *Flowchart of the background estimation algorithm based on GMM. The input of the algorithm is a RGB colour value for each frame, and the output is a classification of the pixel as either foreground or background. Thus, the first two boxes from Figure 4.1 are covered, while noise and shadow removal must be handled afterwards.*

4.2.1 Finding Matching Distribution

A pixel value $\mathbf{x}_k$ matches a distribution if: (In the following, the subscript n for the distribution number has been left out for simplicity unless required)

$$\frac{(\mathbf{x}_k - \boldsymbol{\mu}_{k-1})^{\mathrm{T}}(\mathbf{x}_k - \boldsymbol{\mu}_{k-1})}{\sigma_k^2} < \tau_\sigma^2 \tag{4.3}$$

where:
k is the frame number, and
τ_σ is a predefined number of standard deviations that the pixel value must be within.

Stauffer et. al. uses a threshold of $\tau_\sigma = 2.5.$, which corresponds to misclassifying 1.2% of the background pixels as foreground. This can easily be removed as noise, and the same threshold is therefore chosen here.

4.2.2 Updating the Distributions

The update equation for the weight of a distribution is given as:

$$\omega_k = (1 - \alpha)\omega_{k-1} + \alpha \cdot M_k \tag{4.4}$$

where:
M_k is 1 if the pixel value matches the distribution and 0 otherwise,
α is the weight learning factor, and
the weights are normalised using $\omega_{n,k} = \frac{\omega_{n,k}}{\sum_{c=1}^{K} \omega_{c,k}}$, where n is the distribution number.

The mean and the variance of the matched distribution are updated according to:

$$\sigma_k^2 = (1 - \rho)\sigma_{k-1}^2 + \rho(\mathbf{x}_k - \boldsymbol{\mu}_{k-1})^T(\mathbf{x}_k - \boldsymbol{\mu}_{k-1}) \tag{4.5}$$

$$\boldsymbol{\mu}_k = (1 - \rho)\boldsymbol{\mu}_{k-1} + \rho\mathbf{x}_k \tag{4.6}$$

where $\rho = \frac{\alpha}{\omega_{k-1}}$ is the learning factor for the distributions.

The learning factor ρ for σ^2 and $\boldsymbol{\mu}$ are scaled by the weight of the distribution. This technique was introduced in [41], and it enables newly discovered distributions which are not known very

precisely to be learned faster. Note that the mean and variance of the unmatched distributions are of course not updated.

The predefined learning factor, α, is treated in several different ways in literature. While Stauffer et. al. uses a constant factor [71], Bowden et. al. has improved the performance especially during the first minutes of video capturing by introducing an L-recent window [40, 41, 42]. This technique is also used here, by setting $\alpha = \max(\alpha_{\min}, 1/k)$, where $\alpha_{\min}$ is a predefined constant. For our experiments $\alpha_{\min}$ is adjusted to give a window length of 26 seconds which corresponds to a learning rate of 0.00256 at 15 fps.

4.2.3 Sorting Distributions and Classification of Pixels

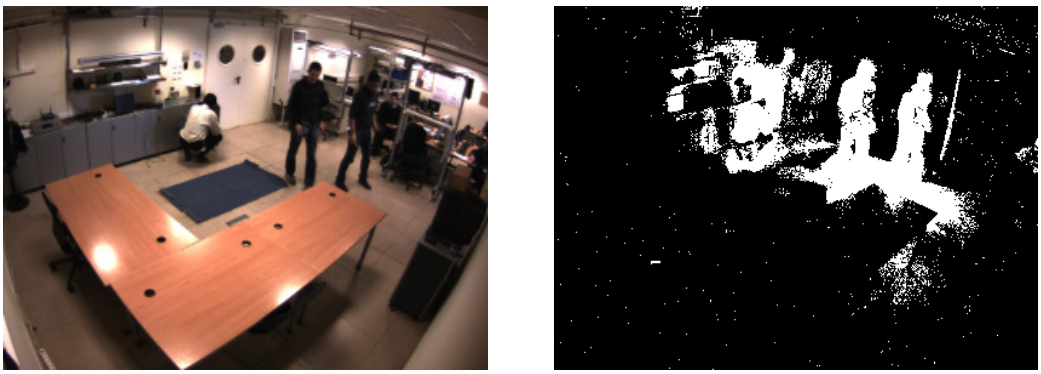
When all distributions have been updated, the pixel must be classified as belonging to either the background and or the foreground. This is done by arranging the distributions according to a fitness factor f, indicating the relative likelihood that each distribution belongs to the background. Background distributions are expected to be present most of the time, and at the same time moving persons tend to yield a larger variance than stationary background. Therefore, the fitness factor is calculated as $f_k = \frac{\omega_k}{\sigma_k}$. After sorting according to f_k ,the first B distributions are chosen as background according to:

$$B = \text{argmin}_b \left(\sum_{c=1}^{b} \omega_{c,k} > T_{\text{background}} \right) \tag{4.7}$$

where $T_{\text{background}}$ is a predefined portion of the time, the background must at least be present.

More elaborate approaches than this hard limit are definitely possible. However, this method was used by Stauffer et. al. [71], and it provides good results using the development data set. For our experiments $T_{\text{background}} = 0.6$ has been used.

An example of the performance of our GMM implementation is shown in Figure 4.5. As can be seen from the figure, the persons are indicated as foreground relatively well by the GMM foreground detector. However, the marked foreground includes beside the persons also shadows and noise. This will be handled in Section 4.4 and 4.5. First the time consumption is analysed, as we would like the program to run in real-time cf. Chapter 3.



(a) Original image. **(b)** Detected foreground.

Figure 4.5: *Example of the foreground estimation. Note that noise and shadows have not been removed.*

4.3 Time Consumption of the GMM Implementation

A test on development data set has shown that the GMM algorithm takes around 1.9 second to run on all 5 images with the resolution 1600×1200 for the four corner cameras and 1024×768 for the top camera, cf. Table D.1 in Appendix D. These high image resolutions might not be required for the foreground detection algorithm. Therefore tests have been carried out to investigate if whether an on-the-fly resolution reduction of the images can make the GMM algorithm run significantly faster, without reducing the quality significantly.

The time consumption of the per-pixel GMM algorithm should in theory depend linearly of the number of pixels that is used. Thus, it is expected that the algorithm will run approximately twice as fast if the picture is of half resolution. A graph of the theoretical and measured time consumption is shown in Figure 4.6. Note, that the measured time consumption follows this theoretical except for an offset caused by downscaling of the image resolution. The conclusion is that this approach will definitely reduce the time consumption.

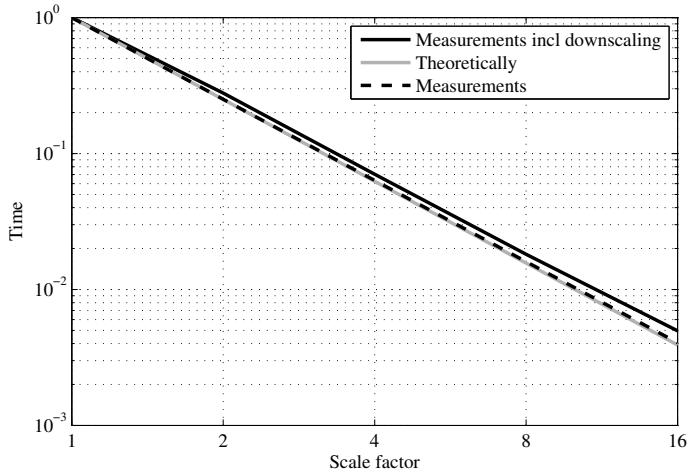
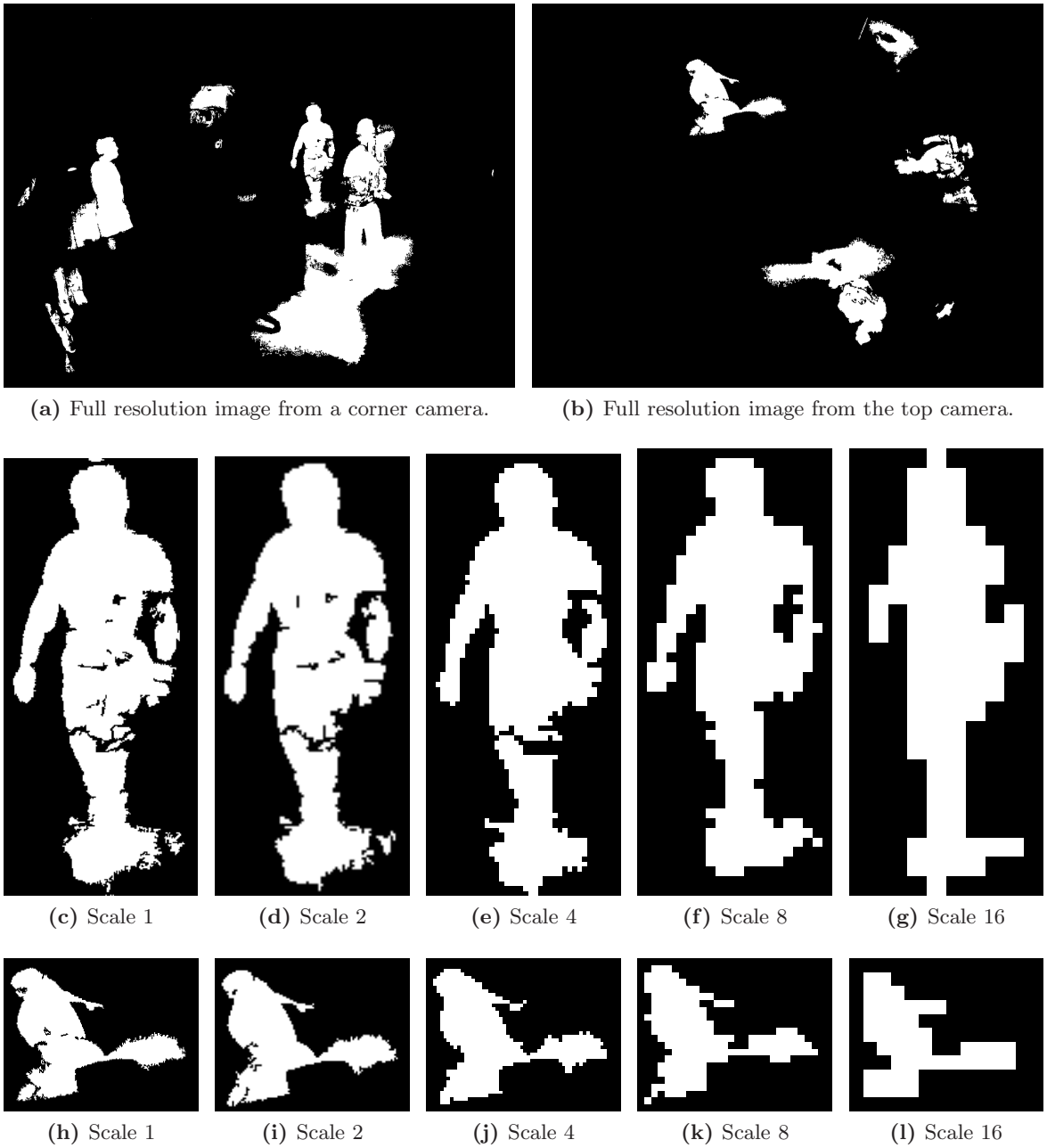


Figure 4.6: *Time measurements of the GMM algorithm normalised to the time of the algorithm running on the full resolution images. The theoretical and measured reduction in time consumption are almost identical. The solid black line shows the time consumption including the time used for downscaling of the images.*

4.3.1 Quality using Images of Reduced Resolution

A scaling of the images must not reduce the quality of the detected foreground to a point, where it is significantly more difficult to recognise the persons from the detections. Therefore, a number of tests have been carried out where the foreground detector was using varying resolution scaling. Some results for camera 0 and 4 (the top camera) are shown in Figure 4.7. A scaling factor at 16 clearly makes it difficult to precisely locate the persons. A scaling factor up to 4 does, however, not seem to affect the quality too much. As this reduces the number of pixels handled by the foreground detector by a factor of $4^2 = 16$ the time consumption is similarly reduced according to the measurements in Figure 4.6. Therefore a scaling factor of 4 is used for the development data set. The recordings from the CLEAR data set that is to be used for testing the system contains different resolutions, ranging from 640×480 800×600. Therefore the scaling factor is made dependant of the frame resolutions, and the minimum width is set to 320 pixels.



(a) Full resolution image from a corner camera. **(b)** Full resolution image from the top camera.

(c) Scale 1 **(d)** Scale 2 **(e)** Scale 4 **(f)** Scale 8 **(g)** Scale 16

(h) Scale 1 **(i)** Scale 2 **(j)** Scale 4 **(k)** Scale 8 **(l)** Scale 16

Figure 4.7: *Quality comparison of foreground detection using varying resolutions of images from frame 61 of the short development data set. The images in Figure (c) through (g) are identical (scaled) parts of the detected foreground from a corner camera. The images in Figure (h) through (l) are similarly identical (scaled) parts of the detected foreground from the top camera. The detected foreground in the complete full resolution image are shown for each of the two cameras in Figure (a) and (b) respectively. Note that noise but not shadows has been removed from both images.*

4.4 Shadow Removal

In this section the shadow removal algorithm is designed. As illustrated in Figure 4.1, the shadow removal algorithm is supposed to be followed by noise removal. Thus, only the majority of the shadows need to be removed in this step if the remaining shadow is distributed as noise. In order to achieve the goal to make a real-time tracker it is chosen to do shadow removal per-pixel instead of analysing larger areas of the foreground masks at once.

One way to handle shadows is to use a colour space with separate channels for brightness and colour information. A conversion of the entire frame as well as the background model to another colour space will, however, be very time consuming. Instead the conversion can be limited to the pixels marked as foreground. In [36], a fast implementation of this is described. Two notations are needed, namely the "brightness distortion" D_b and "colour distortion" D_c. D_b is a scalar that indicates the brightness of a particular pixel compared to the background pixel. D_c is a scalar that indicates the colour difference between a particular pixel and the background pixel. In Figure 4.8 both are illustrated in the RGB space where a background pixel is showed together with a foreground pixel. D_b is defined as the length of the projection of the foreground pixel colour $\mathbf{c}_\mathrm{fg}$ on the background pixel colour $\mathbf{c}_\mathrm{bg}$:

$$D_\mathrm{b} = \arg\min_{\alpha} \left((\mathbf{c}_\mathrm{fg} - \alpha \cdot \mathbf{c}_\mathrm{bg})^2 \right) \tag{4.8}$$

This can be solved by taking the derivative and setting to zero:

$$\begin{aligned} D_\mathrm{b} &= \arg\min_{\alpha} \left(\mathbf{c}_\mathrm{fg}^2 + \alpha^2 \cdot \mathbf{c}_\mathrm{bg}^2 - 2\alpha \cdot \mathbf{c}_\mathrm{fg}^\mathsf{T} \mathbf{c}_\mathrm{bg} \right) \\ &= \left\{ \alpha \;\middle|\; 0 = 2\alpha \cdot \mathbf{c}_\mathrm{bg}^2 - 2 \cdot \mathbf{c}_\mathrm{fg}^\mathsf{T} \mathbf{c}_\mathrm{bg} \right\} \\ &= \frac{\mathbf{c}_\mathrm{fg}^\mathsf{T} \mathbf{c}_\mathrm{bg}}{\mathbf{c}_\mathrm{bg}^2} \end{aligned} \tag{4.9}$$

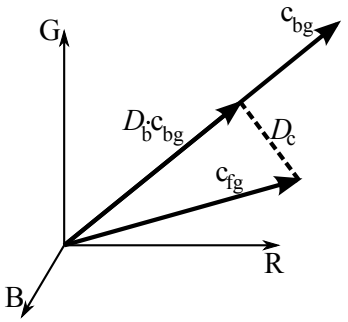


Figure 4.8: *The D_c and D_b values for a set of a foreground and a background pixel in RGB space [36].*

D_c is defined as the orthogonal distance from $\mathbf{c}_\mathrm{bg}$ to the line spanned by $\mathbf{c}_\mathrm{fg}$:

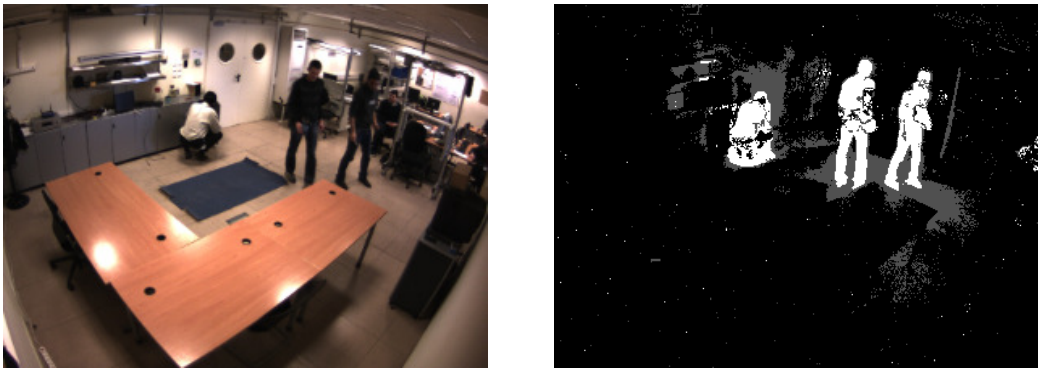
$$D_\mathrm{c} = \| \mathbf{c}_\mathrm{fg} - D_\mathrm{b} \cdot \mathbf{c}_\mathrm{bg} \| \tag{4.10}$$

With these definitions, shadow pixels can be found by comparing a foreground pixel with the corresponding background GMM. To increase the speed, only the most likely distribution in the

GMM is used. If D_b is below 1 and above a threshold τ_{D_b} and D_c is below another threshold τ_{D_c}, the pixel is accepted as shadow. The threshold τ_{D_b} is introduced to prevent too many very dark pixels from getting marked as shadow. The algorithm is summed up as pseudo-code in Figure 4.9. In Figure 4.10, the performance of the shadow detection is illustrated.

- **For** each foreground pixel
  - Calculate D_b between the most likely background model and the pixel.
  - **If** $1 > D_b > \tau_{D_b}$ **then**
    - **If** $D_c < \tau_{D_c}$ **then**
      - Shadow found.

Figure 4.9: *Pseudo-code for detection shadow.*



(a) Original image. **(b)** Detected shadow (in grey).

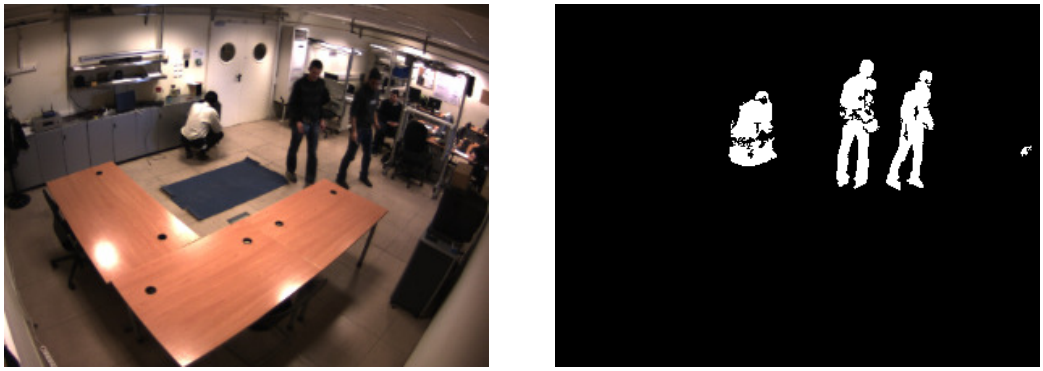
Figure 4.10: *Example of the foreground estimation with detected shadows marked in grey.*

4.5 Noise removal

After shadow removal the only task left in 2D foreground detection is noise removal (as illustrated in Figure 4.1). The foreground detection is based on a stochastic model, where a certain amount of the true background distribution can be expected to fall outside the threshold on the standard deviation, as described in Section 4.2.1. This causes noise to be distributed randomly pixel wise. Several approaches exist in literature to remove this noise, including neighbourhood averaging [29], BLOB analysis [71], and a pixel persistence map (PPM) combined with binary decision trees [51]. The PPM approach works by creating a map of the same size as the foreground mask and store the weight of the fitted Gaussian for each pixel in this map. Thus, lower values indicate pixels that are more likely to be foreground. A binary decision tree that merges the most similar connected regions using a bottom-up approach are then applied, and an appropriate similarity measure and stop criterion will enable this approach to sort out most noise. One-way BLOB analysis, which Stauffer et al. uses [71], is the simplest and fastest of these approaches, and it gives good results in many situations. One limitation is, that falsely detected background due to high similarity between the background and the current foreground is not removed (thus "one-way").

For our system, two-way BLOB analysis is used, which removes small areas of both foreground and background. If moving objects are similar to the background, this can give better results

than one-way BLOB analysis without introduction much extra computation time. BLOBs with an area less than a predefined threshold, τ_{size}, are removed. For our experiments, the minimum area has been set to $\tau_{\text{size}} = 50$, and the filtering result is illustrated in Figure 4.11.



(a) Original image. **(b)** Noise removed.

Figure 4.11: *Example of the final foreground mask after shadows and noise have been removed.*

4.6 Target Protection

In the previous sections, a purely pixel based foreground detector followed by noise removal has been described. This is in many situation a very good general purpose foreground detector. It has, however, a weakness if the foreground becomes immobile, as it will be incorporated into the background after some time. To delay this effect, information from the tracking system can be used to protect the tracked persons, also known as targets, from being incorporated into the background as fast. The background update cannot be stopped completely, however, since this can cause the background to never recover after an object has been moved. Therefore the protection is handled by decreasing the learning rate of a rectangle around the target by 40%, which for the development data set has proven to increase the performance significantly. This means that the final foreground detector besides the input from the camera also gets input from the tracker as feedback as illustrated in Figure 4.12.

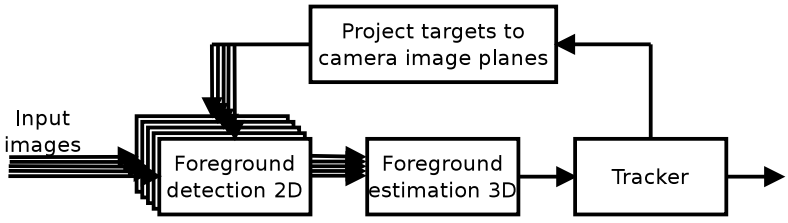


Figure 4.12: *Feedback from the tracker is used to decrease the learning rate of the background estimation algorithm at locations where the persons are tracked.*

The parameter values used as default in our implementation of the 2D foreground detection are shown in Table 4.1.

Parameter	Symbol	Value
GMM models:		
Gaussian distributions per GMM	-	4
Weight of new Gaussian in a GMM	ω_{init}	0.01
Variance of new Gaussian in a GMM	σ^2_{init}	30
Part of the time the background distributions must be present	$T_{\text{background}}$	0.6
Distance for a pixel colour to be accepted as belonging to a Gaussian distribution	τ_σ	2.5 σ
Minimum weight learning rate	α_{min}	0.0025 (at 15 fps)
Relative learning rate for protected targets	-	40%
Shadow and noise removal:		
Shadow brightness distortion threshold	$\tau_{D_{\text{b}}}$	0.65
Shadow colour distortion threshold	$\tau_{D_{\text{c}}}$	45
Minimum BLOB size	τ_{size}	50 pixels
Optimisations:		
Minimum with of images for foreground detection	-	320 pixels

Table 4.1: *Parameters used by the 2D foreground detector.*

Chapter 5

Foreground Estimation in 3D

In this chapter it is discussed and explained how several 2D foreground masks can be combined into a single 3D representation of foreground, as illustrated in Figure 5.1. The purpose of using several cameras for tracking is both to filter out noise, but also to be able to track on the floor plan in dimensions that are independent of the positioning of the cameras. Noise in the 2D foreground exists no matter the method used. By combining information from a number of cameras, the amount of noise can be reduced significantly and thus increasing the robustness of the tracking algorithms. The chapter is organized as follows: In Section 5.1 it is analysed how the 3D space can best be modelled, and in Section 5.2 it is discussed how to best convert the 2D foreground masks into 3D foreground. In Section 5.3 a hierarchical grid structure is presented and analysed, which can make the conversion from 2D to 3D more efficient. The various used parameters are listed in Table 5.1 at the end of the chapter.

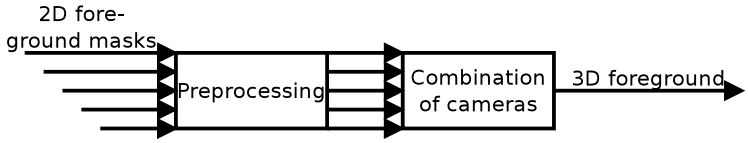


Figure 5.1: *Steps in 3D foreground detection.*

5.1 Modelling the Space in 3D

The 2D image plane of each camera is spanned discretely per pixel. The 3D space can either also be spanned discretely or as a continuous space:

Discrete: Modelling 3D spaces discretely by spanning with 3 dimensional cubes known as *voxels* is a well known method [50, 44, 46, 49]. This approach is very much similar to spanning a 2D plane with pixels. The word "voxel" is a combination of "volume" and "pixel", and a voxel can thus to be considered as a 3D pixel. Using this representation can make it possible to combine information from the different cameras per voxel instead of per person or region.

Continuous: Continuous representations are also common in literature [16, 33]. Using such representations, persons can be found in 3D e.g. by first estimating the position of the

persons in 3D by first identifying large BLOBs of foreground in each camera and then use epipolar geometry to estimate their positions in 3D. This has been done e.g. in [16], where the top of 2D BLOBs in each camera is estimated and afterwards combined into 3D positions. Although this approach can work very well in some situations, it is based on position estimates per camera, and will therefore not fully utilise the noise removal capability of having multiple cameras.

Due to the simplicity and large potential for noise removal, the discrete voxel-based representation of the 3D space is chosen. As described in the introduction of the report, Chapter 3, the precision of the tracker is not required to be larger than 5-8 cm. Also, the width of human body parts such as arms or heads are typically at least 5 cm. Therefore, this resolution is also chosen for the discrete representation of the room. The following section discusses how the 2D foreground masks can best be combined into discrete 3D foreground.

5.2 Combination of 2D Foreground Masks into 3D Foreground

The basic principle in determining if a particular voxel contains foreground is to project it to all of the camera image planes and check the corresponding foreground masks. How to do the projection is described in Appendix A.3.1. If enough cameras detect significant foreground, the voxel can be considered foreground. An important design decision is to determine what *enough* and *significant* means in this context.

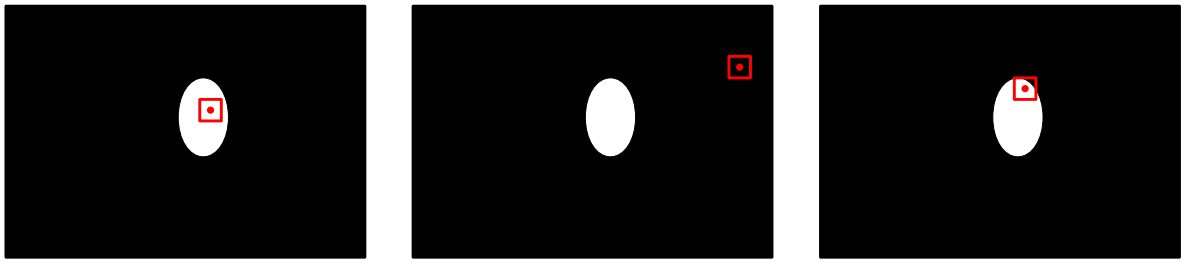
The most exact way to determine whether a voxel projected to the image plane of a camera contains significant foreground is to consider all foreground mask pixels located within that projected voxel. The percentage of pixels with foreground can then either be compared with a threshold for significant foreground, or used as a non-boolean indication of foreground. This is, however, computationally expensive, since pixels in the 2D foreground masks are included in many voxels, and will thus be tested many times.

The speed of the foreground testing can be increased by making certain simplifications. Three possibilities are analysed here:

Solution based on centre pixels: The foreground masks will typically contain a certain number of coherent foreground areas. Thus, in many cases, a projected voxel will be located either completely inside or completely outside a foreground area. In these cases, the centre pixel of a projected voxel indicates correctly if the voxel contains foreground, as illustrated in Figure 5.2. Even when a projected voxel is located on the border of a foreground area, the centre pixel will be a good indication of whether the projected voxel *mostly* contains foreground as shown in Figure 5.2c.

Testing only centre pixels will save much computation time and give good results if no noise is present. It will, however, not be very resistant to noise, since only one pixel needs to be affected by noise for the algorithm to yield a wrong conclusion.

Solution based on blurring filter: Before testing the centre pixel of each projected voxel, a blurring filter can be applied to the foreground masks. This approach has the strength that it saves computation time because only one pixel are tested per voxel, while at the

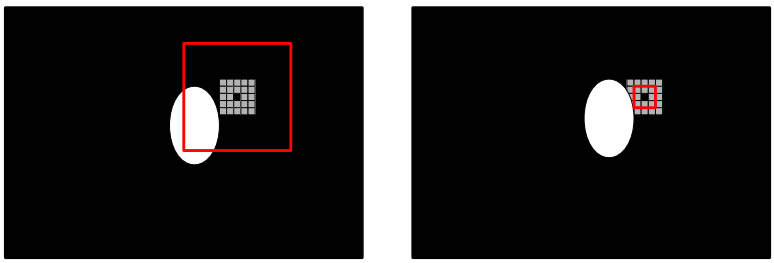


(a) Voxel inside foreground. **(b)** Voxel outside foreground. **(c)** Voxel on border of foreground.

Figure 5.2: *Detected foreground is white and a projected voxel is red and has its centre marked. Projected voxels are likely to be located either completely inside (a) or completely outside (b) a coherent foreground area. If no noise is present, the centre pixel indicates correctly in these situations whether significant foreground is present in that projected voxel. Even when a projected voxel is located in the border of a foreground area (c), the centre pixel often indicates correctly whether most of the projected voxel is filled with foreground.*

same time retaining some resistance to noise. Additionally, a non-boolean indication of foreground can be attained, allowing for a smart combination of information from the different cameras.

A problem can be that the optimal kernel size for the blurring filter is very different for the different projected voxels because the sizes of these are very different. This is caused partly by different distances to the camera and partly by the voxels being of different sizes in 3D (see Section 5.3). Figure 5.3 illustrates that it is impossible to select a kernel size that makes it possible to determine correctly if two voxels of different sizes hold foreground. A compromise can be to simply use a number of different sized kernel. However, many and too large kernels will require much computation time.



(a) Large voxel. **(b)** Small voxel.

Figure 5.3: *Detected foreground is white, a projected voxel is red, and a kernel of 5x5 pixels around the voxel centre is marked in grey. In (a), foreground is falsely not detected, whereas foreground (might be) falsely detected in (b). A larger filter can minimize the problem for large voxels as in (a) but will simultaneously increase the problem for smaller voxels, and vice versa.*

Solution based on distance transform: A way to maintain the property of only having to consider the centre pixel of each voxel but at the same time making the algorithm more robust to different voxel sizes, is to apply a distance transform instead of a blurring filter to the foreground masks [12]. In this way, the distance to the nearest foreground can be found by testing only the value of the center pixel. A comparison to a "radius" of the projected voxel can indicate whether it contains foreground.

A problem with this approach is, that the projected voxels are not circular, and a true radius can therefore not be found. Instead, the radius of an enclosing circle C can be used.

As illustrated in Figure 5.4, this inevitably introduces the possibility to falsely detect foreground.

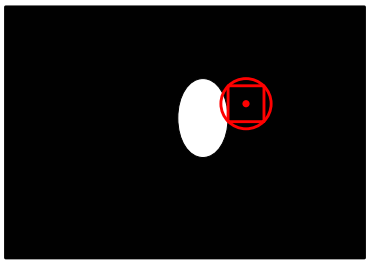


Figure 5.4: *A distance transform of the foreground masks will make it possible to determine the distance to the nearest foreground. The can be compared to the radius of an enclosing circle C of a projected voxel. As illustrated, false positive foreground detections may occur.*

The advantages of the solution based on a blurring kernel is that it natively suppresses noise and that it gives non-boolean foreground indications. However, it is not necessarily advantageous to remove noise per camera in 2D, since noise is also suppressed when redundant foreground information from multiple cameras are combined into 3D foreground. And if necessary, noise suppression can be added to the other approaches by applying blob analysis. Therefore, and to make the system able to handle all sizes of projected voxels in a generic way, the distance transform based solution is chosen. This solution also makes it possible to detect foreground using hierarchies of voxels, as is described in Section 5.3.

5.2.1 Implementation of Distance Transform

An exact distance transform is very time consuming. Therefore, we have implemented an efficient approximating distance transform algorithm following the approach of [12]. It works by applying a 3×3 kernel to the image in two steps. First, the upper/left half of the kernel is run from upper left corner from left to right and afterwards runs the lower/right half of the kernel from button right. This causes the calculated distances to be slightly imprecise, but also enables the time consumption to be comparable to a 3×3 blur kernel. The kernel used is illustrated in Figure 5.5.

+b	+a	+b
+a	0	+a
+b	+a	+b

Figure 5.5: *Kernel used for distance transform. The variables a and b are set to the optimal values 0.95509 and 1.36930 [12].*

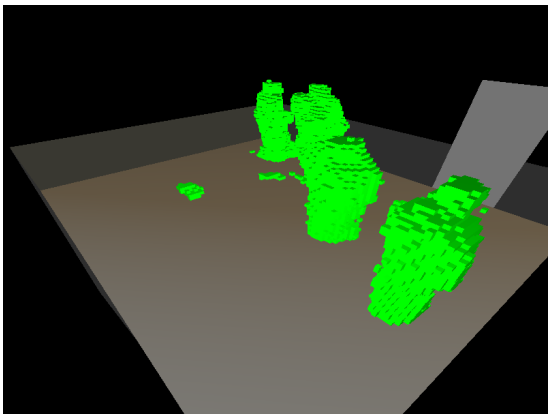
In our implementation, the optimal values 0.95509 and 1.36930 are used for the horizontal/vertical and diagonal entries in the kernel, respectively. This causes the error in the distances to be at most 4.49% [12].

With the choice of the distance transform to combine information from the different cameras, 3D foreground can be estimated. An example of the performance of the algorithm is shown in Figure 5.6. The default settings used in our implementation marks a voxel as containing foreground only if it can be seen from at least 4 of the 5 cameras and all cameras that can

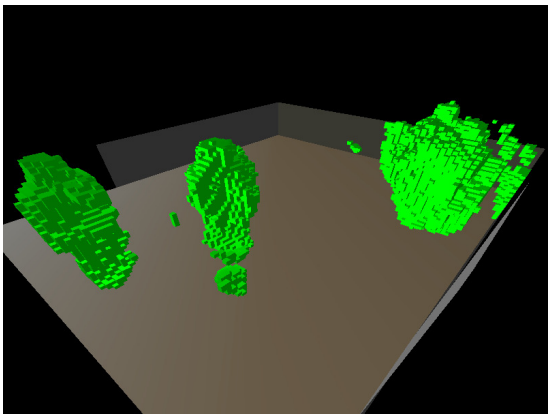
see it detect foreground. The remaining sections of this chapter concern time consumption and optimisations of the algorithm.

(a) Frame from camera 3

(b) Frame from camera 2



(c) 3D foreground from a view point close to camera 3



(d) 3D foreground from a view point close to camera 2

Figure 5.6: *Output from the 3D foreground estimator at frame 2916 in the development data set.*

5.3 Hierarchical Grid Structure

Foreground in the 3D space will mostly be structured in coherent volumes that indicate the presence of persons, as e.g. illustrated in Figure 5.6. Large areas of the space will be completely without foreground. By dividing the space into hierarchies of voxels, these areas can be ruled out efficiently by only testing very large voxels for foreground. Only if a large voxel contains foreground is it necessary to test smaller voxels it contains (its children).

One efficient way to construct hierarchies is to use octrees; that is to divide every voxel on a particular hierarchical level into 8 voxels on a lower level [27]. However, that is not necessarily the most optimal way to partition the voxels. To be able to gain most advantage, our implementation supports an arbitrary number of levels, where a voxel on each level can consist of an arbitrary number of child voxels (e.g. $2^3 = 8$, $3^3 = 27$, or $4^3 = 64$). The algorithm for converting the 2D foreground masks into a grid of foreground voxels is summed up as pseudo-code in Figure 5.7. In Section 5.4, it is investigated which number of hierarchy levels that is optimal with respect to time consumption.

- Span the room with a grid of voxels on N hierarchical levels.
- Project the centre and corners of each 3D voxel on all levels to the image plane of each camera. Use the corners to determine the radius of the enclosing circle C.
- Remove voxels that cannot be seen by a sufficient number of cameras.
- **FOR** each camera:
  - Perform distance transform of the foreground mask.
- Let the set S consist of all voxels on the highest hierarchical level.
- **FOR** each voxel in S:
  1. **FOR** each camera:
     - **IF** the value of the centre of the projected voxel in the distance map is below the radius of C, **THEN** foreground is detected.
  2. **IF** all cameras that can see the voxel detect significant foreground:
     - **IF** the voxel has any children, **THEN** repeat 5.3 with S consisting of all children of the voxel. **ELSE** mark the voxel as a foreground voxel.

Figure 5.7: *Recursive algorithm for converting the 2D foreground masks to a 3D grid of foreground voxels. The number of cameras that must be able to see a voxel for it to be included in the grid is set to 4 of the 5 cameras as default in our system.*

For our system, all of the cameras are stationary. This causes the projection of voxels to the image plane of each camera to be identical for all frames. Therefore, the first tree items in Figure 5.7 can be carried out off-line, leaving 3D foreground testing as the only potentially computationally heavy part.

5.3.1 Comparison of Hierarchical and Non-Hierarchical Algorithms

The purpose of introducing a hierarchical structure is to decrease computation time. It must, however, be avoided that the hierarchical structure in any way decreases the quality of the algorithm. In order to get the same result of the algorithm no matter the number of levels the top level voxel sizes, a few special situations must be taken into account:

1. **Room dimensions:** Voxels at the highest level (of the largest size) may not fill out the room/camera view as good as the voxels on the lowest level.
2. **Camera view:** Some voxels may be visible from a particular camera, even though their parent voxel is not.
3. **Radii:** It is possible for the enclosing circle of a projected voxel to include an area *not* included in the enclosing circle of its projected parent voxel.

Each of these cases are analysed in the following.

Room Dimensions and Camera View

There exist two simple ways to avoid the problem with room dimensions:

- The room can be spanned completely with voxels on the highest level, even though some voxel centres must be placed outside the room. Of course the centres that are outside the room cannot be tested for foreground, so instead these voxels must always be marked as containing foreground.
- If the centre of a voxel is about to be placed outside the room, it can be discarded and replaced by its child voxels. In this way, the room will be filled as best as possible with voxels as large as possible, while avoiding centres outside the room.

The outcomes of the two approaches are identical, and since they are equally simple, the second more efficient approach is chosen. It is illustrated in Figure 5.8 with 3 hierarchical levels.

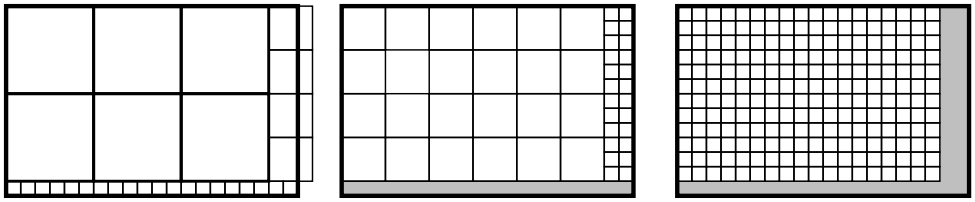
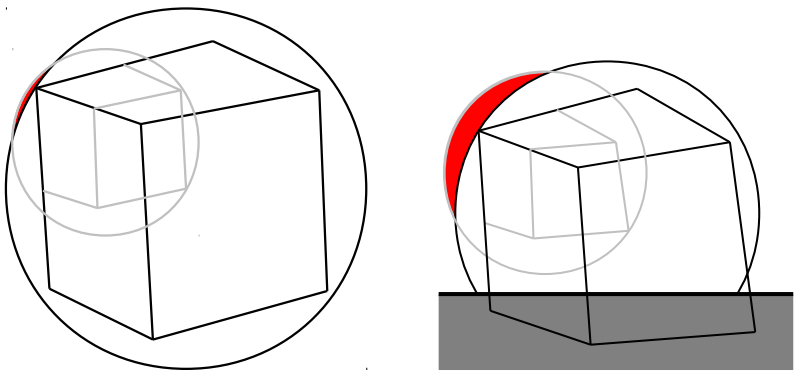


Figure 5.8: *Example of a distribution of voxels when using 3 hierarchical levels. The top level is shown to the left and the bottom level is shown to the right. The grey areas are parts of the room that is fully covered by voxels of the smallest size on a higher level and is therefore not considered on this level.*

The problem with camera views is similar to the problem with room dimensions. A particular camera might be able to see the centre of a particular voxel even though the centre of the parent voxel is not visible. This is handled the same way as the problem with room dimensions: If a particular voxel is seen by more cameras than its parent voxel, the parent is replaced by all of its children.

Radii

In some cases, perspective and camera distortion can cause the enclosing circle of a child voxel (C_{child}) to contain an area not included in the enclosing circle of its parent voxel (C_{parent}). If foreground is present in this area, but *not* in the rest of C_{parent}, this will cause the hierarchical structure to sort out the child voxel, even though foreground exists within its enclosing circle. Figure 5.9 illustrates how such a situation can occur. Minor tests have indicated that around 0.1% of the foreground voxels are sorted out for this reason. The issue could easily be avoided by using a circle slightly larger than C_{parent} for parent voxels. However, since this only happens when there is foreground inside the enclosing circle of a child voxel but *not* inside the voxel itself, there is no actual reason to prevent it.



(a) Case 1: Both voxels completely present.

(b) Case 2: Some of the higher level voxel is outside the room and/or outside the view of a particular camera.

Figure 5.9: *Two cases where the enclosing circle of a voxel on a lower level covers an area that is not covered on an upper level (marked in red).*

5.4 Optimal Hierarchy Levels

This section analyses the reduction in time consumption that is possible by using the hierarchical approach to 3D foreground detection. The purpose of this is to find the optimal number of hierarchical octree levels. This is of course highly dependant of the data used, including the resolution of the cameras and the number of people in the scene. For the test data presented here, the short development data set described in Section 1.1 with between 0 and 3 people is used.

In Section 5.4.1 the possible time reduction is analysed from a theoretical point of view. In Section 5.4.2 this is compared to the actual time reduction achieved by our implementation.

5.4.1 Theoretical Time Reduction

As described in Section 5.1, the voxel size is chosen to 5 cm (on the lowest hierarchical level). This causes the total number of voxels in the area under surveillance to be above 400,000. The amount of these that contain foreground is of course directly dependant of the amount of foreground present in the cameras, which is dependant both of the number of people present in the scene and of the amount of noise. Figure 5.10 shows the average percentage of voxels that contain foreground in the short development data set as a function of the voxel size, after noise and shadow removal. Note that only a few percentages of the lowest level voxels contain foreground. This indicates that the potential for time reduction by using the hierarchical approach is large for this data set.

From these data, a theoretical time reduction by applying hierarchies can be calculated. If no hierarchy is used, the total time consumption in converting the distance maps to 3D foreground is:

$$t_{\text{tot}} = t_{\text{voxel}} \cdot N_1 \tag{5.1}$$

where:

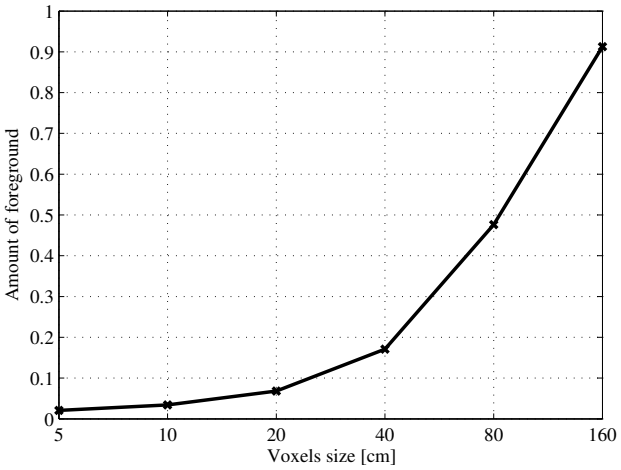


Figure 5.10: *The amount of voxels with foreground as a function of voxel size (height and width). Since larger voxels contain a larger volume of the space, they are more likely to contain foreground.*

t_{voxel} is the time required to determine if one voxel has foreground, and
N_1 is the number of voxels on level 1 (the lowest level).

When k hierarchical octree levels are used, the total time consumption is:

$$\begin{aligned} t_{\text{tot}} &= t_{\text{voxel}} \left(N_{\text{k}} + N_{\text{k-1}} a_{\text{k}} + N_{\text{k-2}} a_{\text{k-1}} + ... + N_1 a_2 \right) \\ &= t_{\text{voxel}} \cdot N_1 \left(\frac{1}{8^{k-1}} + \sum_{n=2}^{k} \left(\frac{1}{8^{n-2}} a_{\text{n}} \right) \right) \end{aligned} \tag{5.2}$$

where:
$N_{\text{x}} = \frac{1}{8^{x-1}} \cdot N_1$ is the number of voxels on level x, and
a_{x} is the amount of voxels on level x that has foreground.

By examining Equation (5.2), the theoretical change in time consumption by adding another level $\text{k} + 1$ can be determined to be:

$$\begin{aligned} t_{\text{tot,k,k+1}} &= N_{\text{k+1}} + N_{\text{k}} a_{\text{k+1}} - N_{\text{k}} \\ &= \frac{1}{8} N_{\text{k}} + N_{\text{k}} a_{\text{k+1}} - N_{\text{k}} \\ &= N_{\text{k}} \left(a_{\text{k+1}} - \frac{7}{8} \right) \end{aligned} \tag{5.3}$$

This means that time can be saved by adding of additional level if and only if the amount of voxels with foreground on that new level is below $\frac{7}{8}$. As shown in Figure 5.10, this is the case for the first 5 levels (up to 80 cm) for the short development data set. The theoretical time consumption for different number of layers compared is compared to the actual measured time consumption in Figure 5.11.

5.4.2 Actual Time Reduction

The total time consumption using between 1 and 6 hierarchical levels is shown in Figure 5.11. The shape of the (theoretical) predicted time reduction curve and the measured time reduction

is similar, but the time is not reduced as much as expected. While around 95% time reduction is expected at 4 hierarchical levels, only around 88% is measured. The reason for this is partly overhead in the implementation and partly the fact, that higher level voxels does not "fit" the room perfectly, as described in Section 5.3.1. This causes some voxels to remain unaffected by the addition of higher levels, and this effect has not been included in the theoretical model.

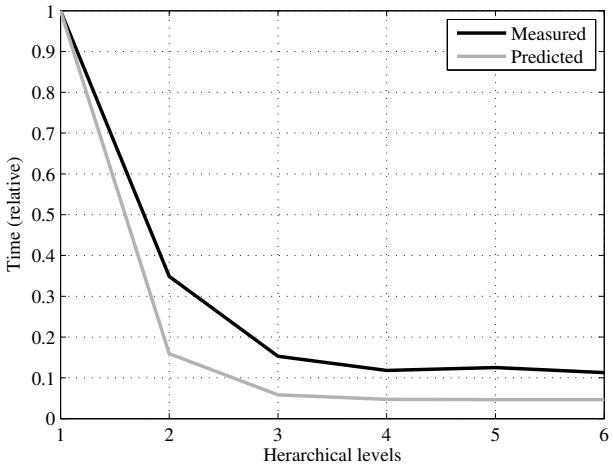


Figure 5.11: *Time consumption in converting distance maps to 3D foreground voxels using a different number of hierarchical octree levels. The value 1 on the x-axis corresponds to only using voxels with 5 cm on each side, 2 corresponds to also using voxels with 10 cm on each side, and so forth. Thus, with 6 levels, the following voxel sizes are used: 160, 80, 40, 20, 10, and 5 cm. A resolution scale factor for the 2D foreground masks of 4 has been used for all measurements.*

Four hierarchical levels provide around 88% reduction in time consumption on the short development data set. Additional levels do not affect the time consumption significantly, according to both the measured and the predicted time consumption. For these reasons, a 4-level hierarchical octree structure will be used as the default setting for the system. This is listed in Table 5.1 together with other relevant parameter settings used as default in our system.

Parameter	**Value**
Voxel size	5 cm
Hierarchy levels	4
Minimum number of cameras to see a voxel for it to be included in the grid	4

Table 5.1: *Parameters used in the 3D foreground estimation.*

Chapter 6

Feature Point Tracking

The concept of feature point tracking and its application to tracking moving objects or persons is both simple and intuitive. Local features, also referred to as *corners* or *Points of Interest* in literature, can be selected, which are easily recognisable from frame-to-frame, and which do not change significantly with movement and limited rotation. In this Chapter, a general introduction to known feature point trackers are given in Section 6.1 followed by a more in depth analysis of the KLT-tracker in Section 6.2. Finally in Section 6.3 an integration of feature point tracking for the purpose of person tracking is designed. Table 6.1 at the end of the chapter shows the parameters used as default in our system.

6.1 Feature Point Detection and Tracking Algorithms

The three basic steps required in a full feature point tracking algorithm are shown in Figure 6.1. Many algorithms for feature detection and tracking in 2D and even in 3D exist in literature. Some of these include all steps in Figure 6.1 while some only include either detection or frame-by-frame matching. In the following subsections a brief overview of relevant algorithms is given, including some of the most well known approaches.

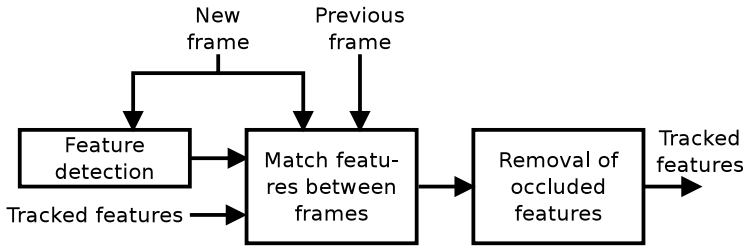


Figure 6.1: *Basic steps in feature point tracking.*

6.1.1 Algorithms for Feature Detection

One of the earliest feature detectors was proposed by H. Moravec in 1980 [62]. It works by comparing window W to all neighbouring windows using a sum of squared differences (SSD)

measure. If W is significantly different from all neighbours, it is chosen as a feature. Although this approach is able to detect many feature points, it is not well-suited for feature point tracking since the actual structure of W is not considered. Instead, only what causes a change in the SSD measure is used.

Another feature detector is the Harris corner detector [32]. This provides a much better performance than the Moravec detector, and has become the basis of several more modern detectors such as the the KLT-tracker and the SIFT-detector. It works by estimating the gradients in the image within a window W. It is detected whether the gradient direction vary much within the window and if it does, a corner is present.

6.1.2 Algorithms for Both Detection and Recognition of Features

A classic feature point tracker that has been used very much since its introduction as a tracker is the Kanade-Lucas-Tomasi (KLT) feature point tracker [56, 69, 74]. The KLT tracker was introduced in 1991 by Tomasi and Kanade [74] and is based on a feature matching algorithm developed in 1981 by Lucas and Kanade [56]. Even though it is not a new method it is still very popular for a variety of applications, including face coding and medical image registration, besides optical flow estimation and feature point tracking [4]. Many variations and improvements have been proposed. One of the most popular improvements was proposed in 1994 by Shi and Tomasi [69] (hence the name Kanade-Lucas-Tomasi), and several open source implementations are available of this version. The algorithm has also been extended to be able to track features in 3D using several cameras [5, 22].

Another very popular and robust feature detector is the Scale-Invariant Feature Transform (SIFT) proposed by Lowe in 1999 [55]. It has been used for a large variety of applications, including video stabilisation and different kinds of tracking [6, 79, 37, 77, 47]. Contrary to the pure detectors, the SIFT algorithm outputs a description of each feature which is invariant to scale, orientation and affine transformations. This makes it possible to locate the same feature in consecutive images and thus use it for tracking. Unfortunately, the algorithm is not very fast and is therefore not suited for real-time applications [55].

An attempt to decrease the computation time of the SIFT algorithm is the Speeded-Up Robust Features (SURF) introduced in 2006 by Bay et. al. [7]. This is designed to run in real-time by making use efficient use of Haar-like feature and integral images, originally introduces in image processing by Viola and Jones in 2004 [75]. It claims to provide similar or better quality than all previous methods, while being several times faster [7]. The exact time consumption of the algorithm is not known, however, and a full open source implementation is not available.

Since the KLT feature point tracker is both fast, reliable and available as an open source implemented, this is chosen. The following section explains the algorithm in more detail.

6.2 The KLT Feature Point Tracker

The KLT feature point tracking algorithm is explained in the Section 6.2.2 through 6.2.3, and available implementations are discussed in Section 6.2.4.

6.2.1 Tracking Features Between Frames

The algorithm used for tracking was developed in 1981 with a very different purpose than tracking in mind [56]. The purpose was to perform efficient template matching, denoted as image registration, to be used for stereo vision. For this reason no feature detection algorithm was presented simultaneously. The tracking algorithm is explained in this section, and the tracking equation is derived mostly following the approach of [10] and [74].

The basic principle of the algorithm is to find the displacement vector $\mathbf{d}$ that minimizes the following error function:

$$\epsilon = \iint_W [J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})]^2 w(\mathbf{x}) d\mathbf{x} \tag{6.1}$$

where:
$\mathbf{x} = \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ is a location vector,
$I(\mathbf{x})$ is a location in the original image,
$J(\mathbf{x}+\mathbf{d})$ is a displaced location in the new image, and
$w(\mathbf{x})$ is a weighting function, e.g. a Gaussian distribution.

The minimisation is done through Newton-Raphson-like iterations. The Newton-Raphson algorithm locates zeros of a function by performing the following step until convergence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

The problem at hand requires the error function in Equation (6.1) to be minimised. This can be done by the Newton-Raphson method by first differentiating with respect to $\mathbf{d}$ and then search for a zero. When $J(\mathbf{x}+\mathbf{d})$ is approximated by its first order Taylor expansion $J(\mathbf{x}) + \mathbf{g} \cdot \mathbf{d}$, this differentiation can be determined:

$$\epsilon \approx \iint_W [J(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})\mathbf{d} - I(\mathbf{x})]^2 w(\mathbf{x}) d\mathbf{x} \Rightarrow \tag{6.2}$$

$$\frac{\partial \epsilon}{\partial \mathbf{d}} \approx \iint_W [J(\mathbf{x}) - I(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})\mathbf{d}]\mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} \tag{6.3}$$

where:
$\mathbf{g}(\mathbf{x}) = \left[\begin{smallmatrix} g_x \\ g_y \end{smallmatrix}\right] = \begin{bmatrix} \frac{\partial J(\mathbf{x})}{\partial x} \\ \frac{\partial J(\mathbf{x})}{\partial y} \end{bmatrix}$ is the gradient vector of $J(\mathbf{x})$.

Setting the derivative of the error in Equation (6.3) to zero yields:

$$\iint_W [J(\mathbf{x}) - I(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})\mathbf{d}]\mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} = 0 \Leftrightarrow \tag{6.4}$$

$$\iint_W [J(\mathbf{x}) - I(\mathbf{x})]\mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} = -\left[\iint_W \mathbf{g}(\mathbf{x})\mathbf{g}^\mathsf{T}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x}\right] \mathbf{d} \Leftrightarrow$$

$$G\mathbf{d} = \mathbf{e} \tag{6.5}$$

where:
$G = \iint_W \mathbf{g}(\mathbf{x})\mathbf{g}^\mathsf{T}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x}$, and
$\mathbf{e} = \iint_W [I(\mathbf{x}) - J(\mathbf{x})]\mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x}$.

Equation (6.5) is the tracking equation that needs to be solved in every iteration. The 2-by-2 matrix G is given as (with the window $w(\mathbf{x})$ set to one for simplicity):

$$G = \begin{bmatrix} \iint_W g_x^2 d\mathbf{x} & \iint_W g_x g_y d\mathbf{x} \\ \iint_W g_x g_y d\mathbf{x} & \iint_W g_y^2 d\mathbf{x} \end{bmatrix} \tag{6.6}$$

where:
g_x and g_y are the entries of the gradient vector $\mathbf{g}(\mathbf{x})$, which are summed over the window W and weighted according to $w(\mathbf{x})$.

The tracking equation is a system of two equations with two unknowns (the entries of $\mathbf{d}$), and it therefore only has a unique solution if the rank of G is 2. This is the case if G has two eigenvalues above 0 (G cannot have eigenvalues below 0). This is only the case if the window under consideration contains texture and thus a varying gradient direction. If for instance the direction of all of the gradients in the window is identical, at least one of the eigenvalues of G will be 0, since the determinant of G is also 0:

$$\det(G) = W g_x^2 W g_y^2 - (W g_x g_y)^2 = 0 \tag{6.7}$$

This is also known as the *aperture* problem: Texture must be present in more than one direction for tracking to be possible [69]. This also makes intuitively sense, since a horizontal edge of course only can be tracked in the vertical direction.

One limitation of this approach is that the first order Taylor expansion used to approximate $J(\mathbf{x} + \mathbf{d})$ in Equation (6.2) must be good enough to enable each iteration to start with a better approximation of $\mathbf{d}$ than the previous one. This might not be the case if the displacement $\mathbf{d}$ is large compared to the size of the window W. A very large window can of course reduce this problem, but this simultaneously reduces the precision. For this reason tracking is in most implementations done iteratively, either varying the window size or by (similarly but faster) varying the resolution of the images. The latter approach works by fixing the size of the window to a relatively small area and building resolution pyramids of the images I and J. Tracking is then initiated in images with a very low resolution, and the outcome of this tracking is used as a starting point for tracking on images with a larger resolution. This approach is described e.g. in [13], and it allows precise tracking of features even when features have moved significantly between the images.

6.2.2 Feature Detection

The feature detection algorithm used in the KLT tracker was introduced in 1991 [74], 10 years after the original tracking algorithm, and it is in many ways very similar to the Harris corner detectorfrom 1988 [32]. It is designed specifically to choose the areas that can be most reliably tracked as features. Feature detection is for this reason based on the tracking equation, given in Equation (6.5). For a feature to be easy to track, the matrix G must have large eigenvalues. Therefore, features are chosen at the locations where the eigenvalues are largest and at least above a predefined threshold:

$$\min(\lambda_1, \lambda_2) > \lambda_{\text{thr}} \tag{6.8}$$

Thus, for possible window location in the image G is calculated. The locations are then sorted according to the minimum eigenvalue, and finally features are chosen from the top of this

list. Each new feature is only accepted if its window do not overlap with previously selected features. The feature detection can either be stopped after a desired number of features have been detected, or be continued until no more features have eigenvalues that fulfil Equation (6.8).

6.2.3 Removal of Occluded Features

Features can be removed for several reasons, including:

- **Out of bounds:** The iterative algorithm causes the window to get too close to the border of the image. This may mean, that the feature has left the view of the camera.
- **Small determinant of G:** If the determinant $\det(G)$ is 0 it means that its rank is less than 2. Thus, the matrix equation $G\mathbf{d} = \mathbf{e}$ either has no or an infinite number of solutions. If $\det(G)$ on the other hand is very small, the matrix equations *can* be solved, but not very reliably. Therefore, the feature point is considered lost if $\det(G)$ gets too small. This can happen if the feature point becomes occluded.
- **Max iterations:** The iterative algorithm does not converge within a fixed number of iterations. This may happen either if the feature point has become occluded, or if it has changed much e.g. due to rotation.

All of these reasons are based on frame-by-frame difference. It may also be an advantage to monitor, if the feature has changed too much over time. In the introduction paper of the KLT tracker from 1991 it is proposed to do this by calculating a dissimilarity between the feature window in the frame where it was initialised and in the current frame [74]. The dissimilarity measure is the RMS intensity difference between the two windows.

In the later paper "Good Features to Track" from 1994, which is often referred to as the origin of the KLT-tracker, it is proposed to use a more advanced dissimilarity measure [69]. Instead of calculating the RMS intensity difference between the window translated from the first to the current frame, an *affine* transformation model is used. Instead of matching the first and the current frame by using Equation (6.1), the following more general matching equation is used:

$$\epsilon = \iint_W [J(A\mathbf{x} + \mathbf{d}) - I(\mathbf{x})]^2 w(\mathbf{x}) d\mathbf{x} \tag{6.9}$$

where:

A is a transformation matrix.

The dissimilarity is then defined as the RMS intensity difference between the original window $I(\mathbf{x})$ and the window of the transformed feature in the current frame, $J(A\mathbf{x} + \mathbf{d})$. While the is a much more difficult system of equations to solve, experiments in [69] show that it provides a more reliable detection of which features that have become occluded.

6.2.4 Implementation of the KLT Tracker

Several open source implementations of the KLT-tracker are available, including a full implementation by Birchfeld et. al. [73], a partial implementation in OpenCV [13], and in implementation for NVidia GPU's [70]. Here, the implementation by Birchfeld et. al. is chosen since this is the only one available that includes the affine dissimilarity check while not being dependent of specific hardware.

6.3 Feature Points for Person Tracking

The KLT-tracker is able to initialise feature points that can be tracked, track them between frames, and remove them if they change too much, e.g. due to occlusion. This means that if a feature point is located on a person, it will be inclined to follow that person. The feature points must first be initialised on known persons, however. Also, if a feature point by mistake looses the track of a person there must be some way to remove it again. These subjects are discussed in the next two subsections.

6.3.1 Initialisation of Feature Points

When initialising new feature points, two sources of information are available indicating the position of the persons: *Targets* and *foreground masks*. The targets are the previously tracked positions of all known persons in the scene, and the foreground masks indicate areas of recent motion, as described in Chapter 4. Based this information, the feature points can be initialised on the targets. The approach is illustrated as a flowchart in Figure 6.2.

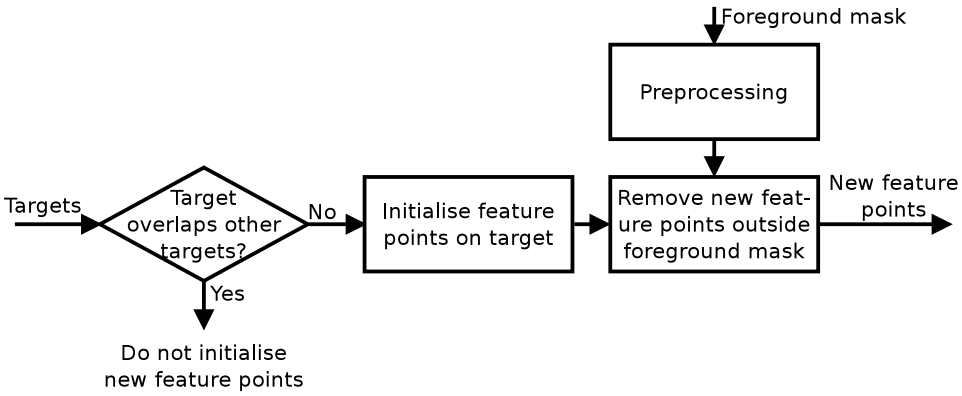


Figure 6.2: *Procedure for initialisation of new feature points on targets.*

Two aspects must be considered to ensure that new feature points are initialised correctly: They must not be initialised on a wrong person and they must not be initialised on the background. To ensure that points are not initialised on a wrong person, the position of the targets relative to other targets (if any) must be determined. If two targets seen from a particular camera are located behind each other, initialisation of feature points on either of them cannot be guaranteed to be located on the correct person. This is illustrated in Figure 6.3, where feature points can only be safely initialised on target 3.

When a target seen from a particular camera is not overlapping other targets, feature points can still risk being initialised on the background. This can be partially prevented by only initialising feature points where movement has recently occurred by comparing with the available foreground mask. While feature points can still be falsely initialised on recently moved objects, most points outside the actual person can be prevented. The complete feature point initialisation algorithm is illustrated in Figure 6.4.

Each feature point corresponds to a window, as explained in Section 6.2, and for our tracker a window size of 7×7 pixels are used. This means that points located on the border of a person also include part of the background. The more background a feature point includes, the more

likely it will be to stick to the background instead of the person. To avoid initialising points with much background, the foreground mask is first preprocessed using morphology, as shown in Figure 6.4d. Closing is first applied to remove noise and erosion is thereafter applied to shrink the mask.

6.3.2 Removal of Odd Feature Points

The KLT-tracker is able to eliminate feature points in two distinct ways: If the frame-by-frame difference becomes too large, and if the difference between the originally initialised feature points and the currently tracked feature points becomes too large (measure after an affine transformation) [69]. An additional elimination function must be included here since the feature points occasionally might be initialised on the background or move to background similar to its target. This is done by measuring the distance between the target and feature point in the view of the camera (see Figure 6.3). If the distance for a point becomes very large it is removed. This approach has proven to work, even though more sophisticated approaches are definitely possible.

6.3.3 Resolution

Even though the KLT-tracker is fast compared to other tracking algorithms, very high resolution tracking cannot be done in real-time. The time consumption of the algorithm when using the development data set in full resolution has been measured to 1.65 seconds (cf. Figure 13.1 in Chapter 13). Therefore the quality of the tracking using different reductions has been compared, and Figure 6.5 shows the result for the full resolutions and a scale factor of 4.

It is worth noting that some persons actually have *more* feature points when using reduced resolution images, for instance the middle person in Figure 6.5c and 6.5d. This is caused by two things: Firstly, when using higher resolution, the same window size for the features was used.

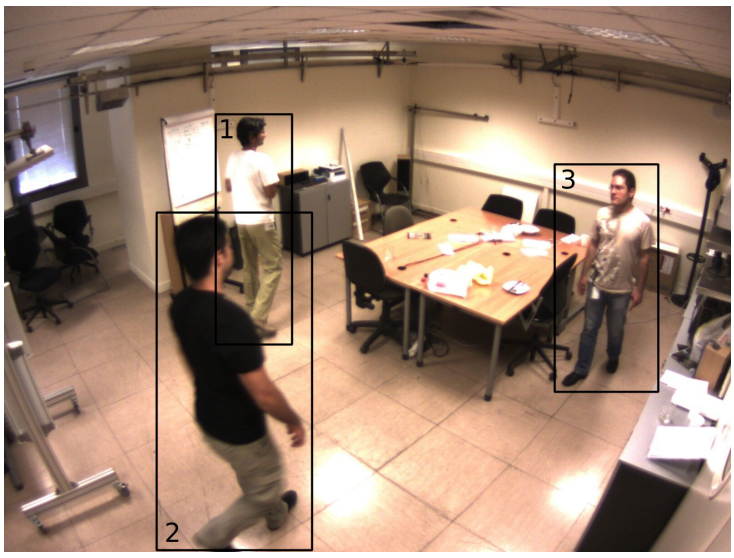


Figure 6.3: *Feature points are only initialised where targets do not overlap. This means that feature points can be initialised on target 3 in this example, but not on target 1 and 2.*

(a) Target. **(b)** Initialised feature points. **(c)** Mask. **(d)** Processed mask. **(e)** Remaining feature points target.

(f) Complete image.

Figure 6.4: *Initialisation of new feature points. A target that do not overlap other targets are first found (a), and feature points are then initialised (b). The processed foreground mask (d) is used to remove feature points initialised on the background, and the final result is shown in (e) and (f).*

(a) Frame 393 in full resolution.

(b) Frame 393 in downscaled resolution.

(c) Frame 1136 in full resolution.

(d) Frame 1136 in downscaled resolution.

(e) Frame 1354 in full resolution.

(f) Frame 1354 in downscaled resolution.

Figure 6.5: *Feature points in three different frames using full resolution frames to the left and frames with a resolution reduced by a factor of 4 to 400×300 pixels to the right.*

This causes the algorithm to search for smaller features, which might not be significant enough if the images are slightly blurred. Secondly, to limit time and memory consumption, a limit of 30 feature points per target per camera was used. With the full resolution images, many points can be grouped on one small background object, and thus few points might be *left* for the real persons. A the lower resolution reduces the number of points that can be located in a small area.

A scale factor of 4 on the development data set corresponds to using image resolutions of 400×300 for the corner cameras. Experiments have shown that this is the minimal resolution that does not significantly reduce the number of points that can be located, and therefore a minimum width of 400 pixels is chosen as default. This reduces the time consumption for feature point tracking in all cameras to 90 ms per frame, cf. Chapter 13.

The paramenters used for the feature point tracking are given in Table 6.1.

Parameter	**Value**
Window size for feature points	7×7 pixels
Maximum number of feature points per target	30
Preprocessing closing kernel	3×3 pixels
Preprocessing erosion kernel	3×3 pixels
Minimum width for images	400 pixels

Table 6.1: *Parameters used in the feature point tracking. Many additional features can be adjusted for the KLT feature tracker, but these have been left unchanged in the used implementation [73].*

Part III

Tracking System

Contents

The person tracking algorithms are designed in this part based on on algorithms for foreground detection and feature point tracking designed in the previous part. In Chapter 7, a general framework for person tracking is presented, and algorithms for managing the targets are designed. Methods to track known targets are introduced from a theoretical perspective in Chapter 8. Based on this, particle filter person tracking is designed in the final two chapters of this part.

Chapter 7

Framework and Target Management

The purpose of this chapter is to present a general framework for tracking multiple persons. The tracking framework is introduced in Section 7.1, and the various utility functions required in a tracking system such as initialisation etc. are grouped in *Target Management*. These functions are designed in the remaining sections of this chapter.

7.1 Tracking Framework

A proper tracking system must include everything necessary to track people in the scene under surveillance, including new people entering the scene and people leaving the scene. Whenever a person is inside the scene, the system must output a *target* indicating the state including the position of this person. A tracking system must include functions to initialise new targets, remove obsolete targets and estimate the state of existing targets. A general tracking framework is illustrated in Figure 7.1 and each of the main tasks are described here:

State estimation: The core of any tracking system is the state estimation algorithm, which has the purpose of estimating the state of the targets in each frame based on the states from the last frame as well as the measurement data. A state includes the location(s), but might also include other characteristics such as size or velocity. The measurement data to be used for state estimation were discussed Part II, and state estimation techniques are presented in Chapter 8. The state estimation algorithms for our system are designed in Chapter 9 and 10.

Target management: Target management includes initialisation and elimination of targets. To be able to initialise new targets, people present in the scene must be detected first. When a person is detected that cannot be matched with an existing target, a new target can be initialised. Targets can become obsolete either if the measurement data contains too little evidence for them to be tracked properly, or if several targets seem to be following the same person. Such targets must be removed. Target management for our system is designed in the following sections in this chapter.

Reasoning: Reasoning is not necessary in a general tracking system whose only purpose is to detect and track people present in the scene. However, if it is intended for a more elaborate application, post-processing can be integrated here. As described in the problem

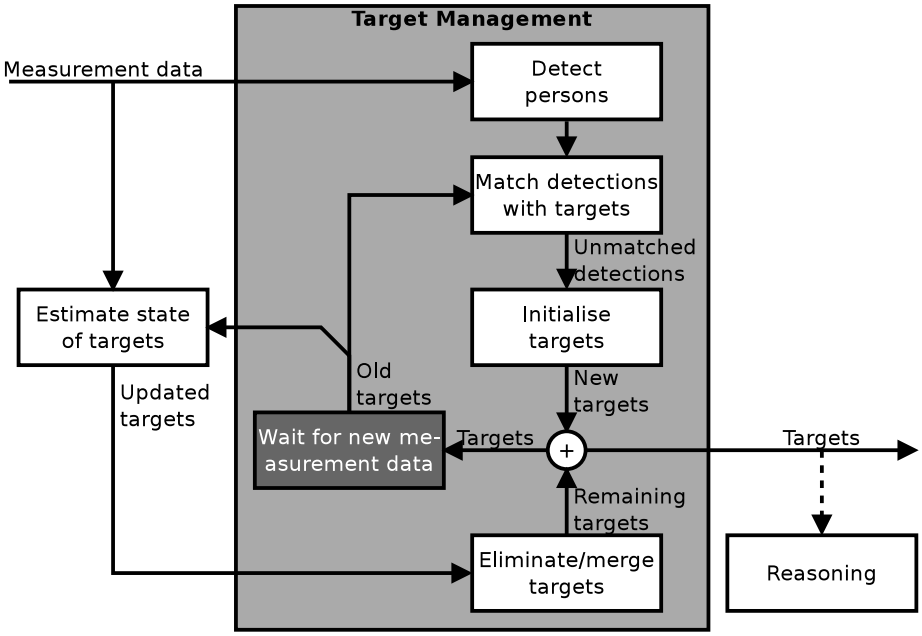


Figure 7.1: *Tracking framework.*

formulation, Chapter 3, one purpose of our system is to determine the mobility of persons in the scene and to detect if a person falls. If all persons can be tracked correctly, this is simple to detect by applying filters to the detected position and height of each person. The precise posture and mobility detection algorithms are developed and implemented for this project are presented in Appendix B.

Most of the functions in the tracking framework presented in Figure 7.1 are part of the target management. This is designed for our system in the following sections. Its main purpose is to initialise new targets when new people enter the scene and eliminate obsolete targets when people have left the scene. To allow initialisation of a hypothetical target without announcing it before sufficient evidence in the measurement data has been collected, a measure of reliability is introduced. A new target can then be introduced in a frame, even if it is impossible to differ between noise and a real new person in the scene. Only if the person can be detected for several consecutive frames, the target will be announced.

In Section 7.2 an algorithm for detection of persons is designed, which forms the basis for both initialisation of targets as well as updating of the reliability measure. The reliability is examined more closely in Section 7.3, and the final initialisation and elimination algorithms are designed in Section 7.4.

7.2 Detection of Persons

The tracker uses two different modalities for tracking, namely foreground and feature points. As described in Chapter 6, feature points are initialised on existing target. This means that only the foreground information can be used to detect new targets, this information comes in the form of foreground voxels from the 3D foreground estimator.

One way to perform target detection is to divide the foreground into groups of coherent voxels by performing BLOB analysis and then consider each large BLOB as a detected person. This method will be able to separate all the persons in the scene if all persons are fully separated and all voxels belonging to the same person are fully connected. The second constraint is rarely a problem. The constraint that different people must be fully separated can easily be a problem, however. When two or more persons are standing close together, this method will detect them as a single person.

Another approach is to use a more general clustering algorithm to count the number of voxel groups. One elementary but very popular method of unsupervised clustering is k-means [25]. In the basic form this algorithm needs to know the number of clusters k in advance. If, in each frame, the number of clusters and their locations from the previous frame are used as a starting point, then the number of clusters only need to be adjusted whenever a person enters or leaves the scene. This can be detected by measuring the internal distance in each cluster as well as the distance between the cluster. If an internal distance becomes very large, a new cluster can be initialised, and when the distance between two clusters becomes very small, they can be merged. This method will be able to separate all persons if they are located far from each other, and it will give a good guess of the location of the persons even if they are standing very close together.

The purpose of detection is partly to be able to initialise new targets on people entering the scene and partly to update the reliability. Of these, the first is by far the most important, since the reliability can also be updated according to the measurement data itself (see Section 7.3). When new people are entering the scene they are very likely to be easily seperable. BLOB analysis is chosen partly for this reason, and partly because this method does not require any post processing to detect new persons entering the scene.

7.2.1 Top/Down BLOB Analysis

The basic BLOB analysis algorithm can be adjusted in several ways to be suitable to the purpose of target detection. A typical situation is illustrated in Figure 7.2, where two persons are positioned close together. It is indicated that the head and the upper body can be separated, while they are connected closer to the ground. This can easily happen, e.g. because of shadows and limited view from the cameras. Near their heads they will, however, often be easier to separate, partly because the heads are located farther from the ground, and partly because the head is thinner than the rest of the body. The idea of the top/down BLOB analysis algorithm is to separate such a BLOB into two different clusters by starting at the top, where they *can* be separated. When a connection point between two clusters is reached, its height is compared with a threshold, τ_1. The clusters are merged only if they are connected above this τ_1. To prevent that people sitting down or fallen are separated into many BLOBs, an additional threshold τ_2 is used. If the height of one of the clusters relative to the connection point is less than τ_2, they are also merged.

7.2.2 Efficient Implementation of Top/Down BLOB Analysis

To make the algorithm run in real-time the computational complexity is analysed here. There are a number of parameters that affect the time consumption, including:

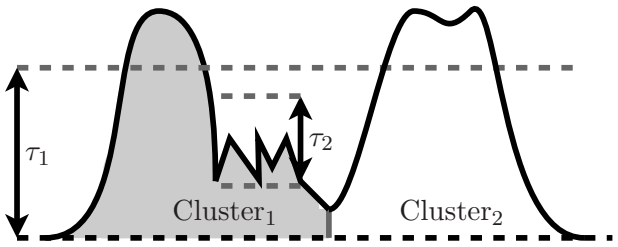


Figure 7.2: *Slice through the 3D foreground in a typical situation. Two people are present, but the foreground is connected into one single, coherent BLOB of voxels. The two persons can be separated by using top/down BLOB detection, and only merging clusters that are connected above a threshold τ_1. Clusters with individual heights compared to the connection point below τ_2 are also connected.*

- The total number of voxels in the room v_{room}.
- The number of foreground voxels v_{FG}.
- The number of neighbouring voxels to each voxel $v_{\text{neighbours}}$ - that is, if e.g. 6-, 18- or 26-neighbourhood is used.
- The number of clusters n_{clusters}.
- The number of foreground voxels for a person. Each cluster correspond in must cases to a person. The number of foreground voxels constituting each person can therefore be approximated by $v_{\text{person}} \approx \frac{v_{\text{FG}}}{n_{\text{clusters}}}$.

Only the values v_{room}, v_{FG} and n_{clusters} depend on the room size and/or the number of persons in the scene. Therefore, the complexity of the algorithm is sought minimized with respect to these. The values v_{room} and v_{FG} also depend on the resolution, which has been chosen to 5 cm voxels, cf. Section 5.1. The value $v_{\text{neighbours}}$ can be chosen to something between 6 to 26, where $v_{\text{neighbours}} = 6$ means that only voxels that share four corners are neighbour voxels, while $v_{\text{neighbours}} = 26$ will include all voxels that share at least one corner with the voxel under consideration. To minimise the computation time, $v_{\text{neighbours}}$ is chosen to 6, since the final result of clustering is almost the same in the development data set (described in Section 1.1).

The room must be searched for foreground voxels from ceiling to floor. Therefore, the voxels are first sorted. This limits the number of voxels that need to be tested for membership of a cluster to half of the neighbours. The algorithm is written as pseudo code in Figure 7.3.

The test to determine if a neighbouring voxel belongs to a cluster is central to the algorithm, because it must be carried out for each foreground voxels, for each neighbour. It can be done in two ways:

Span a matrix of the entire room where each entry correspond to a voxel. If the voxel belongs to a cluster, the matrix entry holds the cluster number. If the voxel is not a foreground voxel, it holds "-1". When the next foreground voxel (from the sorted list) is to be categorised, only three table look-ups are performed. However, time and memory are required to initialise the grid in the beginning of each iteration. The number of calculations $\mathcal{C}_{\text{matrix}}$ and the algorithm complexity is:

$$\mathcal{C}_{\text{matrix}} = v_{\text{room}} + v_{\text{FG}} \cdot \frac{v_{\text{neighbours}}}{2} \tag{7.1}$$

$$\sim \mathcal{O}\left(v_{\text{room}} + v_{\text{FG}}\right) \tag{7.2}$$

- Sort foreground voxels with according to there position in the room.
- **For all** foreground voxels
  - Test neighbour foreground voxels for membership of a cluster.
  - **If** at least one neighbour cluster is found
    - Include this voxel in the cluster.
    - **If** more than one neighbour cluster has been found
      - **If** the clusters are to be merged (according to the conditions in Figure 7.2), merge.
  - **Else**
    - Initialise a new cluster with this voxel.

Figure 7.3: *Pseudo-code for top/down BLOB analysis used to group the foreground voxels into clusters.*

where $\frac{v_{\text{neighbours}}}{2} = 3$ can be neglected in the complexity since it is a constant.

Lists of foreground voxels belonging to each cluster. For each foreground voxel, the list of voxels associated with each cluster is searched. As the voxels are put into the clusters from a sorted list, the clusters can also be kept sorted by just resorting whenever two clusters are merged. When $v_{\text{neighbours}} = 6$, 3 voxels have to be tested, and because of the sorted lists, one of these is the last in a cluster (if it contains foreground). The search for the two remaining neighbour voxels can be carried out by using the binary search algorithm, which as the complexity $\mathcal{O}(\log(n))$ [26]. The number of calculations $\mathcal{C}_{\text{list}}$ required and the algorithm complexity is then:

$$\mathcal{C}_{\text{list}} = v_{\text{FG}} \cdot n_{\text{clusters}} \cdot \left(\frac{v_{\text{neighbours}}}{2} - 1\right) \cdot \log\left(\frac{v_{\text{FG}}}{n_{\text{clusters}}}\right) \tag{7.3}$$

$$\approx \frac{v_{\text{FG}}^2}{v_{\text{person}}} \cdot \left(\frac{v_{\text{neighbours}}}{2} - 1\right) \cdot \log(v_{\text{person}})$$

$$\sim \mathcal{O}\left(v_{\text{FG}}^2\right) \tag{7.4}$$

where the constants are disregarded from the complexity and $\frac{v_{\text{FG}}}{n_{\text{clusters}}}$ is replaced by v_{person}. Note that v_{person} can also be considered a constant, since each person can be expected to cause a similar amount of foreground.

The lower memory requirement as well as the decoupling of the number calculations from the room size in Equation (7.3) is an advantage for the list approach. On the other hand, the linear complexity in Equation (7.2) compared to the quadratic complexity in Equation (7.4) is of course an advantage for the matrix approach. However, it is possible to optimise the list approach considerably by using *bounding volumes*. A bounding volume is for this purpose defined as a box containing a cluster plus one voxel in each direction. These can enable new voxels to be compared only with clusters that include them in their bounding volume [26]. In this way, a simple test determining if the voxel is located inside a box can replace many of the clusters searches. In most cases only one cluster is left after this step. If this is assumed to be true, Equation (7.3) can be split up into two:

$$\mathcal{C}_{\text{list}} \approx \mathcal{C}_{\text{list, one\_cluster}} + \mathcal{C}_{\text{list, bb\_test}} \tag{7.5}$$

where $\mathcal{C}_{\text{list, bb\_test}}$ is the number of calculations required to test all foreground voxels against all bounding boxes. The number of required calculations can be rewritten as (again disregarding constants in the complexities):

$$\mathcal{C}_{\text{list, one\_cluster}} = v_{\text{FG}} \cdot n_{\text{clusters}} \cdot \left(\frac{v_{\text{neighbours}}}{2} - 1\right) \cdot \log\left(\frac{v_{\text{FG}}}{n_{\text{clusters}}}\right) \tag{7.6}$$

$$\approx v_{\text{FG}} \cdot \left(\frac{v_{\text{neighbours}}}{2} - 1\right) \cdot \log\left(v_{\text{person}}\right)$$

$$\sim \mathcal{O}\left(v_{\text{FG}}\right) \tag{7.7}$$

$$\mathcal{C}_{\text{list, bb\_test}} = v_{\text{FG}} \cdot n_{\text{clusters}} \approx \frac{v_{\text{FG}}^2}{v_{\text{person}}} \tag{7.8}$$

$$\sim \mathcal{O}\left(v_{\text{FG}}^2\right) \tag{7.9}$$

By comparing Equation (7.3) and 7.8 it is clear that the bounding box approach removes the constant $\left(\frac{v_{\text{neighbours}}}{2} - 1\right) \cdot \log\left(v_{\text{person}}\right)$ from the quadratic complexity part. The complexity is still quadratic, however, so if the amount of foreground v_{FG} could be arbitrary high, the matrix approach would be preferable. As described in Chapter 3, the purpose of this system is to monitor indoor environments with one or few people present, which means that the amount of foreground present will be limited. Partly to avoid making the algorithm dependant of the room size, and partly to limit the memory requirements, the list approach is chosen.

7.3 Reliability of Targets

When a person enters the scene, a new target is supposed to be initialised. It might not always be possible to tell targets and noise completely apart, though. A reliability measure of each target can make it possible to initialise non-reliable targets, and then attempt to track them for some time without announcing them. Only if evidence in the measurement data support the hypothesised target for some duration, it will become reliable. Thus, the reliability measure can be seen as a filter on the amount of evidence present in the measurement data, supporting a target. The idea is illustrated in Figure 7.4.

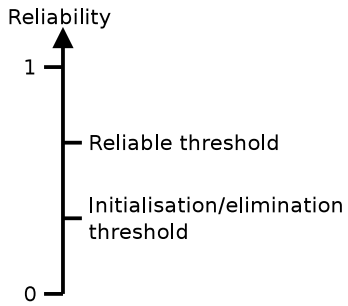


Figure 7.4: *The reliability measure can be between 0 (non-reliable) and 1 (very reliable), and the reliable threshold controls how reliable a target must be, before it is reported by the system.*

7.3.1 Present Targets

An important design decision is what evidence to require before increasing the reliability. A basic approach is to use the output of the target detection algorithm described in the previous section. If a tracked target can be matched with a detected target, declare it as *present*. The reliability of all present targets can the be increased, while the reliability of the remaining targets can be decreased. A problem with this approach is that targets that have faded into the background will be lost - even though they might be perfectly trackable if many feature points are present. To keep targets in such situations, the amount of evidence must also be taken into account, given as the likelihoods (see Chapter 9 for explanation of the likelihood values). If the likelihood from either the foreground or the feature points is large, the target should also be declared as present. Figure 7.5 illustrates this concept.

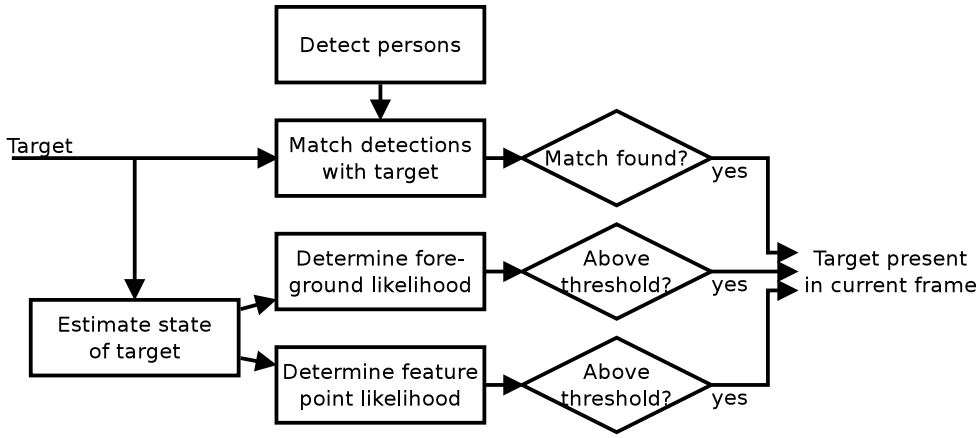


Figure 7.5: *A tracked target is declared present in a frame if either it is matched with a detected target, or if its likelihood is large for either foreground or feature points (the likelihood is explained in Chapter 9). The reliability value is increased for detected targets and decreased for undetected targets.*

7.3.2 Reliability Update Rates

A number of goals must be taken into consideration when designing the reliability filter:

1. **Real persons:** New targets initialised on real persons must become reliable as fast as possible.
2. **Moved objects:** New targets initialised incorrectly, e.g. on an opened door or a moved chair must not become reliable before they are incorporated into the background.
3. **Old targets:** It must be possible for a reliable target to remain reliable for a long duration, even if it has little supporting measurement data and cannot be declared as present. This should prevent targets that are stationary a long distance from any doors or other possible exit points from being lost.

Obviously, item 1 and 2 conflict. To meet both of these requirements, the difference must be examined between moved objects and real persons that have recently entered the scene. Both objects and persons have moved - otherwise they would not have become foreground and would not have caused the initialisation of a new target. However, an object is expected to remain

stationary when no person is moving it. On the other hand, a person that has just entered a room will usually keep moving, at least for a few seconds. Thus, mobility can be used to distinguish objects and persons. Mobility detection is discussed in Chapter B.2, and here it is just assumed that each target is either mobile or immobile.

Item 3 suggests that targets that have been reliable for a long duration should have a small learning rate. This means that the learning rate of a target must depend of the age of that target. All of these requirements is combined into a IIR-filter in the following equation, and is illustrated in Figure 7.6a.

$$r_n = (1-k)r_{n-1} + k \cdot P_n \quad \text{with } k = \begin{cases} k_{\min} & \text{if target is immobile} \\ \min(\frac{1}{n} + k_{\min}, k_{\max}) & \text{if target is mobile} \end{cases} \tag{7.10}$$

where:
r_n is the reliability of a target at frame n, with r_0 set to the reliability value $r_{\text{eliminate}}$ to just prevent the target from being eliminated.,
P_n is 1 if the target is present and 0 otherwise, and
k is the reliability learning factor

To make sure that stationary targets are incorporated into the background before they become reliable as required in item 3, the learning rate of imobile targets $k_{\min}$ must be compared to the learning rate of the adaptive background model. The update equation for the background model is given in Equation (4.4) on page 20, and is repeated here:

$$\omega_n = (1-\alpha)\omega_{n-1} + \alpha \cdot M_n \tag{7.11}$$

where:
ω_n is the weight of a Gaussian distribution, and ω_0 is set to a predefined value ω_{init},
M_n is 1 if the pixel value matches the distribution and 0 otherwise, and
α is the weight learning factor.

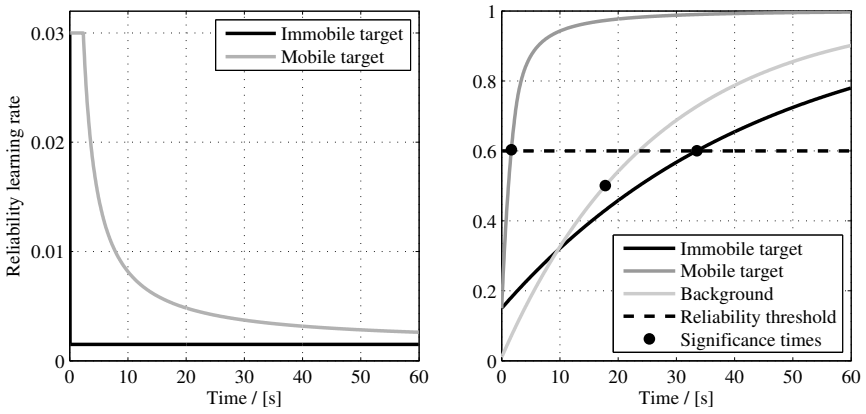
By comparing Equation (7.11) with Equation (7.10) for immobile targets it is clear that $k_{\min}$ must be adjusted to be significantly lower than α. The reliability and background updates are compared in Figure 7.6 with $\alpha = 0.00256$, $k_{\min} = 0.0015$ and $k_{\max} = 0.03$. When a target is considered reliable if it has a reliability value above 0.6, this gives a margin of at least 10 seconds from fading into the background to potential reliability, which is considered sufficient.

7.4 Initialisation and Elimination of Targets

While both initialisation and elimination has already been superficially discussed in the previous sections, this section gives a complete and systematic overview over the area.

7.4.1 Initialisation of New Targets

Target initialisation is mostly based on the target detection algorithm described in Section 7.2. This algorithm simply and efficiently separates 3D foreground voxels into multiple clusters, where each clusters corresponds to a detected person. Note that even though the detection



(a) Reliability learning rates.

(b) Reliability updating of targets compared with updating of the background.

Figure 7.6: *Reliability of newly initialised mobile and immobile targets. In (b), the reliability of new targets is compared with the learning speed of the adaptive foreground detection, described in Chapter 4 to make sure that moved objects are incorporated into the background before its target becomes reliable. The background curve is marked where it crosses 0.5, since the object in all practical situations latest at that point will have become incorporated into the background. The reliability threshold is set to 0.6 to give at least 10 seconds buffer between background incorporation and reliability.*

algorithm is not always able to separate people located very close together, this is not an issue for initialisation, since people entering the scene are moving, and the separation is only required in a single frame. Only in very crowded areas it might be impossible to ever separate a new person, and our intended is not designed for such situations (cf. Chapter 3).

The output of target detection is a number of clusters, each constituted by a 2D location. These locations must be matched to the locations of the targets in the previous frame. This is a classical assignment problem: The cost of a match is set to the Euclidean distance between the 2D positions, and the set of matches which give the minimum cost is chosen. The assignment problem is efficiently solved using the Munkres assignment algorithm, also known as the Hungarian algorithm [11]. Matches larger than a predefined threshold are discarded subsequently, and unmatched detected clusters are then used to initialise new targets.

Two special situations are taken into account to increase the performance of the tracker. To decrease the risk of initialising targets on moved objects, targets are only initialised on detected clusters, that are high enough to be considered standing persons (see Section B.1 on page 123). Since people entering the scene are practically always standing, this only reduces the performance at the beginning of a recording. Attached to the recordings used to test our tracker from CLEAR 2007 are a number background frames, and when switching to the actual recordings, people might appear sitting instantaneously [19]. For this reason, targets are also initialised on low targets in the first non-background frame. Targets initialised in this frame are furthermore also made reliable immediately. The complete target initialisation design is illustrated in Figure 7.7.

7.4.2 Elimination of Obsolete Targets

Targets can become obsolete for several reasons:

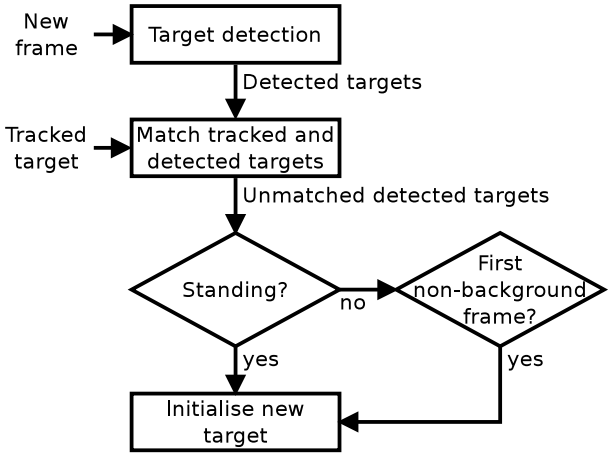


Figure 7.7: *Initialisation of new targets.*

1. **Tracking nothing:** The target has lost its person, and is not tracking anyone anymore.
   (a) **Person left:** The person has left the scene.
   (b) **Tracking error:** The person is still present, but the tracker is not tracking anything.

2. **Tracking wrong person:** The target has lost its person, and has instead begun tracking a wrong person, which is already tracked by another target.

Item 1b and 2 are due to tracking errors, and item 2 is easily handled simply by merging targets that are located very close for several consecutive frames. Item 1b can in general not be handled efficiently because it conflicts when the desire to be able to maintain targets, even though they have little supporting evidence in the measure data. Thus, a compromise must be found by setting elimination threshold on the reliability appropriately, see Figure 7.4.

Item 1a is the only situation, where the target should be eliminated even though tracking has not failed. Thus it makes sense to be particularly aware of people leaving the tracking area. Different approaches can be taken to this:

- **Exit zones:** Persons can only leave the scene at specific locations, e.g. at a door. Thus, one solution is to make it very easy for a target to be eliminated in specific zones.
- **Fast disappearance:** When a persons leaves the scene, the amount of 3D foreground inside the target will be reduced very fast. Note that this can only happens when the person leaves the room - if he is stationary and thus fades into the background, the amount of 3D foreground present will be reduced more slowly.

While both approaches definitely have the potential to work, the first approach need tuning for each specific scene. Therefore the second and more general approach is chosen. it is implemented by maintaining an average amount of foreground present inside a target. If the average is large, but suddenly drops close to zero, the target is eliminated.

Chapter 8

Bayesian State Estimation

Recursive Bayesian estimation is a general approach to estimate the probability density function (pdf) of a process. The basic idea is to estimate the state of a process at a certain time using all available measurement data up to and including that point in time. When the process state at time k is denoted $\mathbf{x}_k$ and the measurement data is denoted $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k = \mathbf{z}_{1:k}$, the estimate of the pdf of the state using all available measurement data is denoted $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. Because all measurement data is used to estimate this pdf, including data from time index k, it is known as the *posterior* pdf. In contrast, the *prior* pdf is given as $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$. The motivation for calculating the posterior pdf is, that this makes it possible to estimate the state itself using any desired criterion [68]. For example, either the Maximum A Posteori (MAP) or the Minimum Mean-Square Error (MMSE) estimate can be used. As an example, the MMSE estimate of the state $\mathbf{x}_k$ is shown here:

$$\hat{\mathbf{x}}_{k|k}^{\text{MMSE}} = \mathbb{E}\{\mathbf{x}_k|\mathbf{z}_{1:k}\} = \int \mathbf{x}_k \cdot p(\mathbf{x}_k|\mathbf{z}_{1:k}) d\mathbf{x}_k \tag{8.1}$$

where the notation $\hat{\mathbf{x}}_{k_1|k_2}$ means the estimate of $\mathbf{x}$ at time k_1 given measurements $\mathbf{z}_{1:k_2}$.

8.1 Conceptual Bayesian Approach

The Bayesian approach assumes that the process is a Markov process of order one with unobservable state, meaning that the state $\mathbf{x}_k$ does only depend on $\mathbf{x}_{k-1}$ and not of states previous to $k-1$ and that the true state cannot be measured directly. The process and measurement models can be written respectively as:

$$\textbf{process:} \qquad \mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}) \tag{8.2}$$

$$\textbf{measurement:} \qquad \mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{w}_k) \tag{8.3}$$

where:
$\{\mathbf{v}_{k-1}, k \in \mathbb{N}\}$ and $\{\mathbf{w}_k, k \in \mathbb{N}\}$ are i.i.d. noise sequences, and
$\mathbf{f}_k$ and $\mathbf{h}_k$ are (possibly non-linear) functions.

When estimating the posterior pdf of $\mathbf{x}_k$, both the previous state $\mathbf{x}_{k-1}$ and the measurement $\mathbf{z}_k$ must be taken into account. In general, this is done through a *prediction* and an *update* equation. The prediction equation predicts the state of $\mathbf{x}_k$ from $\mathbf{x}_{k-1}$ by using a state evolution

model $p(\mathbf{x}_k|\mathbf{x}_{k-1})$, and the update equation takes the output of the prediction equation and updates according to the measurement data $\mathbf{z}_k$ using the Bayes' rule: [58]

$$\textbf{predict:} \qquad p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1} \tag{8.4}$$

$$\textbf{update:} \qquad p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{8.5}$$

where:
$p(\mathbf{z}_k|\mathbf{x}_k)$ is a likelihood function, and
$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k$ is a normalising constant.

Note that the likelihood function $p(\mathbf{z}_k|\mathbf{x}_k)$ is not to be confused with a probability function, since it is the outcome measurement $\mathbf{z}_k$ that is known, while the state $\mathbf{x}_k$ is unknown. For this reason it is often denoted $L(\mathbf{x}_k|\mathbf{z}_k)$, which more intuitively indicates that the function gives the likelihood of the state $\mathbf{x}_k$ given the measurement $\mathbf{z}_k$. The likelihood function is not required to integrate over its variable $\mathbf{x}_k$ to 1, and this is why a normalising function is required in update Equation (8.5).

A wide variety of methods exist to implement Equation (8.4) and (8.5) and thus to estimate the posterior pdf of the state, $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. Two of the most well known groups, the Kalman filter and its extensions and the Particle filters are described in Section 8.2 and 8.3 respectively.

8.2 Kalman Filters

The Kalman filter was first developed in 1960 by R. E. Kalman [43]. It was in the following around 40 years the most important method for state estimation, e.g. for tracking, and it is still widely used.

The Kalman filter is under certain conditions an *optimal* solution to the Bayesian estimation problem defined in Equation (8.4) and (8.5) in the sense, that no algorithm can perform better [68]. It assumes that the posterior $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is always Gaussian and can thus be described completely by its mean and covariance. The conditions are that the state estimate from the previous time step $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ must also by Gaussian, and that Equation (8.2) and (8.3) can be rewritten: [68]

$$\textbf{process:} \qquad \mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_{k-1} \tag{8.6}$$

$$\textbf{measurement:} \qquad \mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{w}_k \tag{8.7}$$

where:
$\mathbf{v}_{k-1}$ and $\mathbf{w}_k$ are zero-mean Gaussian white noise sequences with known covariances, and
$\mathbf{f}_{k-1}$ and $\mathbf{h}_k$ from Equation (8.2) and (8.3) are linear and thus replaced by the matrices $\mathbf{F}_{k-1}$ and $\mathbf{H}_k$.

Under these conditions, it is relatively simple to implement the prediction and measurement equations from Equation (8.4) and (8.5). The predicted mean is e.g. estimated simply using the process matrix from Equation (8.6) as $\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1}$. The specific equations are outside the scope of this description, but can be found in [68].

8.2.1 Non-linear Variants of the Kalman Filter

The Kalman filter is optimal when the mentioned restrictions are met. One of these restrictions is that the process and measurement equations (8.2) and (8.3) must be linear, and this is rarely the case in real life. Several variants have been developed to deal with such situations, two of the most common being the *Extended Kalman Filter* (EKF) and the *Unscented Kalman Filter* (UKF). The EKF has been used for several decades, and works by assuming that Equation (8.2) and (8.3) can be rewritten as: [68]

$$\textbf{process:} \qquad \mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}) + \mathbf{v}_{k-1} \tag{8.8}$$

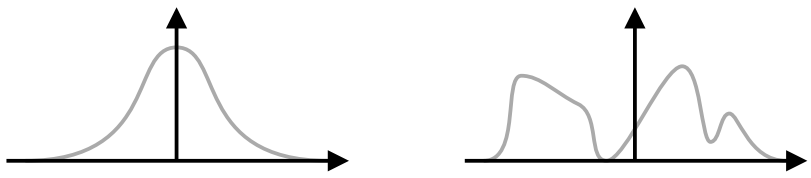
$$\textbf{measurement:} \qquad \mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k) + \mathbf{w}_k \tag{8.9}$$

where:
$\mathbf{f}_{k-1}$ and $\mathbf{h}_k$ are differentiable, possibly non-linear functions, and
$\mathbf{v}_{k-1}$ and $\mathbf{w}_k$ are additive white Gaussian noise with known covariances.

The principle in the EKF is, for each time step, to make local linearisations of $\mathbf{f}_{k-1}$ and $\mathbf{h}_k$ (by first order Taylor transforms) and their covariance matrices (by Jacobian matrices) around the current state. Like the EKF, the UKF also avoids assuming that the process and measurement equations are linear, and the UKF has become very popular since it was proposed in 1997 [39]. It attempts to give a better approximation of the non-linearity by only assuming that the posterior $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is Gaussian, instead of the assumptions in Equation (8.8) and (8.9). The non-linearity of the process and measurement models are incorporated by letting a large number of particles pass the models, instead of only relying on one local linearisation. Note, that both the EKF and the UKF are approximating filters and cannot be considered optimal.

8.3 Particle Filters

The Kalman filter and its variations all assume that the posterior distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is Gaussian. In many situations such as tracking, this might not be the case. Instead, the posterior can have an arbitrary shape as illustrated in Figure 8.1. The particle filters, first introduced as the CONDENSATION algorithm in 1998 by Isard and Blake [38], approximate such arbitrary shapes of the posterior density by using many particles with different hypothesised states. For our system, the posterior density cannot be assumed to be Gaussian, and it might even have more than one local maximum. Therefore it is chosen to design the state estimation based on particle filters instead of Kalman Filters. Different variants of relevant particle filters are presented in the following subsections to provide a foundation for the design of particle filters for our system.



(a) Kalman assumption for posterior. **(b)** Possible real-world posterior.

Figure 8.1: *The Kalman filter and its variations assume that the posterior distribution is Gaussian. In real-world situation it might have as arbitrary shape.*

8.3.1 Sequential Importance Sampling

Sequential Importance Sampling (SIS) is a technique to implement the conceptual Bayesian state estimation presented in Section 8.1 by Monte Carlo simulations that forms the basis for particle filters. Monte Carlo simulations can be described by comparing to deterministic modelling: Where a deterministic model can produce a single output hypothesis by using the most likely input, a Monte Carlo simulation produces a large number of discrete hypotheses by sampling inputs according to probability density functions. While the Kalman filter expresses the posterior distribution by a Gaussian defined by its mean and covariance, the central idea of the SIS algorithm is thus to represent the posterior by a set of particles with associated weights instead. Randomness is used in addition to a state evolution model to update the state of each particle, and the SIS algorithm approaches the optimal Bayesian estimator if the number of particles approaches infinity, $N_\mathrm{p} \to \infty$. If the true posterior is shaped as in Figure 8.1b, the particles with associated weights will be distributed in the same way. Because each particle has a hypothesised state, a particle filter can be seen as a multi-hypothesis tracker.

Ideally, the particles should be distributed according to the posterior $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. Usually it is not possible to sample the posterior, however, so instead a *proposal distribution* $\pi(\mathbf{x}_k|\mathbf{z}_{1:k})$ (also known as *importance density*) with the same support is used. This distribution can reflect both a state evolution model and additive noise. The principle of importance sampling is applied by calculating the weight of the particles according to the "error" in the proposal distribution compared to the posterior distribution. After k time steps, the weights must be given according to the joint posterior:

$$w_k^i \propto \frac{p(\mathbf{x}_{1:k}|\mathbf{z}_{1:k})}{\pi(\mathbf{x}_{1:k}|\mathbf{z}_{1:k})} \tag{8.10}$$

In this way, the particles with normalised weights will reflect the posterior as:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_\mathrm{p}} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i) \tag{8.11}$$

For the proposal distribution to be usable in the Bayesian framework, it must depend only on the previous state and measurements; that is $\pi(\mathbf{x}_k|\mathbf{z}_{1:k}) = \pi(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k)$. The weight update equation can then be rewritten as a recursive formulation, yielding the complete SIS algorithm in Figure 8.2 (refer to [68] for derivation).

The choice of proposal distribution is of major importance. In general, the optimal choice would be the proposal distribution, as this would eliminate the need for weights, c.f. Equation (8.10). When the estimation is required to be recursive, the optimal proposal distribution has been proven to be: [24]

$$\pi_\mathrm{opt}(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) \tag{8.12}$$

This equation is optimal in the sense, that the variance of the weights is minimised, and a lower variance is equivalent to a distribution of particles that is more similar to the posterior distribution. It is not always possible to sample from this distribution, however. Another popular choice as proposal distribution is the state evolution model (also known as the transitional prior): [68]

$$\pi(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{8.13}$$

- **For** $i = 1 : N$
  - Draw $\mathbf{x}_k^i$ from proposal distribution:

$$\mathbf{x}_k^i \sim \pi(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) \qquad (a)$$

  - Evaluate importance weight up to a normalising constant:

$$\tilde{w}_k^i = w_{k-1}^i \frac{L(\mathbf{x}_k^i|\mathbf{z}_k)p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)}{\pi(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k)} \qquad (b)$$

- **For** $i = 1 : N$, normalise importance weights:

$$w_k^i = \frac{\tilde{w}_k^i}{\sum_{j=1}^N \tilde{w}_k^j} \qquad (c)$$

Figure 8.2: *Pseudo-code for the SIS filtering algorithm [58].*

Comparing with Equation (b) in Figure 8.2, the weight update equation reduces to sampling the likelihood function with this proposal distribution:

$$\tilde{w}_k^i = w_{k-1}^i L(\mathbf{x}_k^i|\mathbf{z}_k) \qquad (8.14)$$

To sum up, the SIS algorithm implements the conceptual Bayesian estimation given in Section 8.1 by the use of Monte Carlo simulations. The general update and prediction equations given as Equation (8.4) and (8.5) can be directly compared with Equation (a) and (b) in Figure 8.2 if the proposal distribution is chosen as in Equation (8.13).

8.3.2 Particle Filter with Resampling

The SIS algorithm suffers from the problem that the variance of the importance weights will increase over time, no matter the choice of proposal distribution [24]. After some time, one of the normalised weights will be close to 1 and the remaining close to 0. Thus, much computational power will be spend on particles with almost no influence in the final estimate of the posterior distribution.

A technique to avoid this is to use *resampling* of the particles [68]. Resampling means that the set of particles replaced by a new set, where the particles in the new set are drawn from the particles in the old set with probabilities corresponding to the weights. That is, a particle with a large weight will probably be copied several times to the new set, while a particle with a weight close to 0 will probably not be copied at all. The weight of the new particles are set to $\frac{1}{N_\mathrm{p}}$, with N_p being the number of particles. The resampling algorithm thus distributes the new set of particles according to the weight of the old set, without changing the pdf that are reflected by the particles. Resampling can either be carried out after each iteration of the SIS algorithm, or only when the variance of the importance weights have passed a threshold.

The combined SIS/resampling algorithm constitutes a complete particle filter, as described e.g. as the CONDENSATION algorithm in [38]. A very common variant of the particle filter is the *Sampling Importance Resampling* (SIR) algorithm. Here resampling is performed in every step,

and the proposal density π is chosen to the transitional prior as in Equation (8.13). This is illustrated in Figure 8.3 illustrates.

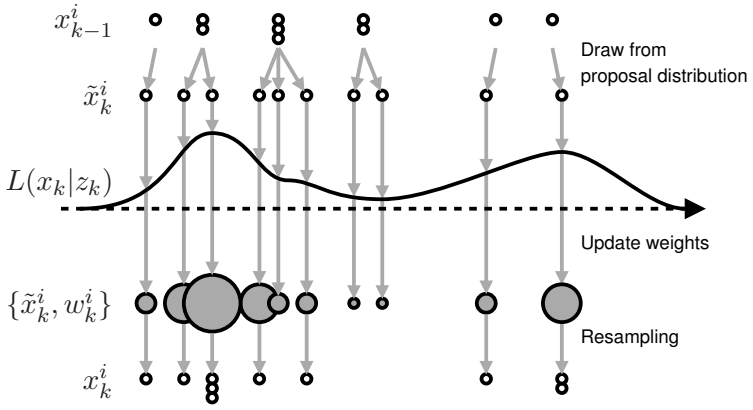


Figure 8.3: *Single cycle of a one-dimensional SIR particle filter. The SIS algorithm from Figure 8.2 is followed by resampling to avoid the degeneracy problem. In other types of particle filters, the weight might depend on the more general expression given in Equation (b) in Figure 8.2 instead of only on the likelihood. Also, resampling is not always performed in each cycle. If not, the states x^i_k will not be grouped as shown here but have different weights instead.*

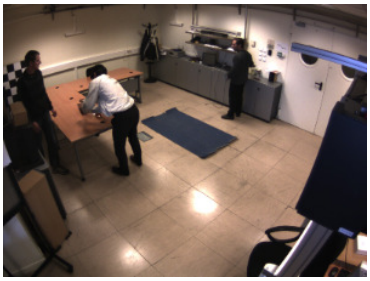
Chapter 9

Likelihood Functions

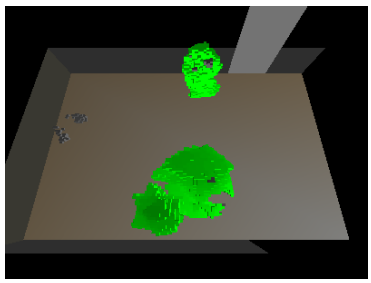
In any state estimation it is of major importance how to determine from measurement data how probable a hypothesised state is. In particle filters, this is done by the likelihood function $L(\mathbf{x}|\mathbf{z})$, which measures how likely a state $\mathbf{x}$ is given measurement data $\mathbf{z}$. As described in Section 8.1, the likelihood is not a probability density function, but is instead only required to integrate over $\mathbf{x}$ up to a normalising constant. In this Chapter, the likelihood functions for the foreground based and feature point based tracking is designed in Section 9.1 and 9.2, respectively. The used parameters are listed in Table 9.1 at the end of the chapther.

9.1 Foreground Likelihood

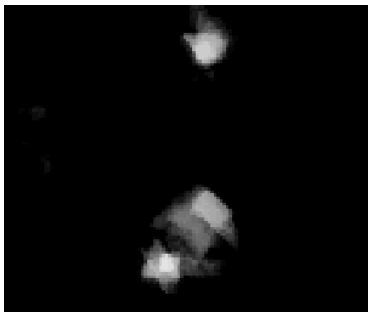
Humans are bounded in their freedom to 2 dimension given by the floor plan. Thus it is sufficient to track in these two dimensions. The 3D representation of the foreground described in Chapter 5 are therefore projected to the floor plan as shown in Figure 9.1. The resulting representation is illustrated in Figure 9.1c. Each pixel corresponds to a column of voxels, and the brightness of a pixel is given by the number of voxels that contain foreground in that column. Therefore, this form can be considered a "2.5D" representation of foreground: The vertical dimension is left out but information about the amount of motion in that direction is kept.



(a) Frame from a corner camera.



(b) Detected 3D foreground.



(c) Foreground projected to the floor plan to 2.5D.

Figure 9.1: *The detected 3D foreground are projected to the floor plan to ease tracking. The projected foreground can be considered a 2.5D representation, since the amount of foreground in the vertical direction is known but not its position.*

9.1.1 States to be Estimated using Foreground

The design of a likelihood function depends on the states to be estimated. In general, this must be kept as low as possible to minimise the required number particles in the particle filter. The minimal possible number of states to estimate with the foreground likelihood function is the two dimensions (x, y) required to represent a position on the floor plan. In addition to this, a size parameter α is also introduced since different persons in different positions will not fill the same area, as is also evident in Figure 9.1c. With this number of parameter, the shape of a state can be chosen either to be a square or a circle. While the circle probably in some situations will be able to describe the measurement data better, it does not fit very well with the discreet representation of the foreground. Therefore, the square shape is chosen. The three parameters to be estimated with foreground are illustrated in Figure 9.2.

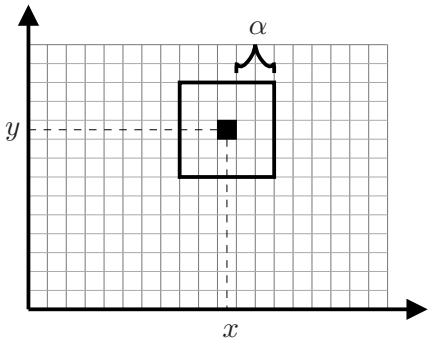


Figure 9.2: *The states that can be estimated using foreground projected to the floor are the position* (x, y) *and the size* α*.*

9.1.2 Foreground Likelihood Function

A number of values can be taken into consideration when designing the foreground likelihood function (in the following, the position (x, y) has been excluded as function parameters for simplicity):

Volume: A person is expected to constitute a certain volume, which can be given as a number of voxels $N(\alpha)$.

Density: A person is expected to fill most of the volume, V, inside his 3D bounding box. This amount is expressed as:

$$F(\alpha) = \frac{N(\alpha)}{V(h, \alpha)} = \frac{N(\alpha)}{h \cdot (2\alpha + 1)^2} \tag{9.1}$$

where α and h are measured in number of voxels. Even though it cannot be determined from the 2.5D foreground representation exactly what lies inside the 3D bounding box of a person, a good approximation can be achieved by assuming that all columns start from the floor. The height h is thus set to the maximum number of voxels in a single column in the area.

Derivative of density: A good state will be centred close to the centre of a person and include most of that person. This means that $F(\alpha)$ is expected to drop fast if α is increased. This

is due to the fact, that most of the area around a person typically is without foreground. The change in $F(\alpha)$ can be measured by its derivative $\frac{\partial F(\alpha)}{\partial \alpha}$, which discreetly is given by:

$$\begin{aligned} F_d(\alpha) &= \frac{\Delta F(\alpha)}{\Delta \alpha} \\ &= \frac{F(\alpha+k) - F(\alpha-k)}{2k} \\ &\approx \frac{1}{2kh}\left(\frac{N(\alpha+k)}{(2(\alpha+k)+1)^2} - \frac{N(\alpha-k)}{(2(\alpha-k)+1)^2}\right) \end{aligned} \tag{9.2}$$

where h is simplified to be the maximum height of the smaller area (which in most cases is identical to that of the larger area).

In Figure 9.3a a 2.5D foreground is shown that comes from a situation where two persons are standing very close together. A good likelihood function must be able to separate these, and the box in the figure indicates a state with $\alpha = 5$ that separates one person well from the other. The rest of Figure 9.3 shows $F(\alpha)$, $F_d(\alpha)$ and $N(\alpha)$ as well as the final likelihood function $L(\alpha)$ when α is varied while x and y are fixed to the centre of the box. From the graphs in Figure 9.3 it seems that $L(\alpha) = -F_d(\alpha)$ would constitute a good likelihood function since it has a clear maximum at $\alpha = 5$. This is not sufficient for the likelihood function, however, since $F_d(\alpha)$ reacts equally strongly on few voxels of noise and a real person. To counter this effect, the likelihood function could be chosen to $L(\alpha) = -F_d(\alpha) \cdot N(\alpha)$. $N(\alpha)$ biases towards larger areas. A problem with this approach is apparent by comparing Figure 9.3a and Figure 9.3c. When α grows to include both persons, $N(\alpha)$ just keeps growing. To avoid including multiple persons, $F(\alpha)$ is also used. When a square state covers multiple persons (corresponding to a cube state in 3D), much of the square will be filled with little or no foreground. This is also evident in Figure 9.3b.

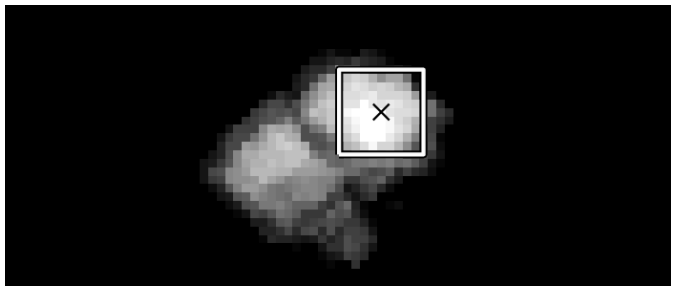
When using a particle filter, the likelihood function needs to be evaluated (at least) once for each particle for each person. Therefore it is essential that this does not take up much computation time. One of the parts that does take some time is to count the number of voxels inside a square, and when evaluating the function $F_d(\alpha)$, the number of particles for two squares must be determined as shown in Equation (9.2); namely $N(\alpha+k)$ and $N(\alpha-k)$. To save computation time, these are also used to evaluate $F(\alpha)$ and $N(\alpha)$, resulting in the final likelihood function:

$$L(\alpha) = -F(\alpha-k)^2 \cdot \sqrt{N(\alpha+k)} \cdot F_d(\alpha) \tag{9.3}$$

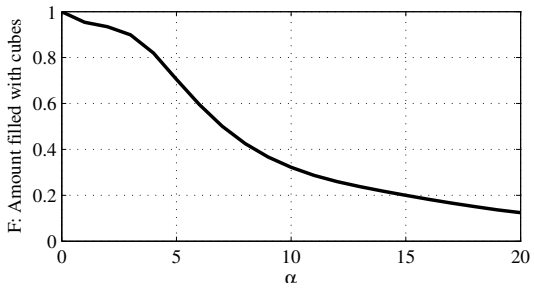
The functions are weighted by squaring $F(\alpha-k)$ and taking the square root of $N(\alpha+k)$ to bias towards single coherent persons. Figure 9.4 illustrates the performance of the likelihood for a particular situation. In Figure 9.4a, the likelihood for all values of x and y with α fixed to 5 is shown. The position (x_i, y_i) that satisfies $(x_i, y_i) = \text{argmax}_{(x,y)}(L(x, y, 5))$ is marked, and α is adjusted at that location to satisfy $\alpha_i = \text{argmax}_{\alpha}(L(x_i, y_i, \alpha))$. The state $\mathbb{S}(x_i, y_i, \alpha_i)$ is shown as a box in Figure 9.4a.

9.2 Feature Point Likelihood

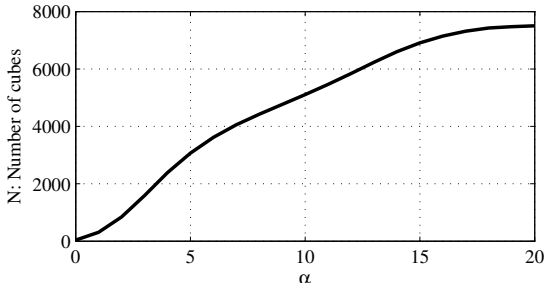
Based on the KLT-tracker [69], feature points are initialised and tracked for each person in each camera as described in Section 6.3. Using these, the feature point likelihood must provide a



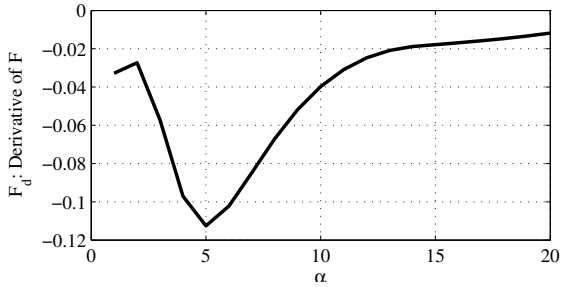
(a) 2.5D representation of foreground with two persons standing very close together. The shown state has been found by manually fixing x and y and then optimising L with respect to α. The maximum is at $\alpha = 5$.



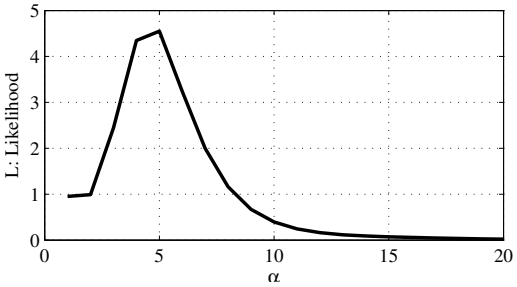
(b) Amount of target filled with voxels, $F(\alpha)$.



(c) Number of voxels within target, $N(\alpha)$.

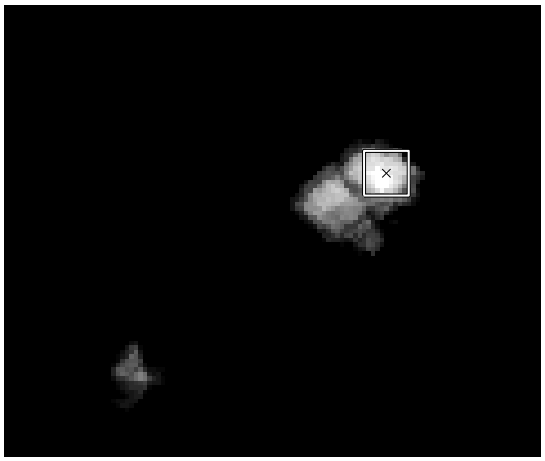


(d) Approximate derivative of $F(\alpha)$, $F_{\mathrm{d}}(\alpha)$.

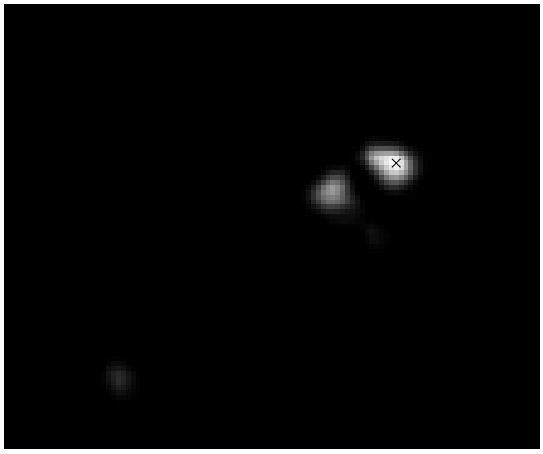


(e) Complete foreground likelihood function, $L(\alpha)$.

Figure 9.3: *Different values as a function of α when the position (x, y) is fixed in the example in (a).*



(a) Figure 9.3a zoomed out to include a third person.



(b) Illustration of the likelihood $L(x, y, \alpha)$ with α fixed to the optimal value from (a).

Figure 9.4: *Illustration of likelihood function.*

measure of how likely it is that a person is located at a hypothesised location on the floor plan, similar what has been described in the previous section for foreground. The foreground is used to estimate both the position (x, y) and the size α. The feature points do, however, not reliably indicate the size of the targets, since they are located randomly on the persons where these have trackable features. Therefore, it is chosen to only estimate the position with the feature points.

Figure 9.5 illustrates how independent feature points located on the same person but seen from different cameras can be used to estimate the position of that person. A feature point in a camera frame corresponds to a line in the 3D space, and how to do the transformation is explained in Appendix A.3.2. If points from different cameras are located on the same person, their lines will pass each other closely near that person. If all lines are first projected to the floor plan they will intersect near their person instead, and this can be used to track a person moving on the floor plan.

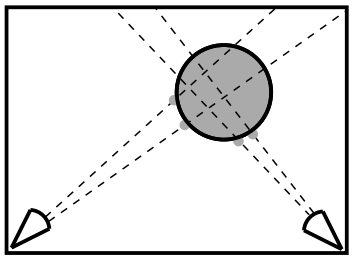


Figure 9.5: *Feature points from different cameras are used to determine 3D lines, which are then projected to the floor plan. The large grey circle is the actual person seen from above.*

After the 3D lines have been projected to the floor, each of them only indicates the position of the target in one dimension; namely the angle seen from the camera. Some additional information can be represented by first analysing the position of the (natively infinite) 3D lines: They all originate from cameras placed near the ceiling and all of them point downwards. Thus, they will cross the ceiling, the floor and borders of the scene under surveillance at some point. Since persons cannot be located outside these limits, the lines can be cut when they reach either of these. Usually it can be assumed that all persons are located within a certain distance from the floor, and the lines can also be cut above this threshold. Figure 9.7 shows an example of lines tracked on three persons from the development data set, and the algorithm is given in Figure 9.6 as pseudo-code for clarity.

9.2.1 Feature Points Likelihood Function

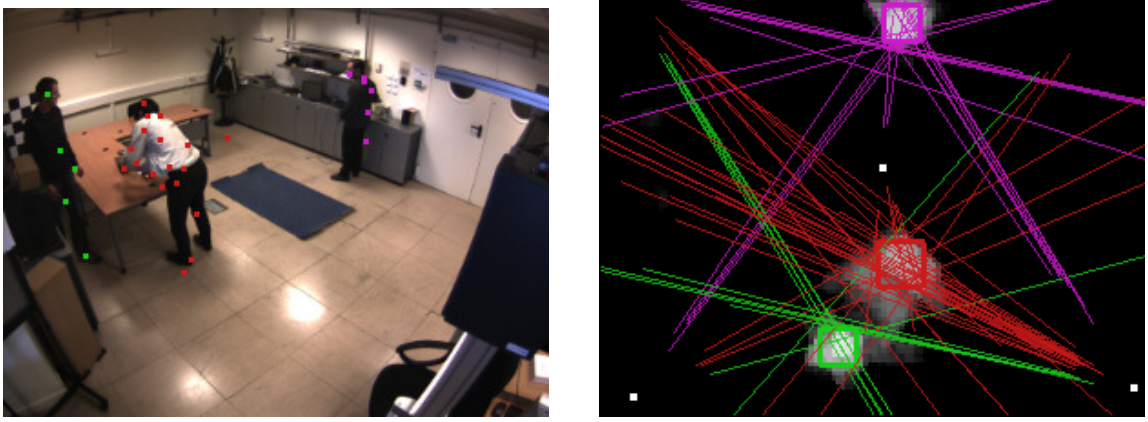
The likelihood function must primarily give higher values, the more lines that pass the hypothesised target. Additionally, since lines from one camera only indicates the position in one dimension, the likelihood function should prefer lines from different cameras. This leads to the following logarithmic likelihood function:

$$L_{\mathrm{FP}}(x, y, \alpha) = \sum_{i=1}^{N_{\mathrm{cams}}} \log_2(L_i + 1) \tag{9.4}$$

where:
N_{cams} is the number of cameras, and
L_i is the number of lines from camera i intersecting the target.

- **For** all targets
  - **For** all feature points in all cameras belonging to that target
    - Determine the infinite 3D line that the feature point represents.
    - Cut the line when it are outside the scene border, below the floor plan or above a predefined threshold, τ_{height}.
    - Project the (finite) lines to the floor plan.
  - The person is most likely to be located where many lines cross.

Figure 9.6: *Pseudo-code for person tracking using feature points. The targets are all known persons in the scene. In the implementation, each the finite, projected line of each pixel in all cameras is calculated offline to minimise the computation time of the algorithm.*



(a) Frame from a corner camera.

(b) 2.5D foreground representation with feature points projected to the floor plan as finite lines.

Figure 9.7: *Example of detected feature points on three persons in (a) and the projection of the points to the floor plan as lines in (b). Each line in (b) corresponds to a feature point and is coloured depending on which target it belongs to. The lines originate from the cameras (indicated as white squares), and they are cut below the floor and above a threshold.*

A one dimensional example of this likelihood function is shown in Figure 9.8, where the bias towards multiple cameras is obvious.

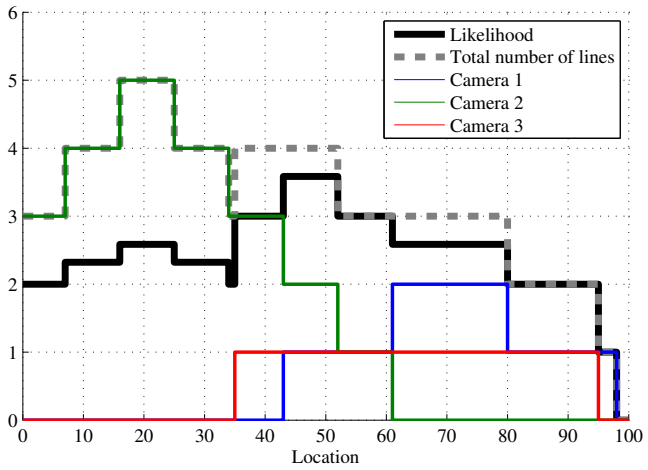


Figure 9.8: *Example of feature point likelihood in one dimension. While the total number of lines is largest around 20 on the x-axis, the likelihood are biased towards preferring lines from more cameras. The likelihood is therefore largest around 50, where all cameras see line(s).*

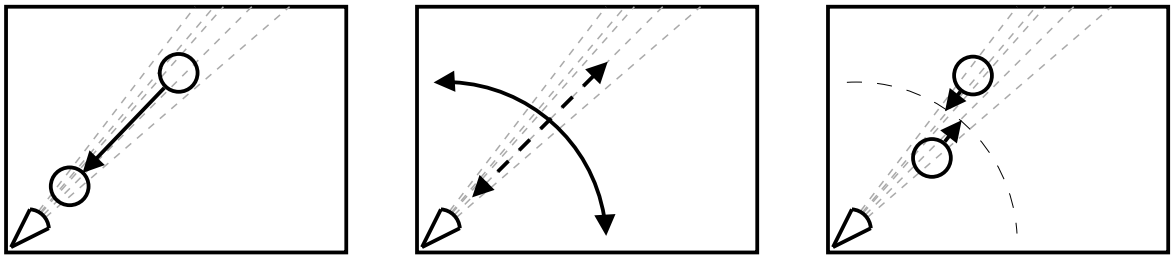
9.2.2 Registration of Lines at Hypothesised Targets

Since the size of the target is not estimated with the feature points, a predefined size is used instead. If the shape of the hypothesised targets are defined to be circular, it is very simple to determine if a line intersects. The distance between the line and the target centre is first calculated and then compare with the radius of the circular target. One problem remains, however, which is illustrated in Figure 9.9a. Since the lines from one camera originates from the same camera, the lines will be located denser closer to that camera. This will cause targets to move towards cameras that see many points on them. A solution is illustrated in Figure 9.9b and 9.9c. Projected lines from a camera do, as previously described, only contain information about the location of the person in one dimension; the angle seen from that camera. Therefore, all targets located at the same angle should receive the same contribution to the likelihood value from that camera. This is done by moving the targets to have the same distance to the camera when evaluating the likelihood, as shown in Figure 9.9c. The distance used is the distance between the target and the camera in the previous frame.

To sum up, the feature points are used to estimate the position of persons in the scene, but not their size. Feature points are converted to 2D lines in the floor plan, and the number of lines intersecting a hypothesised target is determined as illustrated in Figure 9.9. This is used to calculate the likelihood according to Equation (9.4). Table 9.1 shows the parameter values used as default in the system.

Parameter	**Symbol**	**Value**
Step size for approximating $F_{\mathrm{d}}(\alpha)$	k	1 voxel
Max feature point height	τ_{height}	2 m
Radius for feature points target size	r_{FP}	20 cm

Table 9.1: *Parameters used in the likelihood functions. The voxel size was chosen to 5 cm in Section 5.1.*



(a) Target will tend to move towards each camera.

(b) The lines from each camera can only indicate the targets position in one dimension, not the distance from the camera.

(c) To avoid affecting the dimension in the direction of the camera, all targets are moved to have equal distance to the camera when evaluating the likelihoods.

Figure 9.9: *For each camera, the lines will be denser closer to the camera. To avoid that the targets tend to move toward the cameras, they are for each camera moved to have the same distance to the camera when evaluating the likelihood.*

Chapter 10

Particle Filter

In this chapter the main part of the tracking system is designed: The state estimator. As described in Section 8.3, particle filters have been chosen as the basis for state estimation due to their ability to track arbitrary distributions. To be able to test the hypothesis from the introduction, Chapter 3 claiming that a combination of foreground and feature points is better than a tracker using one of these modalities, both a combined particle filter and two individual particle filters will be designed in this chapter. These three particle filters can each be put into the tracking framework described in Chapter 7. This chapter is organised as follower: Section 10.1 introduces the common parts for the particle filters, followed by Section 10.2, 10.3 and 10.4 where the three particle filters are designed. The constants used through out the chapter can be fined in Table 10.1 on page 82 at the back of this chapter.

10.1 Introduction

The particle filters will be designed based on the standard SIS filter followed by resampling as described in Section 8.3. The actual state estimation is then done based on the particle states. The original CONDENSATION algorithm by Blake et. al. [38] was designed to track a single object. It was indicated that the approach should be able to track multiple targets simultaneously. The particle filter is a multiple hypothesis tracker and thus has multi-object tracking potential. The particles should ideally distributed according the posterior probability density function (pdf) and have particles located at all positions where targets might be present. This is the why the authors of the original CONDENSATION algorithm claimed that it should natively be able to track multiple persons. However the multi-target tracking does not come for free, the more the particles are spread over a large area the more particles are needed to have a dense representation of the state space. Most particle filters are implemented to track a single object [18]. This is also the approach that is going to be used here. The design is to a great extent inspired by Blake et. al. [64], and will be based on a single particle filter per target.

In Figure 10.1 the state estimation framework that is used in this chapter is shown. The four blocks in Figure 10.1 and the state space are discussed here:

Dimensionality of the state space: The tracker is, as described in the problem formulation, Chapter 3, meant to track people when they are located in a room monitored by multiple

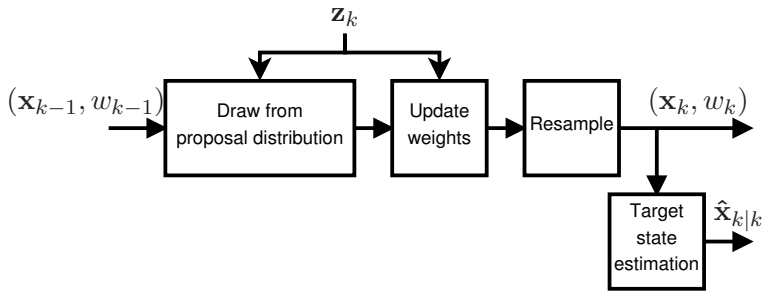


Figure 10.1: *General state estimation framework based on a particle filter. The vectors $\mathbf{x}_k$ and $\mathbf{x}_{k-1}$ includes the states of all particles, and $\hat{\mathbf{x}}_{k|k}$ is the estimated target state at time k given all measurements $z_{1:k}$.*

cameras. Thus the target position on the floor-plane is needed in the state vector $\mathbf{x}_k$ for the particle filter. Other dimensions might also be useful for the given particle filter but not needed in the final output. These will be discussed in the section of the respective particle filter.

The proposal distribution $\pi(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$: This is of mayor design importance in the particle filter design. As described in Section 8.3 the optimal proposal distribution includes the state evolution model, $p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)$, and the measurement data $\mathbf{z}_k$. However, this is in practice often impossible to sample and often just the evolution model is used (cf. Section 8.3). The proposal distributions is designed in the section of the respective particle filter.

Weights: The update equation is given as (cf. Section 8.3):

$$w_k^i \propto w_{k-1}^i \frac{L(\mathbf{x}_k^i|\mathbf{z}_k)p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)}{\pi(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k)} \tag{10.1}$$

With $\sum_{i=1}^{N_p} w_k^i = 1$. The likelihood functions $L_{\text{FG}}(\mathbf{x}_k^i|\mathbf{z}_k)$ and $L_{\text{FP}}(\mathbf{x}_k^i|\mathbf{z}_k)$ designed in Chapter 9 will be used in this chapter.

Resampling: As discussed in Section 8.3 resampling is needed because otherwise the variance of the importance weights will increase over time. This will eventually result in one of the normalised weights be close to 1 and the remaining close to 0. As the resampling principles does not depend on the modality it will be discussed in Section 10.1.2.

Target state estimation: The general Minimum Mean-Square Error MMSE estimate (cf. Equation (8.1) on page 61) was used for state estimation by Blake et. al. [38] in the original CONDENSATION algorithm. This is also the algorithm that is going to be used here. That is the target state will be estimated using the weighted average of the particle states:

$$\hat{\mathbf{x}}_{k|k} = \sum_{i=1}^{N_p} \mathbf{x}_k^i \cdot w_k^i \tag{10.2}$$

where:
N_p is the number of particles in the particle filter,
$\mathbf{x}_k^i$ is the i'th particle state at time k, and
w_k^i is the weight of the i'th particle state at time k

Before designing the three particle filters, the state evolution model will be investigated in Section 10.1.1, as the particle filters are going to be used to track the same persons, and thus has this in common. In Section 10.1.2 resampling will be discussed before moving on the the design of the unique parts of three particle filters.

10.1.1 State Evolution Model

Most objects are moving in fairly predictable patterns and that is why it is good practice in general to design a state evolution model [64]. However, in our scenario there are no obvious paths that the people are following and abrupt movement can be present. For this reason a very general model for the evolution of the state is selected: A Gaussian random walk model. To deal with situations where the track is lost and help the tracker to recover after possible lost tracks, a uniform component is also included as in [64]. The state evolution model is then given by:

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) = (1-\beta_u)\mathcal{N}(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{\Sigma}) + \beta_u \mathcal{U}_{\text{Evul}}(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{10.3}$$

where:
$\mathbf{x}_k$ is the particle states at time k,
$\mathcal{N}(.|\boldsymbol{\mu}, \mathbf{\Lambda})$ is the Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{\Lambda}$,
$\mathcal{U}_{\text{Evul}}(\mathbf{x}_k|\mathbf{x}_{k-1})$ is a uniform distribution, and
$\mathbf{\Sigma}$ is the diagonal matrix with the variances of the random walk models on the components in the state vector.

If the uniform component is spread over the entire state space as in [64] it will have a very little value and the importance will change depending on the size of the room. A uniform component spanning the entire room would also result in particles from all targets would tend to seek to the one location in the room with the largest likelihood and not stay at the target they were meant to track. Therefore it is decided to only distribute it over a set around the particle state. The size of $\mathcal{U}_{\text{Evul}}(\mathbf{x}_k|\mathbf{x}_{k-1})$ the in practice controls how far a particle are allowed to jump in one step.

10.1.2 Resampling

Two things are to be decided concerning resampling: When to resample and which particles to copy:

When: Compared to the remaining algorithms in the system the resampling takes up minimal computational power and resampling in every iteration is done for simplicity as in the standard SIR approach (cf. Chapter 8.3).

How: It is important to ensure that a good hypothesis can survive a short lack of measurement evidence. A "tunneling" effect is therefore needed to avoid killing all particles without measurement data in one iteration [38]. In the resampling the probability of a particle to survive depends on the weight of the particle. This weight is proportional to the likelihood function, and will thus for the likelihood functions in Chapter 9 be zero if no measurements support this particle. Thus it is chosen to also use a uniform component, and let $\beta_{\text{resample}} \cdot N_p$ particles be randomly selected.

After the presentation of the general particle framework and the common components for the three particle filters the three remaining sections of this chapter present the two individual and the combined particle filter.

10.2 Foreground based Particle Filter

In this section the foreground based particle filter will be designed based on the approach in Figure 10.1 and the foreground likelihood function developed in Chapter 9. From the chapter introduction three points where listed to be designed for each particle filter:

Dimensionality of the state space: As mentioned the output of the tracker is a 2D position of the person on the floor-plane. In additional the foreground likelihood function introduced α that describes the target size. The state vector will thus be: $\mathbf{x} = [x, y, \alpha]^\mathsf{T}$.

Proposal distribution: This is designed in Section 10.2.1.

Weights: Updating of the particle weights is done using Equation (10.1) inserting the state evolution model developed in Section 10.1.1, the foreground likelihood function from Section 9.1 and the foreground proposal distribution.

10.2.1 Foreground Proposal Distribution

As described in Section 8.3, the optimal proposal distribution is based on the state evolution model and the measurement data. However, in practice it is often not possible to include all measurement data and simplifications needs to be made. The standard SIR approach handles this by assuming that the proposal distribution is distributed like the state evolution model (cf. Section 8.3). However, in some cases a better approximation of the optimal proposal distribution (cf. Equation (8.12) on page 64) can be advantageous. This is the case if the track get locked or is lost for a short period, e.g. the track can be locked on a shadow or the person can pass by a spot where he cannot be separated from the background. Also sometimes a better proposal distribution can guide the particles better and fewer particles are required for tracking.

Such an approximation of the optimal proposal distribution is e.g. used by Blake et. al. [64] for their motion modality based on absolute frame subtraction. They use a combination of the evolution model and measurement data. The latter is introduced by evaluating the likelihood function on a regular grid spanned on a subset of the state space. The proposal distribution is then constructed as a sum of the state evolution model and motion likelihood on some grid points symmetrically spread in the room at a resolution depending on the available computational resources. In the beginning of each iteration the likelihood is evaluated on these grid points, and all grid points with a likelihood value larger then a predefined threshold are selected. A Gaussian distribution is then centred on each of the points $\mathbf{p}_{\text{high}}$ and added to the proposal distribution. The equation is:

$$\pi(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) = \beta_{\text{RW}} \cdot \mathcal{N}(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{\Sigma}) + \frac{1-\beta_{\text{RW}}}{N_{\text{high}}} \sum_{i=1}^{N_{high}} \mathcal{N}(\mathbf{x}_k|\mathbf{p}_{\text{high},i}, \mathbf{\Sigma}) \tag{10.4}$$

where:
β_{RW} is a constant determining the ratio between the evolution model (a random walk) and the grid points, and
N_{high} is the number of grid points that have a likelihood that exceeds the predefined threshold.

This way of constructing the proposal distribution has the advantage that the tracker can always recover as the proposal distribution can guide the particles to places with motion. This comes at the cost of evaluating the grid points, but the requisite number of particles needed will also be reduced as each particle can search most of the state space. In our case, however, the particles cannot jump to an arbitrary location in the state. In order not to make all targets jump to the person with the largest likelihood value it is chosen to only allow the particles to jump to an area around the current location (cf. Section 10.1.1). Thus a grid inside this area will be used. With this limitation a proposal distribution that supports jumps inside the area can be made. To decrease the computational complexity and focus on the most important dimensions, the grid will only be spanned according to the position, leaving the α unchanged. As the grid points are located around the target position, all grid points are valid new state positions and there are no need to threshold to get rid of weak grid points. Thus it is decided to give each grid point positions $\mathbf{p}_i$ a probability depending on the likelihood value at the grid point:

$$\pi_{\text{FG}}(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{z}_k) = \beta_{\text{RW}} \cdot \mathcal{N}(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{\Sigma}) + \frac{1-\beta_{\text{RW}}}{\sum_{i=1}^{N_{\text{grid}}} L_{\text{FG}}(\mathbf{p}_i,\alpha_{k-1})} \sum_{i=1}^{N_{\text{grid}}} L_{\text{FG}}(\mathbf{p}_i,\alpha_{k-1})\mathcal{N}(x_k,y_k|\mathbf{p}_i,(\sigma_x,\sigma_y)) \quad (10.5)$$

where:
N_{grid} is the total number of grid points and
$L_{\text{FG}}(.)$ is the foreground likelihood function defined in Section 9.1.

The number of grid points N_{grid} is chosen to be odd in order to have a grid point on the location of the old particle state. In Figure 10.2 a one dimensional example of the proposal distribution is shown with three grid points.

10.3 Feature Point based Particle Filter

As for the foreground based particle filter the feature point likelihood function is designed in Chapter 9. From the chapter introduction Section 10.1 three points where listed to be decided up on in the particle filter design:

Dimensionality of the state space: As previously mentioned, the output of the tracker is a 2D position of the person on the floor-plane. No additional dimensions was introduced in the feature point likelihood function, thus the state vector is: $\mathbf{x} = [x, y]^{\mathsf{T}}$.

Proposal distribution: This is designed in Section 10.3.1.

Weights: Updating of the particle weights is done using Equation (10.1) using the state evolution model developed in Section 10.1.1, the feature point likelihood function designed in Chapter 9 and the feature point proposal distribution.

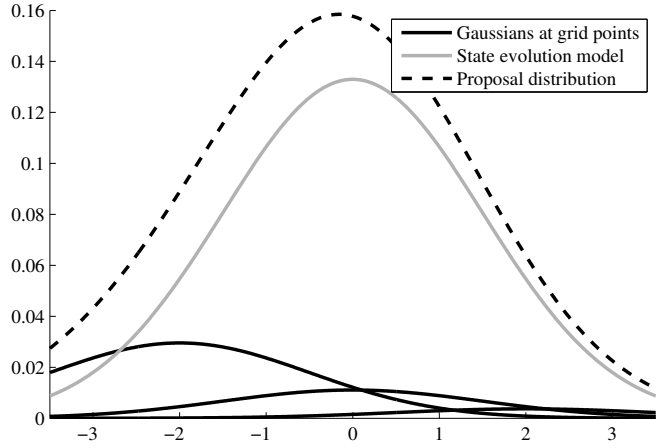


Figure 10.2: *A one dimentional example of the foreground proposal distribution with three grid points. The likelihood value at the grid point located at -2,0,2 are from left: 8, 3 and 1. The Proposal distribution is a sum of the three Gaussian located at grid points and the state evolution model.*

10.3.1 Proposal distribution

For the foreground particle filter proposal distribution the standard evolution model based approach was supported by measurement data. This was done mainly to guide the particles better after possible lost tracks. For the feature points, however, new points cannot appear on the person if the track is lost, as new feature points are initialised inside the target box (cf. Section 6.3). Thus an advanced proposal distribution will have very little impact on this matter and the standard SIR approach described in Chapter 8 is chosen.

10.4 Combined Particle Filter

In this section the combined state estimator is designed by fusing the foreground and feature point modalities. Two ways of combining different modalities are layered sampling and combined likelihood [8, 64]. The idea of combined likelihood is to compose a joined likelihood function and use this as the foundation of a combined particle filter, e.g.:

$$L_{\text{comb}}(\mathbf{x}) = f\big(L_{\text{FG}}(\mathbf{x}), L_{\text{FP}}(\mathbf{x})\big) = c \cdot L_{\text{FG}}(\mathbf{x}) + (1-c) \cdot L_{\text{FP}}(\mathbf{x}) \tag{10.6}$$

where c is a weighting constant.

The idea of layered sampling is to cascade two particle filters with resampling in between. If the state space can be divided into independent subspaces so that each dimension in the original space is searched in only one subspace, this is also known as partitioned sampling [57, 64]. In partitioned sampling each subspace can be searched independent of the others. Since the dimensionality of all subspaces are lower that that of the original space, the number of particles can be reduced significantly while providing the same tracking performance.

For our system, however, it is not possible to make two independent subspaces because both modalities describe the position. However, only the foreground modality uses the size α which does allow layered sampling to provide some improvement. In this case with correlated subspaces the sequence of the two modalities is important for the precision of the tracker. The first modality

can be used to guide the particles in the right direction, as areas with high likelihood from the first modality will have a more particles after resampling than areas with a low likelihood. This means that the main effort on the second modality can be used to increase the precision in the area where the person is most likely to be located. The precision of the foreground modality is higher than the precision of the feature point modality, as foreground is most likely to be present at the position of person while feature points are present wherever they are initialised inside the 2D foreground mask (cf. Section 6.3). This combined with the higher dimensionality of the foreground likelihood means that the foreground modality must come last in a layered sampling approach.

So even though the modalities both describes the position, layered sampling does have some advantages compared to a joined likelihood. Here, the following the two candidates are compared:

- Layered sampling:
  - Reduces the needed number of particles as the first layer (of reduced dimensionality) can guide the second.
  - Updating according to the most precise modality last ensures the same precision as the most precise modality in the situations where this is present.
- Joined likelihood:
  - A smart combination can ensure that the most reliable likelihood function is weighted most, e.g. use foreground when targets are perfectly separable and feature points otherwise.
  - The likelihood value of the modalities must have the same order of magnitude, e.g. by using normalised likelihood values.

A good combined likelihood function depend on the ability to combine inputs from the different modalities. Equally good measurements from the modalities must give a comparable likelihood value. It is often difficult to design good normalised likelihood functions, however, and as documented in Chapter 9 this has not been attempted for our system. Thus a good combination of the modalities might be tricky. Although a combined likelihood function has a good potential it might require much work to make the behaviour similar in different video recordings as no good way of normalisation the likelihood functions were discovered in Chapter 9. Because of the problems with the combined likelihood together with the effectiveness of the layered sampling approach, the latter is chosen. This means that the combined particle filter will be a cascade of the two individual particle filters as shown in Figure 10.3. After each of the two elements in the cascade, resampling is performed as in [64] in order to utilise the first modality to guide the next.

All parameters used in the particle filter are shown in Table 9.1 along with their default setting.

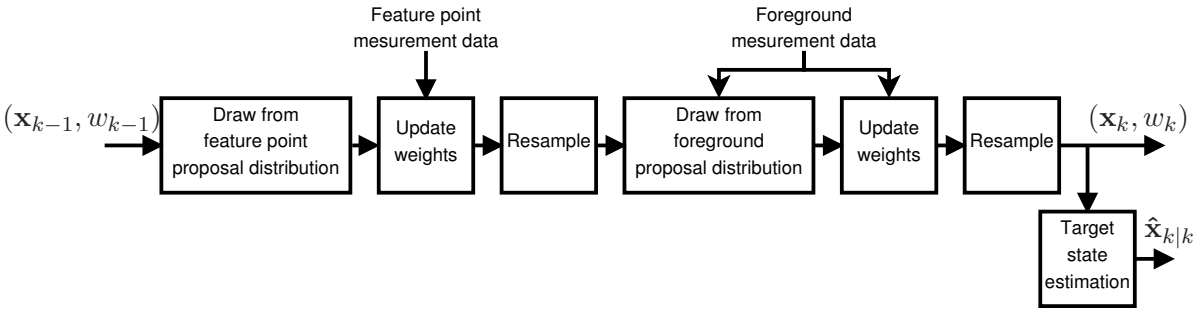


Figure 10.3: *Flowchart of the combined tracker designed using layered sampling.*

Parameter	**Symbol**	**Value**
Variance of the dynamics	$(\sigma_x, \sigma_y, \sigma_\alpha)$	$(1.5, 1.5, 0.3)$
Grid step size	N_{gridStep}	2
Number of grid points	$N_{\text{grid}} = N_{\text{gridWidth}} \times N_{\text{gridWidth}}$	3×3
Area of the uniform part of the state evolution model	-	7×7
Number of particles	N_p	50
Ratio of the particles to choose randomly when resampling	β_{resample}	0.3
Ratio of the influence of the grid in $\pi_{\text{FG}}(\mathbf{x}_k \vert \mathbf{x}_{k-1}, \mathbf{z}_k)$	β_u	0.5

Table 10.1: *Parameters used by the particle filters. All distances are relative to the voxel size which has been chosen to 5 cm, cf. Section 5.1.*

Part IV

Implementation

Contents

In the previous two parts, algorithms for tracking persons based on foreground and feature points was designed. An implementation of these algorithms in C++ is designed in Chapter 11. In Chapter 12 a distributed version of the implementation is designed to make live and real-time execution possible.

Chapter 11

Software Design

This chapter documents the design of the C++ software framework that has been developed for the implementation of the tracking algorithms designed in the previous parts of the report. The framework is designed to run in real-time and to support the documentation of the project.

As the algorithms needed for tracking and the large amount of video data that is used by the system can be very heavy to handle in real-time by a standard single core CPU, the framework is designed with parallelism in mind. Apart from running in real-time, additional requirements exist. Firstly, the framework must be configurable in order to ease the development during the project, i.e. it must be possible to select video input, and modify parameters of algorithms etc. without recompilation. Secondly, the framework must support displaying intermediate and final results for documentation and debugging purposes i.e. a 3D view of the estimated foreground.

The algorithms must also be able to run sequentially to make it possible to measure the time consumption of the individual algorithms. The basic structure and central elements of the program are described in Section 11.1. In Section 11.2 the additional threads needed for a parallel execution of the algorithms are introduced, and in Chapter 12 a network distributed version of the system is designed.

11.1 Central Classes

The tracking algorithms can be divided into two groups: The algorithms that is to be executed per camera which can be run in parallel, and the algorithms that combine the information from the cameras. Additionally, the framework must be able to display both intermediate and final results of the tracking algorithms. These three tasks are handled by the three classes *PerCam*, *Sequential*, and *Renderer* respectively, which are illustrated in Figure 11.1. In order to synchronise the algorithms in PerCam and Sequential, they are called from the Application class. Renderer is running in parallel with the tracking algorithms by having Renderer execute in a separate thread while Application is executed in the main thread. The results in smooth GUI rendering and window resizing without delaying the tracking algorithms.

The implementation of the "active" class Renderer and later on the parallel executing of PerCam, an abstract class Runnable is made. The Runnable class has a pure virtual run function, which is executed in a new separate thread when an instance of a derived class is created. The inheritance

from Runnable is shown in the class diagram in Figure 11.1.

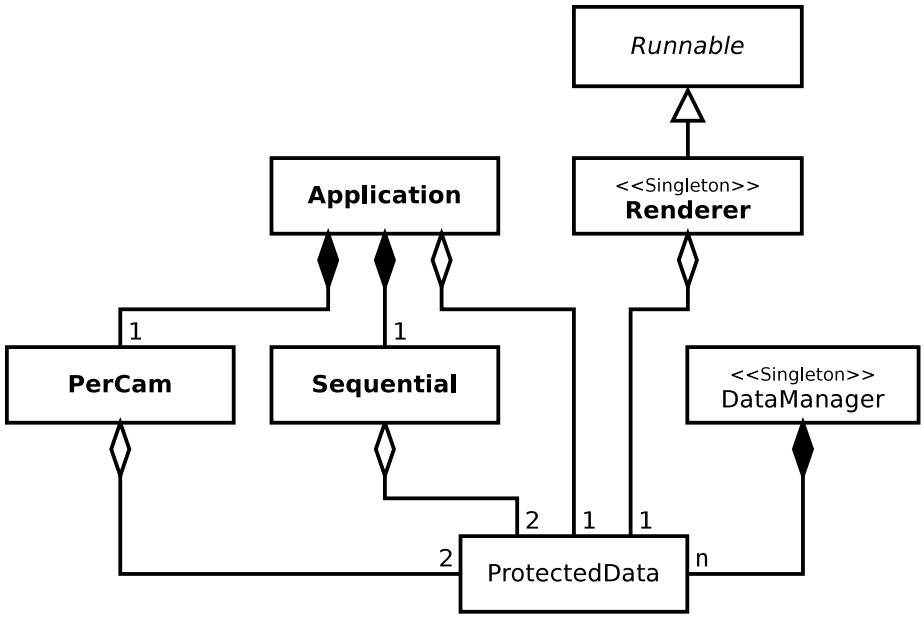


Figure 11.1: *Class diagram showing the four main classes Application, PerCam, Sequential, and Renderer (marked in bold) along with the abstract class Runnable and Data/Manager/ProtectedData which implement buffering of the intermediate tracking results.*

As can be seen in Figure 11.1, two classes are introduced in addition to the previously mentioned: *DataManager* and *ProtectedData*. They are necessary to give Renderer access to the intermediate results from the tracking algorithms. ProtectedData is introduced to hold all data to be shared and thus contains intermediate results from the tracking along with the estimated target positions. Some of the results, which are calculated by PerCam and Sequential, depend on the output from last frame. This means that two ProtectedData instance is required, which can be switched between all frames. Renderer also need access to an instance of ProtectedData, however, and to avoid delaying the tracking algorithms when they need to switch frames, 3 instances of ProtectedData are created. The one used by Renderer is called the *front buffer*, the one used to store results by the algorithms is called the *back buffer*, and the last one is an unused, free buffer. The algorithms that require results from the old frame can access these in the front buffer. The buffering of the ProtectedData is implemented by the class DataManager, and whenever tracking has completed for a particular frame and all results are stored in the back buffer, Application requests DataManager to swap the buffers. If Renderer is not actively using the front buffer, the front and back buffers are switched. Otherwise, the back buffer is switched with the free buffer.

Apart from the classes in Figure 11.1, two utility components are worth mentioning. These are the class Configuration and the function onKey. The onKey function receives keyboard commands, e.g. to pause the video and rotate the 3D view of the room. The four main classes from Figure 11.1 together with Configuration are described in the following:

Configuration: The Configuration class keeps track of all configuration settings, and thus serves to make the program easy to reconfigure. It is a singleton, so the settings can be retrieved (and some settings changed) from all other parts of the program. When the program is launched, one or more configuration file(s) can be specified and loaded by this

class. Although the program can execute without configuration files, they enable the user to override most of the default hard-coded configuration settings. Three main categories of parameters that can be adjusted in the configuration file:

- **Video source:** A path to a web-cam, a video file or to a folder of images.
- **Algorithm parameters:** Most parameters of the tracking algorithms can be specified to tune performance.
- **Intermediate Displays:** It can be specified if each of the intermediate results are to displayed.

Application: This class is responsible for executing the tracking algorithms. After the class is created it runs the main thread. The thread runs in a loop which first calls PerCam followed by a call to Sequential to run the tracking algorithms and ends by requesting DataManager to swap the ProtectedData buffers.

PerCam: This class contains all the algorithms that are to be run per camera, as illustrated in Figure 11.2. In the beginning of each iteration PerCam is called by Application to execute the algorithms. All results are stored in the back buffer.

The algorithms must be able to run both sequentially to make time measurements of the different algorithms possible and also in parallel to decrease the execution time. Therefore the algorithms in PerCam is designed so that they can run for all the cameras or for a specific camera.

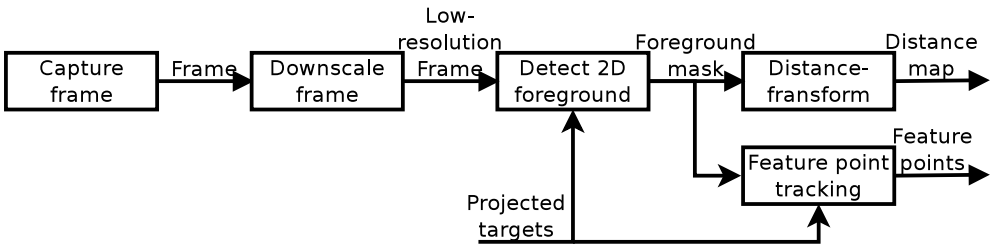


Figure 11.2: *Overview of the algorithms implemented in the class PerCam.*

Sequential: This class contains all the algorithms that cannot be parrallelised for each camera, as illustrated in Figure 11.3. In each iteration Application executes these algorithms and, as for the algorithms in PerCam, all results are stored in the back buffer.

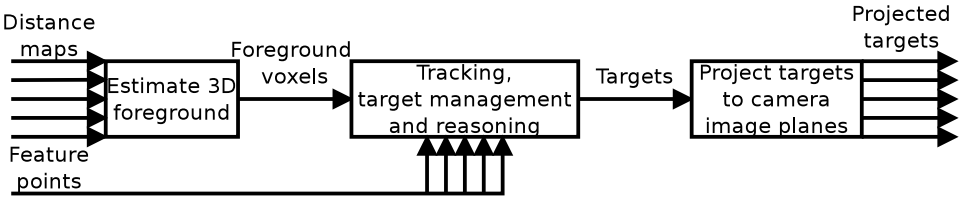


Figure 11.3: *Overview of the algorithms implemented in the class Sequential.*

Renderer: The purpose of this class is to render everything of interest on the display. This can include the videos, the 3D view, the reasoning results, and any debugging displays that have been enabled. It has a member function onDisplay which performs all rendering of a single frame, and this function is called whenever no other events are pending in the thread, which on a sufficiently fast computer means it runs with the frame rate of the

graphics card. If no new results are to be displayed and the windows have not been resized Renderer will just sleep a few milliseconds.

11.2 Parallel Implementation

The software framework for sequential execution of the algorithms was designed to allow easy parallelisation of the algorithms that are to be performed per camera. More parallelisation are indeed possible, e.g. the first algorithms in Sequential that combines 2D foreground into 3D is independent of the feature point tracking. Also feature point tracking is independent of the foreground estimator, except initialisation of new feature points. However, 97% of the time is spend on the algorithms in PerCam (cf. time test carried out in Chapter 13) and it is therefore chosen to parallelise only these algorithms for simplicity. The algorithms in PerCam can easily be split into five parallel tasks which is more than enough to utilise a dual core CPU. In this section the additional active class PerCamRunner is introduced. This class utilises the abstract class Runnable as shown in Figure 11.4.

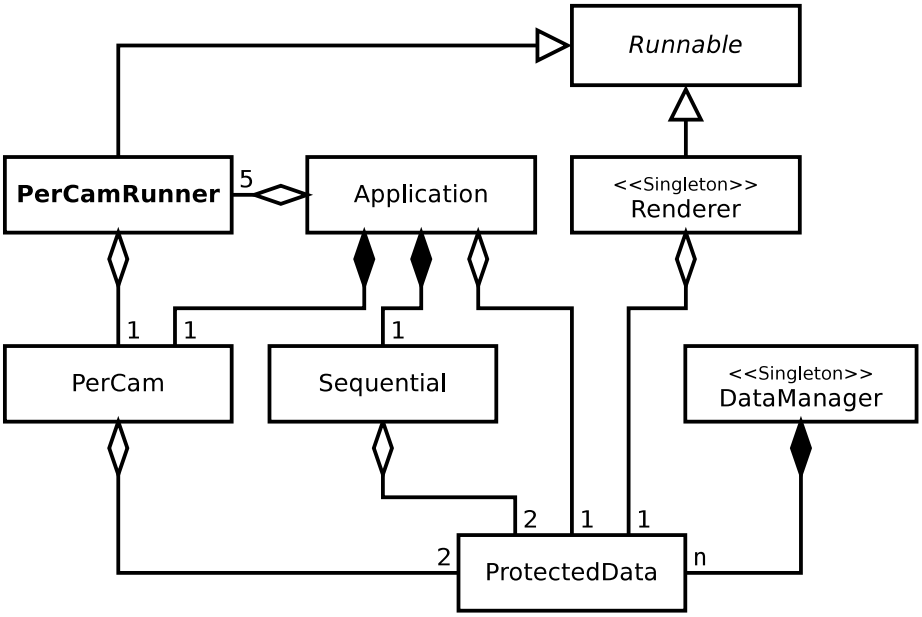


Figure 11.4: *The class diagram from Figure 11.1 extended with the active class PerCamRunner (marked in bold).*

When the algorithms in the PerCam class is called, it is specified which cameras the algorithms are to be executed on, as described in Section 11.1. Thus, Application initialises an instance of PerCamRunner with a camera number specified. The internal thread in PerCamRunner will wait for signal to start executing the algorithms in PerCam on the current frame, and will signal to Application when the algorithms has finished. Therefore five instances of the PerCamRunner class can use the same PerCam class to run the algorithms in parallel.

The parallelisation can be enabled/disabled in the configuration file. The source code of the tracker prototype is found on the attached CD-ROM together with build instructions and dependencies are listed in Appendix C.

Chapter 12

Network Distribution

The tracking system is intended to be able to run in real-time, cf. Chapter 3) which is not possible on a single computer caused by the computational complexity of the algorithms. Instant we have access to six computers at AIT [1] where five of them are connected to a USB-camera as discussed in Section 1.1.1. In this chapter a distributed version of the program from Chapter 11 is designed to run live at this setup.

12.1 Network Topologies and Protocol

A lot of different network distribution topologies exists. In our system, the information extracted from each camera needs to be combined in each iteration on one computer. For instance, the 2D foreground from all cameras must be combined when estimating a complete 3D foreground. Thus we need a central computer to control the flow. For this, two topologies exist: "Master-Slave" and "Divide and Conquer". The latter is preferred if the tasks that are to be carried out can be divide into subtasks that again can be split. In this case, however, the obvious way to go is to have the algorithms performed at the computer that are connected to the camera if possible. This will eliminate the need for sending the colour images between different computers. Thus the "Master-Slave" approach is chosen, where the master is functioning as server and the slaves are functioning as clients.

The computers at AIT are connected with Ethernet through a 1 GBit switch. There are two commonly used protocols for this kind of connection, namely *User Datagram Protocol* (UDP) and *Transmission Control Protocol* (TCP). For our system, data loss is not acceptable and the overhead implemented in TCP to ensure that all data arrives exactly as intended is preferred over the faster but unreliable UDP. This also means that the communication in the distributed system will be connection oriented.

12.2 Network Interface

When distributing the system between multiple computers, an important design decision is which parts of the computation to perform on the slaves and which parts to perform on the master.

For our system, the algorithms in the class PerCam (cf. Chapter 11) can easily be distributed since they are already designed to be able to run in parallel. This is therefore chosen.

Depending on the amount of data to be sent from the slaves to the master, some kind of data compression might be advantageous. The input to the PerCam class is the projected targets and the output is a floating point distance map and some feature points, as indicated in Figure 11.2 on page 87. This means that distance maps and some feature points must be send to the master while the projected targets must be send to the slaves. The total time of sending the raw data from a slave to the master is:

$$t_{\text{slave-master}} = \frac{s_{\text{distMap}} \cdot n_{\text{pixels}} + s_{\text{FP}} \cdot n_{\text{FP}}}{BW} \tag{12.1}$$

where:
s_{distMap} is the size of the data type used to store the distance map,
n_{pixels} is the image size of the distance map, and
n_{FP} is the number of feature point,
s_{FP} is the memory size of one feature point,
BW is the bandwidth of the Ethernet connection.

The distance maps are of the same resolution as the video frames. For the development data set, the image resolution is 1600×1200 for the four corner cameras and 1024×768 for the top camera (cf. Section 1.1) and these are scaled down by a factor of 4 as described in Chapter 4. On the used platform the floating point numbers used for the distance map are stored in 4 bytes (s_{distMap}=4) and the feature points are stored in two integers of 4 bytes each ($s_{\text{FP}} = 2 \cdot 4 = 8$). The number of feature points varies from frame to frame, an a worst case value cannot generally be chosen. Here, 30 feature points per per camera per person and 10 persons is assumed. The network speed is 1 GBit/s = 1/8 GByte/s, if discarding network delay and network overload the time of sending the data from a slave with the two different resolutions are:

When using are a network connection of 1 GBit/s = 1/8 GByte/s the theoretically time of sending the data from a slave with the two different resolutions are:

$$t_{\text{corner-master}} = \frac{4 \cdot \frac{1600}{4} \cdot \frac{1200}{4} + 8 \cdot 10 \cdot 30}{1/8 \cdot 10^9} = 3.84 \text{ ms} + 19.2 \ \mu\text{s} = 3.86 \text{ ms} \tag{12.2}$$

$$t_{\text{top-master}} = \frac{4 \cdot \frac{1024}{4} \cdot \frac{768}{4} + 8 \cdot 10 \cdot 30}{1/8 \cdot 10^9} = 1.57 \text{ ms} + 19.2 \ \mu\text{s} = 1.59 \text{ ms} \tag{12.3}$$

This times must be expected to be a bit higher i practice as, network delay and network overload.

If all slaves are sending the data at the same time, the total time of receiving the data will be:

$$\begin{aligned} t_{\text{slaves-master}} &= 4 \cdot t_{\text{corner-master}} + t_{\text{top-master}} \\ &= 4 \cdot 3.86 \text{ ms} + 1.59 \text{ ms} = 17.02 \text{ ms} \end{aligned} \tag{12.4}$$

However the slaves will in practice never finish the algorithms at the exact same time and thus not start sending the data at the same time. There are multiple ways to reduce this time consumption, e.g. reduce the resolution of the distance maps by using 2 bytes per pixel instead of 4 or calculate the distance map on the master and send the binary foreground mask from the slaves instead. Both of these solutions will of course take up a bit more time on the master, either because of the distance transformation or because of required decompression. However, the 17 ms out of a total time of $1/15 = 66.7$ ms is not considered a problem and this interface will be kept for simplicity. Therefore, the algorithms in PerCam are executed on slaves and the algorithms in Sequential are executed on the master for the distributed system.

12.3 Program Flow

With the network interfaces in place, the program flow can be designed. First the flow at the slaves will be discussed followed by the flow on the master.

12.3.1 Slaves

The algorithms to be executed on the slaves are the ones included in the PerCam class, shown in Figure 11.2. When the algorithms have finished, the distance maps and feature points are ready to be send to the master. To utilise the slaves as much as possible, the slaves could start the next iteration while sending the results to the master. However, some kind of synchronisation between the master and slaves is needed. Also, both the 2D foreground detection and the feature point tracking need feedback from the person tracker. The slaves therefore need to wait for the projected targets from the master. The two first functions to be performed on the slaves (loading of images and downscaling of the images) do, however, not depend on the algorithms performed on the master. Therefore, the slaves can perform these two tasks first, grupoed into the function prepareFrames. When the projected targets have been received from the master, the remaining algorithms can be executed, grouped into processFrames.

One can argue that the loading of the next frame should wait for signal from the master in order to be synchronised. However, the system is designed to run in real-time, which for the test setup at AIT is 15 fps. With this frame rate, one synchronisation per iteration is sufficient, as the speed difference between the slaves is small compared to the movement of the persons between two frames. The flow on the slaves after TCP connections have been established to the master is shown as pseudo-code in Figure 12.1.

- `Wait for start signal from the master.`
- **`While`** `not exit`
  - `Call the function prepareFrames of the class PerCam.`
  - `Wait for projected targets from the master.`
  - `Call the function processFrames of the class PerCam.`
  - `Start sending the distance map and the feature points to the master.`

Figure 12.1: *The program flow on the slaves.*

12.3.2 Master

The master receives the distance maps and feature points from the slaves and stores these in the back buffer instance of the ProtectedData class (see Section 11.1), and the algorithms to be run by the master is the algorithms of the Sequential class, illustrated in Figure 11.3. After finishing this, the master must send the projected targets to the slaves so that they can continue on the next iteration. The program flow on the master is summed up as pseudo-code in Figure 12.2.

- `Send start signal to the slaves.`
- **`While`** `not exit`
  - `Send the projected targets to the slaves so that they can go on with the next iteration.`
  - `Wait for the distance maps and feature points from the slaves.`
  - `Call the algorithms of the class Sequential.`

Figure 12.2: *The program flow on the master.*

12.4 Implementation

To ease the implementation and at the same time make it possible to send/receive data at the same time as running the algorithms, a separate thread is created on both sides per connection from the master to the slaves. These extra threads are implemented as active classes, *MasterConnectionHandler* and *SlaveConnectionHandler* in the same manner as the Renderer class described in Chapter 11. A class diagram of both the master and slaves is shown in Figure 12.3. The two classes Renderer and ProtectedData are the same as introduced in Chapter 11 but of course not all functions are used by neither master nor slaves.

The master is implemented as a TCP server. When the program is started, it blocks waiting for five slaves to connect and starts a separate thread with a socket for each connecting slave. Initial data such as camera calibration parameters and various configuration parameters is shared between the master and slaves. When all connections have been established, the master sends a signal to the slaves indicating that they can start the algorithms. The master also sends an empty list of projected targets to the slaves so they can start the first iteration. A sequence diagram is shown in Figure 12.4.

12.5 Discussion

In this chapter the design of a distributed version of the full system implemented in Chapter 11 has been described. However, due to limited time for this project, some parts of this was not actually implemented:

- No feedback of the projected targets is implemented, only a signal is sent from the master to the slaves indicating when they can continue with the next iteration.
- The feature point tracker is not available and no feature points are sent from the slaves.
- Only the camera id and image resolution are sent as initial data. The rest of the configuration parameters are manually specified in the configuration files for both the master and the slaves before the system can be started.

This limited system has been tested to run live at 14 fps on the test setup at AIT, cf. Chapter 13. The most important missing part of the tracking system, which is also the most time consuming part that has not been included in the distributed system, is the feature point tracker. As

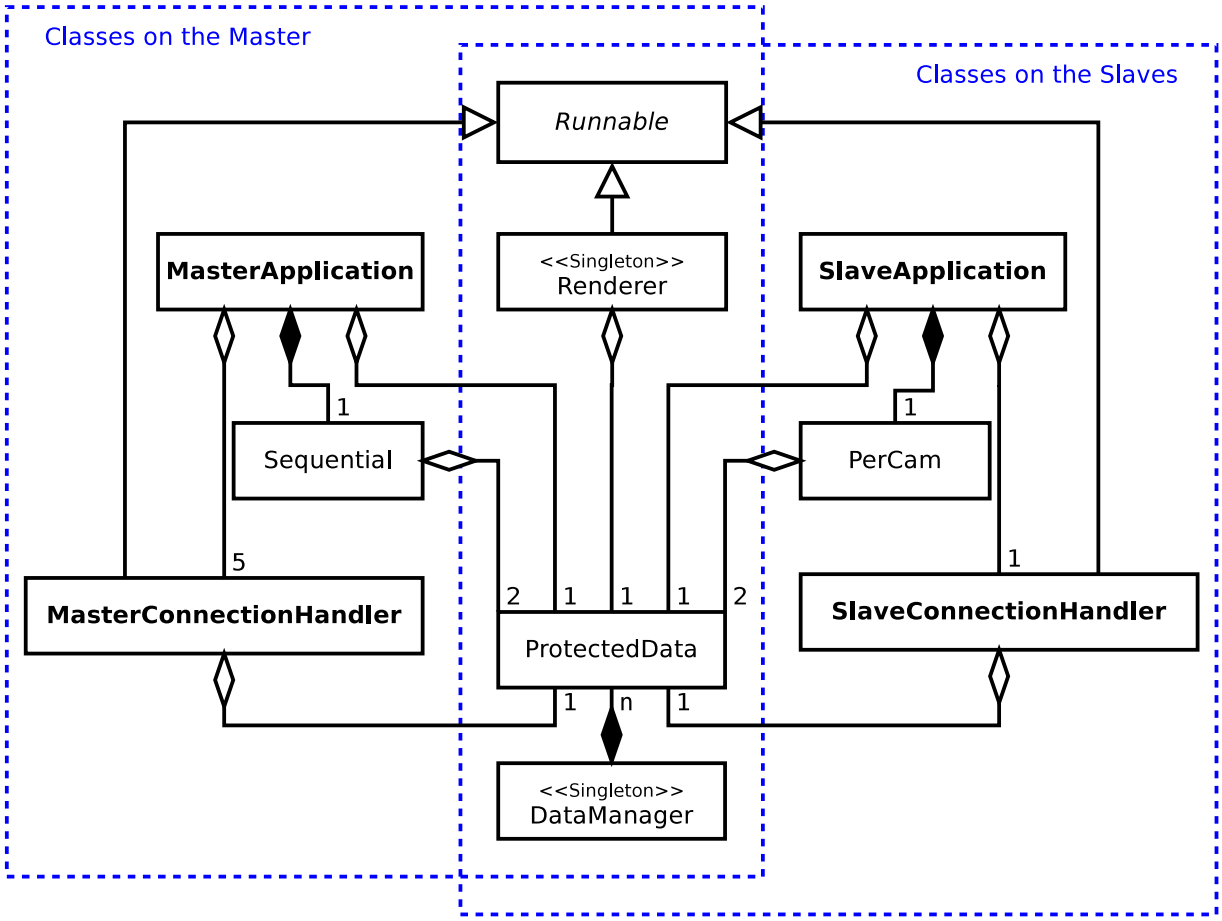


Figure 12.3: *Class diagram of the master and slaves with new classes introduced in the distributed system marked in bold.*

discussed in Section 11.2, this part is mostly independent of the other algorithms that runs per camera, an can thus be parallelised since the benefit of this on a single dual-core computer will be very limited. However, in the distributed system this could be advantageous. Also the feature point tracking can run in parallel with sending of the distance maps to the master. This also means that the master can begin executing the algorithms that do not depend on the feature points, which is the combination of the distance maps and some of the target management algorithms.

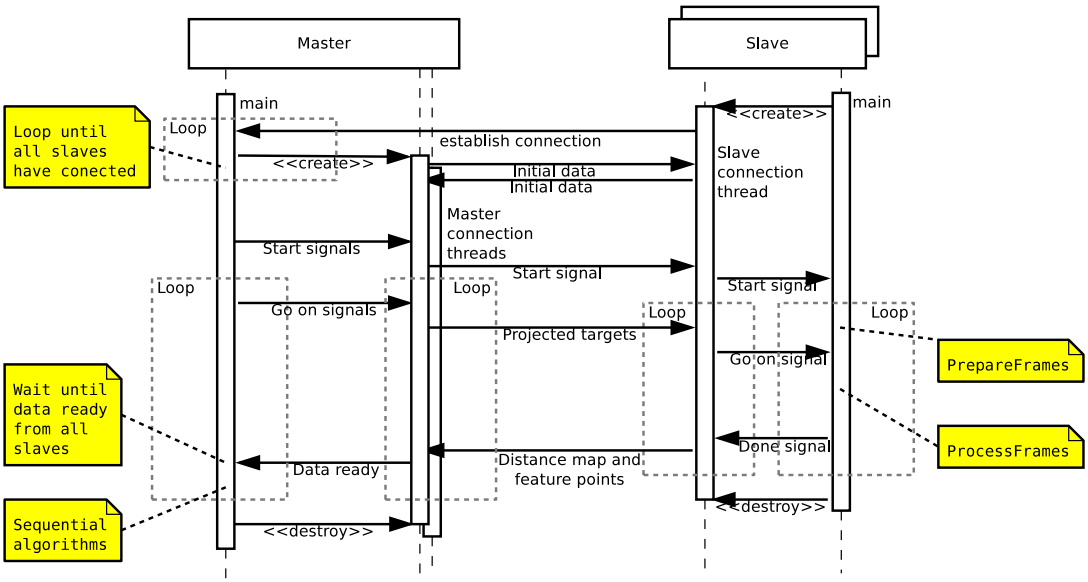


Figure 12.4: *Sequence diagram of the distributed system.*

Part V

Evaluation

Contents

The developed system is in this part evaluated through various tests to provide the foundation for a conclusion on the entire project in Chapter 15. In Chapter 13, the time consumption of the system is measured and analysed with the purpose of real-time execution in mind. A qualitative test of the systems ability to reason about the mobility and body posture of persons in the scene is also presented. In Chapter 14, the systems ability to track multiple persons is determined by testing on the standardised CLEAR 2007 data set. The performance of the system using either foreground, feature points, or both for tracking is measured and compared to previously developed systems that has been tested on the same data set.

Chapter 13

Time Consumption and Reasoning Tests

The purpose of this chapter is both to measure the computational requirements of the developed system and to determine whether real-time execution is possible, and to test the systems performance for reasoning about mobility and body posture of the persons in the scene. As described in Section 1.1.1, the development setup will be used to test the live performance of the system distributed to 6 computers, 5 of these connected to cameras and one functioning as master. Ideally, both the reasoning performance as well as the time consumption of the system should be measured on standardised data sets to allow comparison with similar systems. However, reasoning and time consumption is not part of the CLEAR evaluation metrics, and papers using the CLEAR data set does not generally publish the time consumption of their algorithm. Therefore it is chosen only to measure both reasoning performance and time consumption on a single set of videos; the development data set described in Section 1.1.1. The time consumption measurements are presented in Section 13.1 and 13.2 and the reasoning performance is evaluated in Section 13.3.

The time consumption with four different settings of the system is tested:

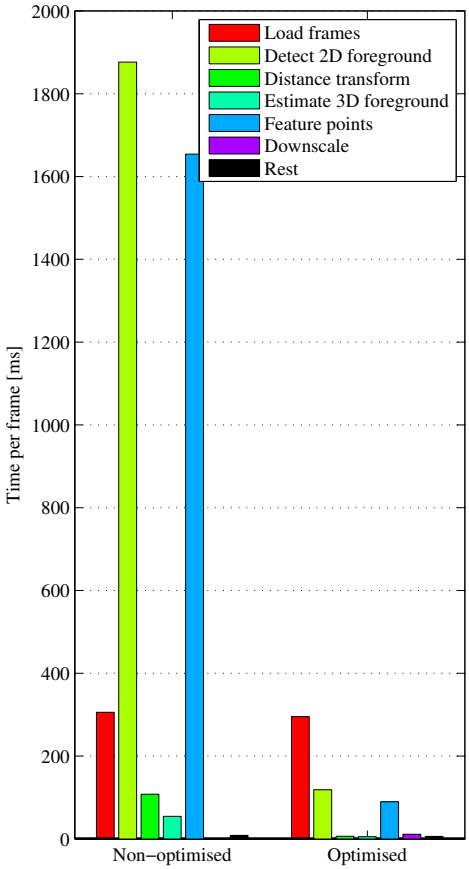
- **Non-optimised:** The non-optimised setting uses full resolution images and the non-hierarchical 3D foreground estimation.
- **Optimised:** The optimized setting uses downscales the images by a factor of 4 for both foreground detection and feature points, and uses the hierarchical 3D foreground estimation algorithm designed in Chapter 5 with a 4 level octree.
- **Parallel:** The parallel setting uses the same algorithm as the optimised setting, but performs a number of the calculations in parallel to take full advantage of the dual core CPU. The parallelisation is described in Section 11.2
- **Distributed:** The distributed system described in Chapter 12 running on live video.

The performance with the first three settings can be compared directly since the same hardware and the same recording is used. Additionally, the quality with the optimised/non-optimised settings are almost identical, as described in Section 4.3.1, 5.3.1 and 6.3.3, while the parallelisation does not affect the algorithms at all. The performance with the distributed setting cannot

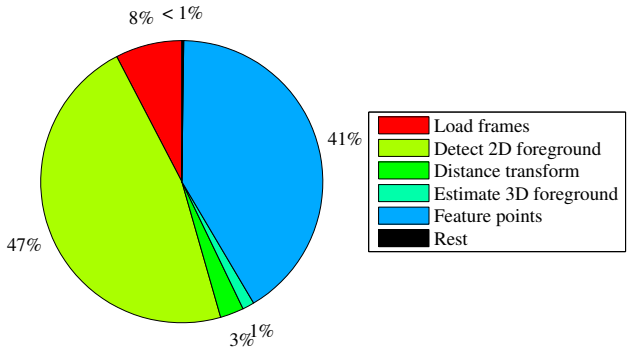
be compared directly with the other, however, for a number of reasons including: Only the foreground only tracker is used, other computers perform the test and live video is used with a different number of persons than in the recorded data set.

13.1 Single Computer Tests

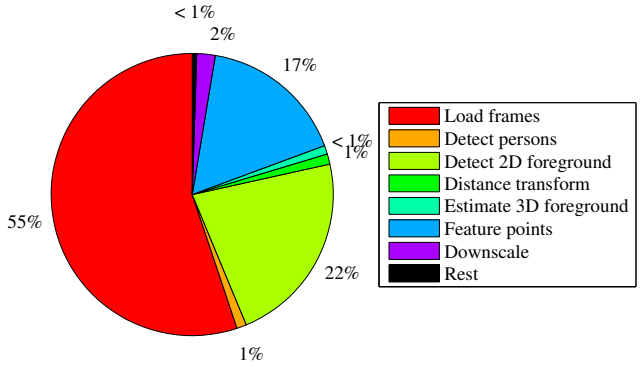
All single computer tests was carried out on a 2.2 GHz dual-core computer with 5 GB of memory. Figure 13.1 shows the time consumption for the optimised and non-optimised settings (the precise time comsumption are given in the Appendix in Table D.1). The optimisation reduced the total time from 4.01 sec to 535 ms, primarily by reducing the image resolution and thus the time required both to detect foreground in 2D and to track feature points. Also the time to estimate 3D foreground is reduced significantly, however, by using the hierarchical approach designed in Chapter 5.



(a) Absolute time consumption with different settings.



(b) Relative time consumption with the non-optimised setting.



(c) Relative time consumption with the optimised setting.

Figure 13.1: *Time consumption using the non-optimised and optimised settings. The total time is reduced from 4.01 sec to 535 ms by applying the optimisations. Most of the reduction is achieved by reducing the image resolution and thus the time required both to detect foreground in 2D and to track feature points.*

With the parallel setting, it is not possible to measure the time consumption of the different parts of the system in a meaningful way. Instead, the computation speed for the entire system with the three different settings has been measured. The results are illustrated in Figure 13.2.

The time consumption has been reduced to 372 ms per frame compared to 535 ms with the optimised sequential setting. This corresponds to a frame rate of 2.69 fps.

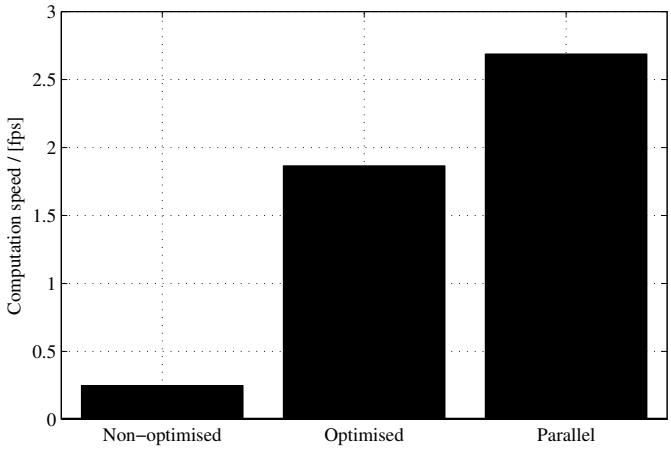


Figure 13.2: *Computation speed for the non-optimised, optimised and parallel implementations.*

13.1.1 Discussion

With the optimised setting, the single computer tests show that most of the time consumption is devoted to loading of images. As shown in Figure 13.1c, the loading constitutes 55% of the total time consumption with the optimised sequential setting. Since this time is used to load from the hard drive, it will not have changed significantly in the parallel system. On a live system, however, use of the hard drive is not required, and capturing of images can be expected to run significantly faster.

Loading, downscaling images, detecting 2D foreground, and tracking feature points together constitute 96% of the time consumption with the optimised sequential setting, as illustrated in Figure 13.1c. All of these parts is well suited for distribution to multiple computers, as described in Chapter 12. Therefore, distribution can also be expected to speed up the computation speed significantly.

13.2 Distribution Test

The distributed test was carried out on the system described in Section 1.1.1 using live video for 33 minutes. All computers in the setup are dual-core and running at 3.0 GHz, and the frame rate achieved was 14.03 fps. The time consumption for the different parts of the algorithm run on the distributed system is is illustrated in Figure 13.3 for the master and slaves, respectively. The exact measured time consumptions are given in the appendices in Table D.2.

13.2.1 Discussion

In Figure 13.3a, the time consumption for each of the slaves is shown, and especially the first and the last slave behaves differently than the rest. Both of these are older model computers than

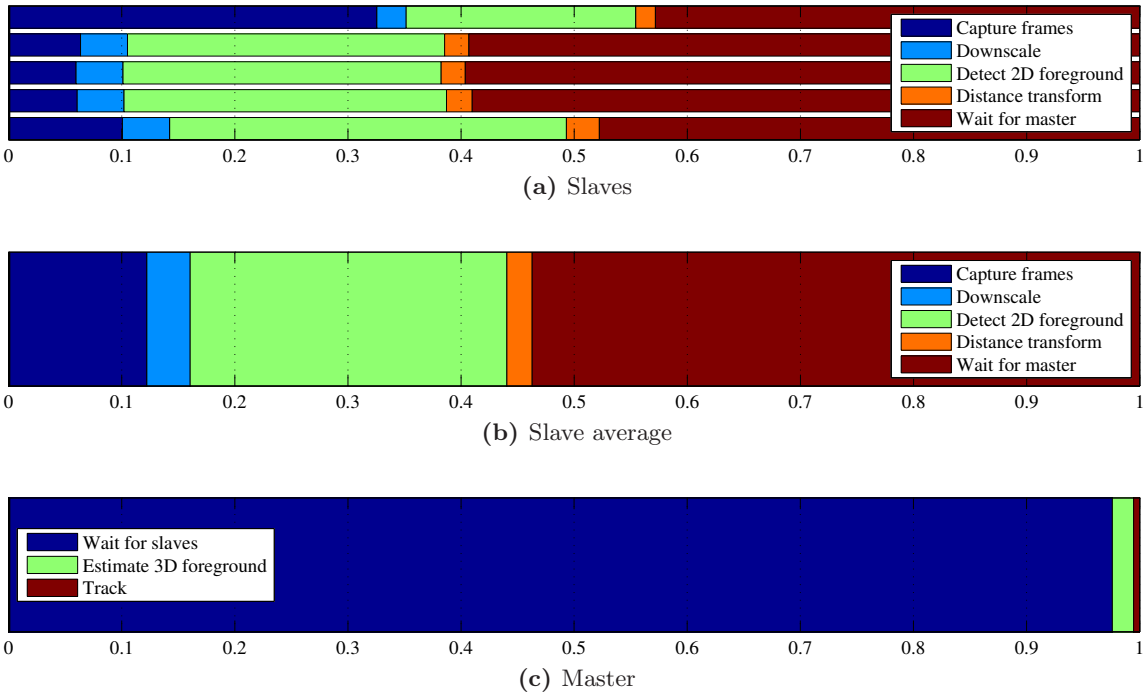


Figure 13.3: *Relative time consumptions for the distributed system. The total time consumption for one cycle took in average 71.3 ms, but has here been normalised to 1. Note that data transfer is handled in parallel threads, and is therefore included in the waiting times here.*

the rest, and even though their CPU run at the same clock frequency they are slightly slower. The first slave was connected to the top camera. This camera has a lower resolution and this causes the computation time for this slave to be smaller for the calculation tasks (downscaling, detecting 2D foreground, and performing distance transform). Also, the driver used for the top camera was different, resulting in a larger time necessary to capture new images.

As described in Section 1.1.1, it is possible to record from the cameras in the test setup at a speed of 15 fps. Therefore, with the achieved frame rate of approximately 14 fps, it is not immediately clear whether capturing of new images or the bandwidth of the network is the major bottleneck in the system. Figure 13.3b shows that the master uses almost all of its time waiting for the slaves, and Figure 13.3c shows that the slaves use most of their time waiting for the master. Capturing of images only takes up about 12% of the time consumption for the slaves in average. This seems to indicate that the network is the limiting factor in the system.

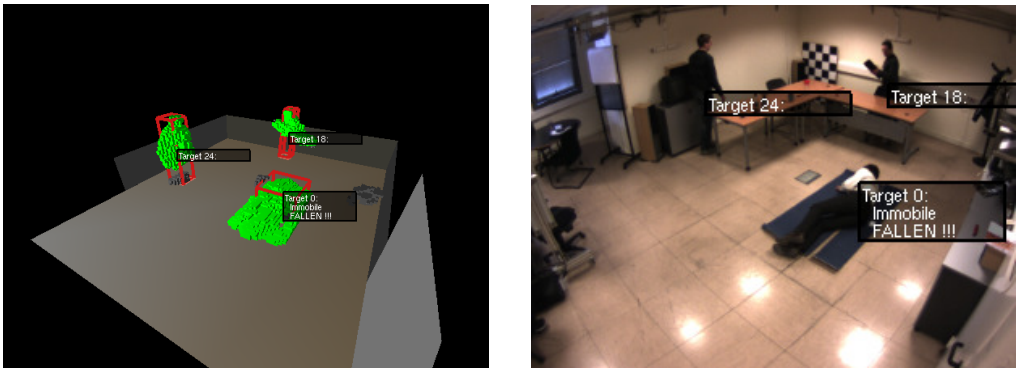
Figure 13.3a shows that the first computer uses about 35% for capturing images, however, which could also be a limiting factor. According to the calculations in Equation (12.2) on page 90 concerning data transfer from the slaves to the master, the time required to send data from each slave (except the top camera slave) will be at least 3.86 ms. This means that the total time required to send the data from all four corner camera slaves is 15.36 ms, corresponding to 21.7% of the total measured time consumption of 71.3 ms. By comparison of the different slaves in Figure 13.3a it can be seen that the time from the first slave starts sending until the top camera slave starts sending is approximately 17% of the total time. This means that the corner slaves have not finished the transfer when the top camera slave begins, and the long time required to capture the images does therefore not delay the total time consumption. It must therefore be concluded that the network is indeed the bottleneck in the system.

The feature point part of the combined tracking algorithm was not implemented in the distributed system due to time constraints of this project. As described in Section 12.5, the feature point tracking can be carried out on the slaves while the distance maps are send. For the single computer sequential implementation, the feature point tracking took up 90 ms in total, corresponding to 18 ms per camera. The computers used in the distributed setup are faster than the one used for the single computer tests, and thus the time consumption can be expected to be lower. Since is takes *at least* 17 ms to send the distance maps (cf. Equation (12.4) on page 12.4), including the feature point tracking cannot be expected to increase the total time consumption significantly. Therefore it is concluded that real-time execution of the entire combined tracking system should be possible.

13.3 Qualitative Reasoning Tests

As described in the problem formulation, Chapter 3, one purpose of our system is to determine the mobility of persons in the scene and to detect if a person falls. Due to the absence of standardised test material to test these abilities, the standard development data set has been used to perform a qualitative test instead. In this, up to four people move around in the scene, and at 3 occasions in total does a person fall while a person sits down at 6 occasions. All of these events are detected correctly and there are no standing/sitting persons that are falsely detected as fallen. When a person kneels or bows he is occasionally classified as sitting but not as fallen.

The mobility detection is able to correctly classify all persons that are completely immobile as stationary and all people that are walking as mobile. People that are not walking but are active in the same location (e.g. by moving objects around) are sometimes classified as stationary and sometimes as mobile. An example of both mobility estimation and fall detection is shown in Figure 13.4, and a video documenting the reasoning performance of the system on the development data set is provided on the CD attached to this report.



(a) 3D foreground with detected targets and reasoning results. Targets not shown as "immobile" have been classified as mobile.

(b) Frame from corner camera with the same reasoning information as in (a).

Figure 13.4: *Example of the fall detection from the development data set. A video documenting the reasoning performance of the system on the entire development data set is provided on the attached CD.*

Chapter 14

CLEAR Evaluation

As described in Chapter 1, the developed system is to be tested on the CLEAR 2007 data set [19] to be able to compare its performance with similar systems. The CLEAR data set contains 40 video recordings of meeting rooms equipped with 5 cameras. The videos are recorded at five different locations with 8 recordings of 5 minutes from each location (cf. Section 1.1). The performance of the tracking systems running on these recordings are evaluated by comparing with hand annotated ground truth. The scores are calculated according to the official CLEAR evaluation metric, which is described in Section 1.2. The test results for our tracker system is presented in Section 14.1, and a comparison with the participating systems from the CLEAR 2007 workshop is given in Section 14.2. The exact results for each video sequence are shown in Appendix D.

14.1 Test Results

In Table 14.1 the CLEAR test results of the three trackers are shown; the foreground only, the feature point only and the combined tracker. The displayed values are averages over all the 40 recordings in the CLEAR data set. The parameters of our system have been adjusted identically for all recordings as described in the report except for the recordings from AIT. The camera setup is different for these recordings, in the sense that much of the room is only visible by 2-3 cameras. Therefore, all voxels in scene that can be seen by at least two cameras have been used for the 3D foreground repressentation here, compared to four required cameras for all other recordings.

A comparison of the foreground only and feature point only trackers reveals that the MOTP is much better for the foreground tracker than the feature point tracker, as was to be expected. On the other hand, the MOTA score is better for the feature point tracker.

Even though the MOTA score for the feature point tracker is better than for the foreground tracker, Table 14.1 shows that it is not better in every sub-score. The foreground tracker has a miss rate that is more than twice as high as the feature point tracker, while the feature point tracker has a much larger amount of false positives than the foreground tracker.

Table 14.1 also includes the results of the combined tracker. The MOTA value is increased from 29% and 37% for the individual trackers to 50% for the combined tracker. The combined

Tracker type	MOTP (mm)	Miss rate (%)	False pos. rate (%)	Mismatches	MOTA (%)
Foreground	137	65.42	4.70	247	29.30
Feature points	215	30.76	31.09	422	37.16
Combined	145	43.87	5.57	334	49.80

Table 14.1: *CLEAR test results for the three trackers.*

tracker does not outperform the individual trackers on all sub-scores, but has almost as low a false positive rate as the foreground tracker while the miss rate is approximately the average of the two individual trackers. The MOTP value is only slightly lower for the combined tracker than for the foreground tracker. This is caused by the fact that foreground is not always present. When no foreground is present, the foreground tracker simply fails, while the combined tracker might be able to keep tracking correctly, although with a lower precision.

14.1.1 Discussion

A number of aspects are worth pointing out in the scores, first in the comparison between the different tracking systems:

Precision of trackers: The combined tracker achieved almost as good MOTP as the best of the individual trackers. That the foreground tracker performed slightly better is not a real problem, since a match with low precision is much to be preferred compared to a miss.

Difference between individual trackers: Even though the feature point tracker got a higher MOTA score than the foreground tracker, it is an important point that these scores completely different sub-scores, both seen per parameter and per video. This is a major reason that these modalities was selected in the first place (cf. Section 2.3). The foreground tracker can easily loose track of a person if he remains immobile for a period, and this gives a high miss rate. The feature point tracker on the other hand is able to keep track of immobile targets. If a stationary scene object by mistake is tracked by the feature point tracker, however, it will result in a high false positive rate.

A closer look at the performance of the combined tracker has revealed three general problems:

Modality infighting: The miss rate of the combined tracker was approximately the average of the two individual trackers, which is of course higher than what should be expected. A look through the tracking performance expose the reason. A problem occurs when a person is faded into the background but still is well marked with feature points. In this case the feature point tracker has no problem keeping the track but the combined tracker tends to fail if a person is mobile nearby. What happens is that the foreground part of the tracker tries to move the target to the area with foreground, while the feature point part tries to keep the target at the originally tracked person. The target is first moved so that it is located between the two persons. This fight between the different parts are always lost by the feature point part because feature points far away from the target are erased.

Initialisation problems: The results that the individual trackers scored below 40% in the MOTA is not ostentatious. In Appendix D the results from the CLEAR data set is shown per test video. It shows that especially on some of the test videos from UKA both trackers have huge problems. In some of the UKA videos all persons are sitting and remaining very immobile during the entire video. Since the system is designed to only initialise targets in the first frame or on persons that are standing, no targets are initialised after the first frame. This means that the system relies very much on the frames of the background provided with the video to separate the persons from the background in the first frame. In some of the videos, the people are sitting very close and pens and papers have also been moved around while the lightning conditions has changed between the background frames and the first frame. This causes the initialisation algorithm (designed in Section 7.2) to be unable to separate the persons, and therefore too few targets are initialised. The lack of movement leaves very little chance to initialise new feature points even if the target is tracked, and therefore all targets that *are* initialised, are lost after some time.

Few feature points: Another thing that is obvious when looking through the tracking results of the feature point tracker is that in scenes with many persons, only feature points from a few cameras are present. This is caused by the design decision that feature points are only initialised in a particular camera if the person from that point of view does not overlap with another persons.

To sum up on the performance of the combined tracker, each parameter it almost as good as for each of the two individual trackers except the miss rate which is close to the average of the two. Overall, the combined tracker is much better than the individual trackers and thus fulfils the hypothesis in Chapter 3.

14.2 Comparison with Previous Systems

To determine how well the tracker performs, the results are compared with the participants of the CLEAR 2007 workshop. Table 14.2 presents the result of the combined foreground and feature point tracker together with the 7 CLEAR 2007 systems. The precision of our tracker is comparable to the other systems, as our tracker with regards to the MOTP value is placed 4'th of the 8 systems. Compared to the other systems, our tracker has a very low number of mismatches and a low rate of false positives. However, a large miss rate causes the MOTA value to be only 7'th. As described in Section 1.2 the mismatches do not have a big impact on the score and the low false positive rate is not enough to compensate for the high miss rate.

Although the performance of our combined foreground and feature point tracker is significantly better than the individual trackers, it can not keep up with most of the competing systems in the CLEAR scenario where the persons are sitting down and remaining stationary most of the time. The primary reason is that the persons tend to fade into the background after some time.

System	MOTP (mm)	Miss rate (%)	False pos. rate (%)	Mismatches	MOTA (%)
AIT Prim. [46]	92	30.86	6.99	1139	59.66
AIT Cont. [46]	91	32.78	5.25	1103	59.56
FBK [52]	141	20.66	18.58	518	59.62
UKA Prim. [8]	155	15.09	14.50	378	69.58
UKA Cont. [8]	222	23.74	20.24	490	54.94
UPC Prim. [17]	168	27.74	40.19	720	30.49
UPC Cont. [17]	147	13.07	7.78	361	78.36
Proposed system	145	43.87	5.57	334	49.80

Table 14.2: *CLEAR test results for the 7 systems presented at the CLEAR 2007 workshop compared with our proposed combined foreground and feature point tracker.*

Chapter 15

Conclusion

In this project methods for tracking multiple people in multi-camera environments have been examined, and a novel algorithm based on a combination of adaptive foreground estimation and feature point tracking has been proposed. Tracking systems are relevant for many applications, one of these being assistive living environments for elderly. The importance of this area is already huge and it will only increase in the coming decades since the amount of elderly in the western societies are going to increase significantly; in Denmark from 16% of the population in 2010 to 25% in 2042 [20]. With this application in mind, reasoning about mobility and body posture of the tracked persons was also developed.

15.1 Methods

A general framework for tracking was presented in Chapter 7, and for this system it was chosen to base tracking on particle filters. This is a discrete multi-hypothesis approach to Bayesian filtering, and it natively supports the integration of more modalities. Much research has previously been done within particle filter tracking using both one or multiple modalities. For this project, a novel approach based on adaptive foreground estimation and feature points was chosen. Both techniques are well known: The adaptive foreground estimation is based on work by Stauffer et. al. [71] and the feature point tracking based on the KLT-tracker [69]. It was argued in Chapter 2 that they complement each other well, since foreground is present when a person is moving while feature points can be tracked indefinitely when a person is stationary. Thus, it was hypothesised that a tracker based on both will outperform a tracker based solely on one of them.

The particle filter technique was presented in Chapter 8 and the particle filter used for this project was designed in Chapter 10. The two modalities are combined using a layered sampling approach, first introduced by Blake et. al. in 2000 [64]. The developed algorithm was implemented in software as described in Chapter 11, and a distributed version was designed to allow the program to run live. The distributed version is described in Chapter 12. It was only partly implemented, however, so the distributed system tracks persons based only on foreground.

15.2 Results

The tracking performance of the system was tested on a standardised data set from the CLEAR 2007 evaluation and workshop [19]. This data set contain 40 multi-camera recordings from 5 different locations. Ground truth exist for the entire data set and metrics have been defined (described in Section 1.2), which allow comparison of different tracking systems, and it has been used by several other previously developed systems [72].

The main result for our system was a Multiple Object Tracking Accuracy (MOTA) score of 49.80, cf. Chapter 14. To test the hypothesis that a tracker using both foreground and feature points will outperform a tracker based solely on one of them, the system was also tested using each individual modality. The hypothesis was accepted, since the individual trackers both scored below 40.

The CLEAR data set does not show people falling and the CLEAR metrics does not measure the speed of the algorithms. Therefore, a data set was created using a multi-camera setup at the Athens Information Technology [1] with 0-4 people moving around and occasionally falling. Using this, a qualitative test showed that our system could detect 3 of 3 falls without showing any false positives, while also registering whenever each person moved or remained stationary. The speed of the algorithm was measured in Chapter 13, and a frame rate of 2.69 fps was achieved on a 2.2 GHz computer. Much time was spend simply on loading the images from the hard drive, however, and the distributed system was able to run live at 14.03 fps using 6 computers with 3.0 GHz CPU's. The distributed system does not include the feature point tracking, but an analysis of the time consumption has shown that this should not reduce the frame rate significantly. Thus, real-time performance was not achieved on a single computer but is theoretically possible on a distributed setup.

15.3 Perspectives and Possible Improvements

Our foreground tracking system has been described in a paper entitled "Three-Dimensional Adaptive Sensing of People in a Multi-Camera Setup", which has recently been accepted for the 2010 EUSIPCO conference [28]. It is attached to this report as Appendix E (note that this is a draft, and that the final version might include minor changes based on the reviews).

Our complete system currently scores 7'th of 8 when compared to the tracking systems participating in the CLEAR 2007 workshop [72]. While this is not an impressive performance, we believe that significant improvements are possible. Three major problems of the system are described in Section 14.1.1, and each of these could be minimised or eliminated completely:

- **Modality infights:** When no foreground is present at a stationary person, the tracker tends to jump to a nearby moving person. This could be avoided by balancing the foreground and feature point parts better, e.g. by combining modularities in a combined likelihood, as described in Section 10.4, Equation (10.6). Another possibility is to consider the foreground part as unreliable and only let it affect the target little if no or little foreground has been present for some time.
- **Initialisation problems:** To avoid initialising targets on moved objects such as chairs, an object is required to be as tall as a standing person to cause initialisation. Only in the first

frame is it accepted for targets to be initialised on lower objects. In some of the CLEAR videos this causes persons which are sitting the entire videos to be missed completely. We believe the initialisation procedure could be improved, either by fine tuning or by using an additional modality such as face detection to detect stationary persons.

- **Few feature points:** When many persons are located close to each other they might overlap from most of the cameras. To avoid initialising feature points on a wrong person, no points are initialised in this situation. If no or very few points get initialised on a stationary person, the tracker will fail when he fades into the background. A possible improvement could be to instead initialise feature points in the part of each target that *do not* overlap with other targets.

It is our goal to get an additional paper, describing the entire system, accepted in a recognised journal. For this to succeed, the performance of the system must not be significantly worse than the state-of-the-art in the field. Therefore, some or all of these improvements will be attempted.

Bibliography

[1] Athens Information Technology. http://www.ait.gr/.

[2] M. S. Allili and D. Ziou. Object tracking in videos using adaptive mixture models and active contours. *Neurocomput.*, 71(10-12):2001–2011, 2008.

[3] B. Antić, J. O. N. Castaneda, D. Ćulibrk, A. Pižurica, V. Crnojević, and W. Philips. Robust Detection and Tracking of Moving Objects in Traffic Video Surveillance. *Advanced Concepts for Intelligent Vision Systems*, pages 494–505, 2009.

[4] S. Baker and I. Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *IJCV*, 56(3):221–255, February 2004.

[5] S. Baker, R. Patil, K. M. Cheung, and I. Matthews. Lucas-Kanade 20 Years On: Part 5. Technical Report CMU-RI-TR-04-64, Robotics Institute, Pittsburgh, PA, November 2004.

[6] S. Battiato, G. Gallo, G. Puglisi, and S. Scellato. SIFT Features Tracking for Video Stabilization. In *ICIAP '07: Proceedings of the 14th International Conference on Image Analysis and Processing*, pages 825–830, Washington, DC, USA, 2007. IEEE Computer Society.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded-Up Robust Features. *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[8] K. Bernardin, T. Gehrig, and R. Stiefelhagen. Multi-level Particle Filter Fusion of Features and Cues for Audio-Visual Person Tracking. In *Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007, Baltimore, MD, USA, May 8-11, 2007, Revised Selected Papers*, pages 70–81, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: the CLEAR MOT metrics. *J. Image Video Process.*, 2008:1–10, 2008.

[10] S. Birchfield. Derivation of Kanade-Lucas-Tomasi Tracking Equation, January 1997. http://www.ces.clemson.edu/ stb/klt/birchfield-klt-derivation.pdf.

[11] S. S. Blackman. *Multiple-target tracking with radar applications.* Dedham, MA, Artech House, Inc., 1986.

[12] G. Borgefors. Distance transformations in digital images. *Comput. Vision Graph. Image Process.*, 34(3):344–371, 1986.

[13] J.-Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the algorithm. http://robots.stanford.edu/cs223b04/algo_tracking.pdf.

[14] J.-Y. Bouguet. Camera Calibration Toolbox for Matlab, 2008. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[15] T. Bouwmans, F. E. Baf, and B. Vachon. Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey. *Recent Patents on Computer Science*, 1(3):219–237, 2008.

[16] C. Canton-Ferrer, J. R. Casas, and M. Pardàs. Towards a Bayesian Approach to Robust Finding Correspondences in Multiple View Geometry Environments. In *International Conference on Computational Science (2)*, pages 281–289, 2005.

[17] C. Canton-Ferrer, J. Salvador, J. R. Casas, and M. Pardàs. Multi-person Tracking Strategies Based on Voxel Analysis. In *Multimodal Technologies for Perception of Humans*, pages 91–103, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] O. Cappe, S. Godsill, and E. Moulines. An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924, May 2007.

[19] CLEAR - Classification of Events, Activities and Relationships. http://www.clear-evaluation.org/.

[20] Danmarks Statistik. Andelen af ældre vil stige i mere end 30 år. *Nyt fra Danmarks Statistik*, 219, May 2010.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[22] F. Devernay, D. Mateus, and M. Guilbert. Multi-Camera Scene Flow by Tracking 3-D Points and Surfels. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:2203–2212, 2006.

[23] A. Dore, A. Beoldo, and C. Regazzoni. Multitarget Tracking with a Corner-based Particle Filter. *International Workshop on Visual Surveillance*, 2009.

[24] A. Doucet, S. Godsill, and C. Andrieu. On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10:197–208, 2000.

[25] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley-Interscience Publication, Second edition, 2000.

[26] C. Ericson. *Real-Time Collision Detection.* Morgan Kaufmann, 1 2005.

[27] C. Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology).* Morgan Kaufmann, January 2005.

[28] EUSIPCO - 2010 European Signal Processing Conference. http://www.eusipco2010.org/.

[29] X. H. Fang, W. Xiong, B. J. Hu, and L. T. Wang. A Moving Object Detection Algorithm Based on Color Information. *J. Phys.: Conf. Ser.*, 48(1):384+, 2006.

[30] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian Filtering for Location Estimation. *IEEE Pervasive Computing*, 2(3):24–33, 2003.

[31] N. Friedman and Russell. Image segmentation in video sequences: A probabilistic approach. In *Thirteenth Conf. on Uncertainty in Artificial Intelligence*, pages 175–181, 1997.

[32] C. Harris and M. J. Stephens. A combined corner and edge detector. *Proceedings of the Fourth Alvey Vision Conference*, 1988.

[33] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, Second edition, 2004.

[34] J. Heikkila and O. Silven. A Four-step Camera Calibration Procedure with Implicit Image Correction. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1106, 1997.

[35] Cognitive Care and Guidance for Active Aging: HERMES Technology. http://www.fp7-hermes.eu/hermes-technology.html.

[36] T. Horprasert, D. Harwood, and L. S. Davis. A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection. In *ICCV Frame-Rate WS*, 1999.

[37] A. hua Chen, M. Zhu, Y. hua Wang, and C. Xue. Mean shift tracking combining SIFT. *ICSP 2008. 9th International Conference on Signal Processing*, pages 1532 –1535, October 2008.

[38] M. Isard and A. Blake. CONDENSATION - Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[39] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL*, 1997.

[40] P. KaewTrakulPong and R. Bowden. Adaptive Visual System for Tracking Low Resolution Colour Targets. In *BMVC*, 2001.

[41] P. Kaewtrakulpong and R. Bowden. An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection. In *Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems*, September 2001.

[42] P. KaewTrakulPong and R. Bowden. A real time adaptive visual surveillance system for tracking low-resolution colour targets in dynamically changing scenes. *Image Vision Comput.*, 21(10):913–929, 2003.

[43] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[44] N. Katsarakis, A. Pnevmatikakis, and M. C. Nechyba. 3D Tracking of Multiple People Using Their 2D Face Locations. In *AIAI*, pages 365–373, 2007.

[45] N. Katsarakis, A. Pnevmatikakis, and J. Soldatos. Person tracking for ambient camera selection in complex sports environments. In *DIMEA '08: Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, pages 458–465, New York, NY, USA, 2008. ACM.

[46] N. Katsarakis, F. Talantzis, A. Pnevmatikakis, and L. Polymenakos. The AIT 3D Audio / Visual Person Tracker for CLEAR 2007. In *Multimodal Technologies for Perception of Humans*, pages 35–46, Berlin, Heidelberg, 2008. Springer-Verlag.

[47] Z. Khan, I.-H. Gu, T. Wang, and A. Backhouse. Joint anisotropic mean shift and consensus point feature correspondences for object tracking in video. pages 1270–1273, July 2009.

[48] H. Kim, R. Sakamoto, I. Kitahara, T. Toriyama, and K. Kogure. Robust silhouette extraction technique using background subtraction. *10th Meeting on Image Recognition and Understand (MIRU 2007), Hieroshima, Japan*, July 2007.

[49] A. Koutsia, T. Semertzidis, K. Dimitropoulos, N. Grammalidis, A. Kantidakis, K. Georgouleas, and P. Violakis. Traffic Monitoring using Multiple Cameras, Homographies and Multi-Hypothesis Tracking. In *3DTV Conference, 2007*, pages 1–4, May 2007.

[50] J. L. Landabaso and M. Pardàs. Foreground Regions Extraction and Characterization towards Real-Time Object Tracking. In *Proceedings of Multimodal Interaction and Related Machine Learning Algorithms (MLMI)*, Lecture Notes in Computer Science. Springer, 2005.

[51] J. L. Landabaso, M. Pardàs, and L.-Q. Xu. Hierarchical Representation of Scenes using Activity Information. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Philadelphia, PA, USA, March 2005. IEEE Computer Society.

[52] O. Lanz, P. Chippendale, and R. Brunelli. An Appearance-Based Particle Filter for Visual Tracking in Smart Rooms. *Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007, Baltimore, MD, USA, May 8-11, 2007, Revised Selected Papers*, pages 57–69, 2008.

[53] D. S. Lee. Effective Gaussian Mixture Learning for Video Background Subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):827–832, 2005.

[54] L. Li, W. Huang, I. Y.-H. Gu, and Q. Tian. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Transactions on Image Processing*, 13(11):1459 –1472, November 2004.

[55] D. Lowe. Object recognition from local scale-invariant features. *The Proceedings of the 7'th IEEE International Conference on Computer Vision*, 2, 1999.

[56] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Seventh IJCAI*, pages 674–679, 1981.

[57] J. MackCormick and A. Blake. A Probabilistic Exclusion Principle for Tracking Multiple Objects. *International Journal of Computer Vision*, 39:57–71, 2000.

[58] S. Maskell and N. Gordon. A Tutorial on Particle Filters for On-line Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2001.

[59] S. J. McKenna, Y. Raja, and S. Gong. Tracking colour objects using adaptive mixture models. *Image and Vision Computing*, 17(3-4):225 – 231, 1999.

[60] A. S. Micilotta, E. jon Ong, and R. Bowden. Real-time upper body detection and 3d pose estimation in monoscopic images. In *In European Conference on Computer Vision*, pages 139–150, 2006.

[61] D. Moellman and P. Matthews. Video Analysis and Content Extraction. http://videorecognition.com/vt4ns/vace_brochure.pdf.

[62] H. P. Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. PhD thesis, Stanford University, Stanford, CA, USA, 1980.

[63] M. C. Nechyba, L. Brandy, and H. Schneiderman. PittPatt Face Detection and Tracking for the CLEAR 2007 Evaluation. 2008.

[64] P. Perez, J. Vermaak, and A. Blake. Data Fusion for Visual Tracking with Particles. *Proceedings of the IEEE*, 92(3):495–513, Mar 2004.

[65] A. Pnevmatikakis and L. Polymenakos. Kalman Tracking with Target Feedback on Adaptive Background Learning. In *MLMI*, pages 114–122, 2006.

[66] A. Pnevmatikakis, J. Soldatos, F. Talantzis, and L. Polymenakos. Robust multimodal audio–visual processing for advanced context awareness in smart spaces. *Personal Ubiquitous Comput.*, 13(1):3–14, 2009.

[67] Y. Raja, S. J. Mckenna, and S. Gong. Segmentation and Tracking Using Colour Mixture Models. In *In Asian Conference on Computer Vision*, pages 607–614, 1998.

[68] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter, Particle Filters for Tracking Applications*. Artech House, 2004.

[69] J. Shi and C. Tomasi. Good Features to Track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, 1994.

[70] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*, :, November 2007. Implementation available at http://cs.unc.edu/ ssinha/Research/GPU_KLT/.

[71] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):747–757, 2000.

[72] R. Stiefelhagen, R. Bowers, and J. Fiscus, editors. *Multimodal Technologies for Perception of Humans*. Springer Berlin / Heidelberg, 2008.

[73] T. Thormaehlen and S. Birchfield. Open source KLT tracker implementation. http://www.ces.clemson.edu/ stb/klt/.

[74] C. Tomasi and T. Kanade. Detection and Tracking of Point Features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[75] P. Viola and M. J. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[76] A. Waibel and R. Stiefelhagen, editors. *Computers in the Human Interaction Loop*. Springer-Verlang London, 2009.

[77] W.-B. Yang, B. Fang, Y.-Y. Tang, Z.-W. Shang, and D.-H. Li. Sift features based object tracking with discrete wavelet transform. *Wavelet Analysis and Pattern Recognition, 2009. ICWAPR 2009. International Conference on*, pages 380 –385, July 2009.

[78] Y.-C. You, J.-D. Lee, J.-Y. Lee, and C.-H. Chen. Determining location and orientation of a labelled cylinder using point-pair estimation algorithm. *11th IAPR International Conference on Pattern Recognition*, pages 354 –357, 1992.

[79] H. Zhou, Y. Yuan, and C. Shi. Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding*, 2009.

Part VI

Appendices

Contents

Appendix A

Camera Calibration

Calibration of a fixed camera establishes a correspondence between its images and the surrounding world. It is highly relevant for this project, since images from multiple cameras need to be compared to find foreground in 3D.

A camera can be described using two sets of parameters: *Extrinsic* and *intrinsic* parameters. The extrinsic parameters describe a transformation from world coordinates to camera coordinates, while the intrinsic parameters describe a transformation from camera coordinates to image coordinates. The relationships are illustrated in Figure A.1.

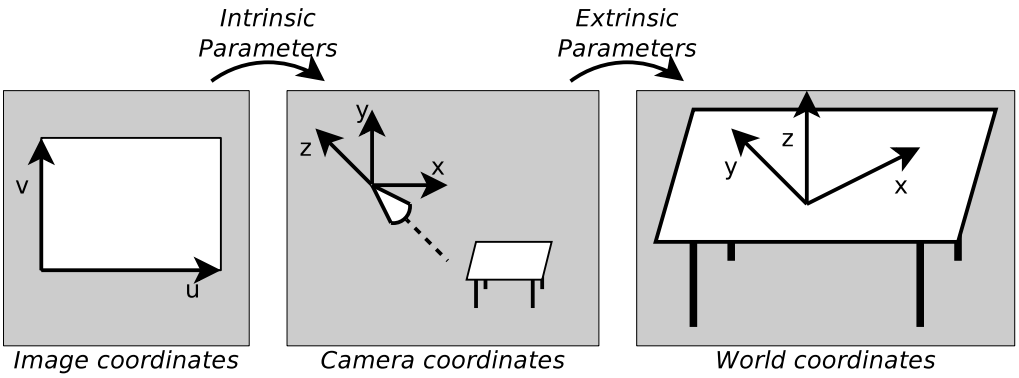


Figure A.1: *Intrinsic and extrinsic parameters.*

For camera calibration in this project, the Camera Calibration Toolbox for Matlab has been used [14] which is based on a paper by Heikkilä et. al. [34]. This toolbox uses the well-defined structure of a 2D checkerboard to estimate all camera parameters from a number of images. For using the calibration parameters to transform 3D points in the world coordinate system into 2D points in the camera image planes, we have made our own C++ implementation that follows the approach of the Matlab implementation.

A.1 Extrinsic Parameters

The extrinsic parameters describe how the camera is positioned and oriented in the world. The extrinsic parameters are given by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ that we define according to [78]. If $\mathbf{p}_\mathrm{w}$ is a point in the world coordinates and $\mathbf{p}_\mathrm{c}$ is the coresponding point in camera coordinates, these can be combined as follows:

$$\begin{bmatrix} \mathbf{p}_\mathrm{w} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_\mathrm{c} \\ 1 \end{bmatrix} \tag{A.1}$$

This equation can be rewritten to give the transformation from world coordinates to camera coordinates:

$$\mathbf{p}_\mathrm{c} = \mathbf{R}^\mathrm{T} \cdot (\mathbf{p}_\mathrm{w} - \mathbf{t}) \tag{A.2}$$

The extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ are estimated using the previously calibrated intrinsic parameters and an image containing a checkerboard. By keeping the checkerboard fixed while calibrating the extrinsic parameters of all cameras, it is ensured that camera coordinates from all cameras can be transformed into the same world coordinate system.

A.2 Intrinsic Parameters

The intrinsic parameters describe the properties of the camera itself, and they are constituted by the *focal length*, the *principal point*, *skew coefficient*, and *distortions* [14].

Calibration of the intrinsic parameters of a camera is normally only necessary once, even if the camera is moved. Only if some of the cameras own parameters are changed, e.g. the focal length, recalibration is necessary. The intrinsic parameters are estimated by the toolbox based on a number of images containing a 2D checkerboard as seen from different viewing angles. Once the intrinsic parameters of the camera have been estimated, it is possible to use this information for calibrating extrinsic parameters from any image containing that same checkerboard.

As an example, both the extrinsic and intrinsic parameters for one calibration of camera 1 is shown in Table A.1:

A.3 Implementation

For our system it is necessary to map both from a world coordinate 3D point to image plane coordinates and back into a 3D vector shooting from the camera in world coordinates. How to do this is explained in the following subsections.

A.3.1 Mapping of 3D Point to Image Plane Pixel

Using the precalibrated extrinsic and intrinsic parameters it is simple to transform from world coordinates to image coordinates in four steps:

Type	Parameter	Value for camera 1	Comments
Extrinsic	Translation:	$\mathbf{t} = \begin{bmatrix} 422 \\ 210 \\ 2259 \end{bmatrix}$	[mm]
	Rotation:	$\mathbf{R} = \begin{bmatrix} 0.66 & -0.25 & 0.71 \\ -0.75 & -0.17 & 0.64 \\ 0.03 & -0.95 & -0.30 \end{bmatrix}$	
Intrinsic	Focal length:	$\mathbf{f} = \begin{bmatrix} 1087 \\ 1084 \end{bmatrix}$	[pixels]
	Principal point:	$\mathbf{c} = \begin{bmatrix} 779 \\ 562 \end{bmatrix}$	[pixels]
	Skew coefficient:	$\alpha = 0$	Typical 0 in modern cameras. Not used in our calibrations.
	Distortions:	$\mathbf{k} = \begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} -0.36 \\ 0.13 \\ -0.0011 \\ 0.0014 \\ 0 \end{bmatrix}$	k_1, k_2, and k_3 are different orders of radial distortion and p_1 and p_2 are tangential distortion. Only the fish-eye camera uses k_3.

Table A.1: *Calibrations parameters. The values for camera 1 are shown as an example.*

Step 1:
Use the extrinsic parameters to convert the point from world coordinates to camera coordinates according to Equation (A.2) to give $\mathbf{p}_\mathrm{c} = \begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix}$.

Step 2:
By normalizing according to the z-value, $\mathbf{p}_\mathrm{c}$ is projected to the image plane (without taking distortion into account):

$$\mathbf{p}_\mathrm{n} = \begin{bmatrix} x_\mathrm{n} \\ y_\mathrm{n} \end{bmatrix} = \begin{bmatrix} \frac{x_\mathrm{c}}{z_\mathrm{c}} \\ \frac{y_\mathrm{c}}{z_\mathrm{c}} \end{bmatrix} \tag{A.3}$$

Step 3:
Both tangential and radial lens distortion is taken into account:

$$\mathbf{n}_\mathrm{n} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \cdot \mathbf{p}_\mathrm{n} + \mathbf{d}_\mathrm{x} \tag{A.4}$$

where the tangential distortion vector $\mathbf{d}_\mathrm{x}$ is defined by:

$$\mathbf{d}_\mathrm{x} = \begin{bmatrix} 2p_1 xy + p_2(r^2 + 2x^2) \\ p_1(r^2 + 2y^2) + 2p_2 xy \end{bmatrix} \tag{A.5}$$

Step 4:
The final pixel coordinates can now be found by using the focal length $\mathbf{f}$ and principal point $\mathbf{c}$ (ignoring the skew coefficient α):

$$\mathbf{p}_\mathrm{pix}(i) = \mathbf{f}(i) \cdot \mathbf{p}_\mathrm{n}(i) + \mathbf{c}(i) \quad \text{for } i = \{1, 2\} \tag{A.6}$$

A.3.2 Mapping of Image Plane Pixel to 3D Vector

Unfortunately, there exists no general algebraic solution to this inverse mapping due to the high order of the distortion model [14]. Instead a numerical iterative algorithm from [14] is implemented in four steps. The first two steps convert the 2D image plane pixel $\mathbf{p}_{\text{pix}}$ into the normalized (2D) image projection $\mathbf{p}_{\text{n}}$. This can be converted to a vector shooting out of the camera in the direction of the pixel under consideration in the third step simply by adding 1 as a z-coordinate. The fourth and last step is to convert this vector into the world coordinate system using the extrinsic parameters. The names of all camera parameters are given in Table A.1.

Step 1:
The normalised image projection is approximated by ignoring distortion:

$$\mathbf{p}_{\text{d}}(i) = \frac{\mathbf{p}_{\text{pix}}(i) - \mathbf{c}(i)}{\mathbf{f}(i)} \quad \text{for } i = \{1, 2\} \tag{A.7}$$

Step 2:
The first guess of the non-distorted projection is $\mathbf{p}_{\text{n}} = \mathbf{p}_{\text{d}}$. The distortions are included by repeating iteration the following equations until convergence of $\mathbf{p}_{\text{n}} = \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ or for a fixed number of times:

$$\begin{aligned} \mathbf{r} &= \sqrt{x^2 + y^2} \\ k_{\text{radial}} &= 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \\ \mathbf{d}_{\text{x}} &= \begin{bmatrix} 2p_1 xy + p_2(r^2 + 2x^2) \\ p_1(r^2 + 2y^2) + 2p_2 xy \end{bmatrix} \\ \mathbf{p}_{\text{n}} = \begin{bmatrix} x \\ y \end{bmatrix} &= \frac{\mathbf{p}_{\text{d}} - \mathbf{d}_{\text{x}}}{k_{\text{radial}}} \end{aligned} \tag{A.8}$$

Step 3:
This is converted to a vector shooting out of the camera in the direction of the pixel under consideration by adding 1 as a z-coordinate: $\mathbf{p}_{\text{c}} = \left[\begin{smallmatrix} \mathbf{p}_{\text{n}} \\ 1 \end{smallmatrix}\right]$, where the index c stands for the camera reference frame.

Step 4:
Equation (A.1) is used to rotate and translate $\mathbf{p}_{\text{n}}$ into the world coordinate system point $\mathbf{p}_{\text{w}}$. The 3D line l is then a line from the camera through the point $\mathbf{p}_{\text{w}}$:

$$l = \mathbf{t} + \alpha \cdot (\mathbf{t} - \mathbf{p}_{\text{w}}) \tag{A.9}$$

where:
$\alpha \in [0, \infty[$

Appendix B

Reasoning

As discussed in the introduction, one important application of tracking is to support elderly in living on their own. Therefore a post processing reasoning stage is implemented on top of the tracker to determine whether a person has fallen, is sitting, or is immobile. Fall detection can be used to alarm assisting personnel while the other detections can be used in higher-level post processing to determine whether a person is behaving "normally" or "anomalous".

B.1 Body Posture

When noise is present in the detected 3D foreground, it is mostly located close to the floor. This is partly due to shadows and partly due to the fact, that other kinds of noise in the 2D foreground detections in most cases are filtered out by the combination of the cameras. This means that the height of the persons can be accurately estimated by taking the maximum vertical position of the foreground voxels located within the 2D-area of the tracked target. By comparing the height with different thresholds, the body posture is identified as either standing, sitting or fallen. A flowchart of the approach is shown in Figure B.1.

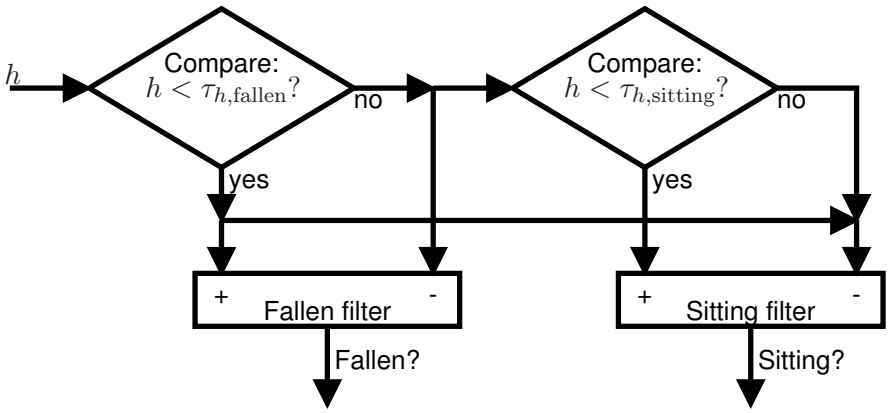


Figure B.1: *Body posture reasoning is based on the maximum height, h, of each target. Counter based filters are applied to provide robustness to noise.*

Noise may cause the maximum height of a target to fall below a threshold in a single frame or for a short duration. To make the posture detection robust, filters are applied on the output.

Counting filters are chosen to make it possible to control the delay from the first time the height gets below a threshold until a fall or sit is detected:

$$s_{\text{fallen}} = s_{\text{fallen}} + 2d_{\text{fallen}} - 1 \tag{B.1}$$

$$s_{\text{sitting}} = s_{\text{sitting}} + 2d_{\text{sitting}} - 1 \tag{B.2}$$

where:
$d_{\text{fallen}} = 1$ if $h < \tau_{h,\text{fallen}}$ and 0 else, and
$d_{\text{sitting}} = 1$ if $\tau_{h,\text{fallen}} \leq h < \tau_{h,\text{sitting}}$ and 0 else.
When the counters s_{fallen} and s_{sitting} gets above specified thresholds, the particular body posture is detected. The counters are thresholded to remain within 0 and 4/3 of the detection delays.

B.2 Mobility

Whether tracked persons are mobile or stationary can be detected by analysing the movement of each target over a predetermined period of time, T. The structure of the mobility reasoning is shown in Figure B.2. The standard deviation in the distance σ_{dist} from the mean 2D location in the period under consideration is calculated and compared to a threshold. If σ_{dist} is below the threshold, immobility is detected. A filter on the mobility detection is not necessary, since σ_{dist} is already an average over a time window with length T.

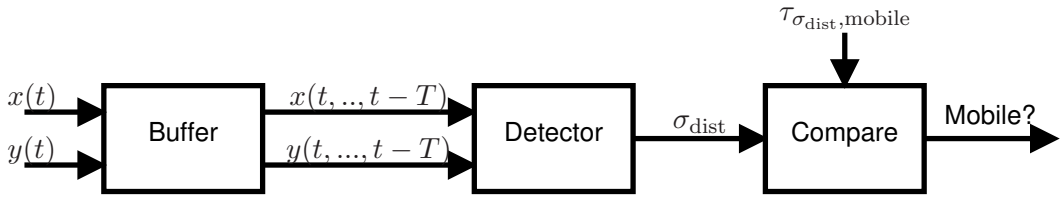


Figure B.2: *Mobility reasoning is based on the 2-dimensional position of the targets within a window of length T. The standard deviation in the distance from the mean position within the window σ_{dist} is compared to a predefined threshold to determine if the target is mobile.*

Appendix C

Software Dependencies

The software framework is written in C++ and is developed, tested, and run in Linux (Ubuntu 9.10). It requires a number of external libraries for capturing video, performing some standard computer vision calculations, rendering etc. These are all open source libraries[1], and they are listed here:

- **OpenCV:** Open Computer Vision Library provides high-level functions for capturing video, working with images, and performing computer vision related calculations.
- **S. Birchfeld et. al. KLT-tracker:** Used for feature point tracking [73].
- **OpenGL:** Is used for rendering.
- **GLUT:** Also used for rendering, and for managing render window and OpenGL context.
- **Boost:** Provides utility classes, e.g. smart pointers and thread support.

The source code of the tracker prototype is found on the attached CD-ROM together with build instructions.

[1] Some are released under the GPL (GNU General Public License), thus forcing the framework to be GPL.

Appendix D

Test Results

Table D.1 shows the measured time consumption for the different parts of the algorithm using a non-optimised setting and an optimised setting, respectively, and Table D.2 shows the time consumption for each computer during the live test of the distributed system. In Table D.3, D.4 and D.5 the complete CLEAR test results for the three trackers are shown.

Function	**Non-optimised**	**Optimised**	**Parallel**
Load frames	305.94 ms	295.38 ms	-
Detect persons	5.90 ms	6.01 ms	-
Detect 2D foreground	1876.29 ms	118.93 ms	-
Distance transform	108.09 ms	6.36 ms	-
Estimate 3D foreground	54.46 ms	5.25 ms	-
Feature points	1654.02 ms	89.70 ms	-
Track	2.52 ms	2.39 ms	-
Total	4007.24 ms	535.35 ms	372.26 ms

Table D.1: *Data from the time test performed on the development data set.*

Function	**Server**	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Capture frames	-	7.1 ms	4.3 ms	4.2 ms	4.5 ms	23.1 ms
Downscale	-	2.9 ms	2.9 ms	2.9 ms	2.9 ms	1.8 ms
Detect 2D foreground	-	24.7 ms	20.1 ms	19.9 ms	19.8 ms	14.4 ms
Distance transform	-	2.0 ms	1.6 ms	1.5 ms	1.5 ms	1.2 ms
Waiting	69.0 ms	33.6 ms	41.7 ms	42.2 ms	41.9 ms	30.4 ms
Estimate 3D foreground	1.3 ms	-	-	-	-	-
Track	0.4 ms	-	-	-	-	-
Total	70.7 ms	70.3 ms	70.7 ms	70.7 ms	70.6 ms	70.9 ms

Table D.2: *Data from the live test on the distributed system.*

Video	MOTP (mm)	Miss rate (%)	False pos. rate (%)	Mismatches	MOTA (%)
AIT_20061020_A	180	50.57	8.37	13	39.42
AIT_20061020_B	117	77.97	0.17	0	21.86
AIT_20061020B_A	122	71.84	0.86	6	26.72
AIT_20061020B_B	123	69.75	0.33	2	29.69
AIT_20061020C_A	116	60.55	3.89	4	35.05
AIT_20061020C_B	127	63.50	4.52	14	30.67
AIT_20061020D_A	121	73.74	3.24	3	22.48
AIT_20061020D_B	163	32.68	26.83	6	37.56
IBM_20060810_A	150	28.24	2.09	3	69.45
IBM_20060810_B	170	24.24	10.17	10	64.86
IBM_20060811_A	128	47.64	9.67	3	42.49
IBM_20060811_B	120	55.27	11.22	2	33.38
IBM_20060814_A	149	55.29	7.75	3	36.74
IBM_20060814_B	160	69.64	11.57	4	18.47
IBM_20060815_A	135	35.76	2.57	10	60.97
IBM_20060815_B	135	24.76	7.62	6	67.21
ITC_20060922A_A	176	69.64	2.36	1	27.91
ITC_20060922A_B	161	84.71	1.43	6	13.38
ITC_20060922B_A	157	70.73	3.47	4	25.48
ITC_20060922B_B	191	68.15	8.55	10	22.43
ITC_20060927_A	118	63.48	1.87	1	34.54
ITC_20060927_B	90	85.14	0.11	0	14.75
ITC_20060928_A	131	37.22	0.10	2	62.47
ITC_20060928_B	128	60.02	0.11	2	39.65
UKA_20060912_A	30	99.93	2.26	0	-2.19
UKA_20060912_B	0	100.00	0.78	0	-0.78
UKA_20061116_A	208	94.16	6.20	9	-1.02
UKA_20061116_B	227	91.08	3.87	6	4.64
UKA_20061120_A	141	62.27	8.05	10	28.36
UKA_20061120_B	110	85.28	1.15	1	13.49
UKA_20061207_A	166	73.19	6.14	10	19.94
UKA_20061207_B	180	82.38	5.06	16	11.67
UPC_20060620_A	148	73.54	3.83	10	21.91
UPC_20060620_B	148	82.05	5.33	14	11.70
UPC_20060713_A	120	86.89	0.59	6	12.13
UPC_20060713_B	124	74.12	1.18	4	24.44
UPC_20060720_A	106	72.84	0.90	1	26.19
UPC_20060720_B	154	70.29	3.85	18	24.67
UPC_20060720B_A	129	56.63	1.56	7	41.17
UPC_20060720B_B	139	31.48	8.59	20	58.14
Overall	137	65.42	4.70	247	29.30

Table D.3: *CLEAR test results for the foreground only tracker shown individually for all CLEAR videos together with the overall score.*

Video	MOTP (mm)	Miss rate (%)	False pos. rate (%)	Mismatches	MOTA (%)
AIT_20061020_A	201	12.80	19.26	26	64.64
AIT_20061020_B	134	75.66	1.73	3	22.36
AIT_20061020B_A	239	32.57	7.38	1	59.96
AIT_20061020B_B	220	35.60	15.40	14	47.43
AIT_20061020C_A	206	51.63	9.42	8	37.94
AIT_20061020C_B	249	34.43	25.12	15	39.04
AIT_20061020D_A	259	50.54	4.68	4	44.06
AIT_20061020D_B	209	27.80	40.00	9	27.80
IBM_20060810_A	226	3.58	53.06	24	41.58
IBM_20060810_B	257	17.03	61.62	28	19.34
IBM_20060811_A	208	23.23	70.24	20	5.15
IBM_20060811_B	268	26.17	89.46	7	-16.11
IBM_20060814_A	310	63.91	116.01	13	-80.87
IBM_20060814_B	325	52.93	58.96	6	-12.37
IBM_20060815_A	154	9.79	15.76	20	73.06
IBM_20060815_B	189	11.84	47.21	16	39.86
ITC_20060922A_A	194	9.09	18.45	6	71.91
ITC_20060922A_B	154	33.76	14.97	6	50.80
ITC_20060922B_A	163	31.61	48.95	13	18.39
ITC_20060922B_B	202	29.14	55.24	17	14.14
ITC_20060927_A	115	0.82	4.67	3	94.17
ITC_20060927_B	119	0.33	0.00	0	99.67
ITC_20060928_A	163	4.19	11.15	5	84.15
ITC_20060928_B	141	2.07	1.74	0	96.19
UKA_20060912_A	452	97.81	45.65	0	-43.46
UKA_20060912_B	458	99.87	59.68	1	-59.61
UKA_20061116_A	202	75.55	31.68	7	-7.74
UKA_20061116_B	287	66.97	21.93	13	10.19
UKA_20061120_A	281	31.13	35.36	23	30.47
UKA_20061120_B	124	25.16	18.34	1	56.41
UKA_20061207_A	274	30.78	31.86	23	35.69
UKA_20061207_B	218	49.25	16.23	29	32.91
UPC_20060620_A	175	2.60	26.03	5	71.01
UPC_20060620_B	207	41.29	59.49	12	-1.56
UPC_20060713_A	164	11.87	24.52	3	63.41
UPC_20060713_B	137	19.80	7.91	9	71.70
UPC_20060720_A	171	4.19	4.39	0	91.42
UPC_20060720_B	163	18.83	22.55	7	58.16
UPC_20060720B_A	172	2.01	18.85	3	78.87
UPC_20060720B_B	190	12.88	28.71	22	56.44
Overall	215	30.76	31.09	422	37.16

Table D.4: *CLEAR test results for the feature point only tracker shown individually for all CLEAR videos together with the overall score.*

Video	MOTP (mm)	Miss rate (%)	False pos. rate (%)	Mismatches	MOTA (%)
AIT_20061020_A	165	32.32	9.13	15	56.65
AIT_20061020_B	113	75.08	0.00	0	24.92
AIT_20061020B_A	141	66.48	2.01	7	30.84
AIT_20061020B_B	119	58.48	2.46	5	38.50
AIT_20061020C_A	124	55.28	2.26	6	41.71
AIT_20061020C_B	158	54.47	5.46	18	38.38
AIT_20061020D_A	125	72.48	0.90	2	26.26
AIT_20061020D_B	147	33.17	24.39	8	38.54
IBM_20060810_A	149	17.66	4.02	4	78.02
IBM_20060810_B	167	9.16	11.54	7	78.79
IBM_20060811_A	134	7.79	9.60	4	82.34
IBM_20060811_B	178	19.99	11.56	2	68.32
IBM_20060814_A	159	29.71	12.03	4	57.97
IBM_20060814_B	187	37.67	13.33	4	48.67
IBM_20060815_A	141	32.08	3.06	8	64.31
IBM_20060815_B	143	20.95	7.14	6	71.50
ITC_20060922A_A	136	32.36	2.45	8	64.45
ITC_20060922A_B	137	47.61	2.47	6	49.44
ITC_20060922B_A	144	56.53	6.21	3	37.02
ITC_20060922B_B	174	24.35	7.33	5	67.89
ITC_20060927_A	120	9.92	1.40	1	88.56
ITC_20060927_B	147	0.22	0.33	0	99.45
ITC_20060928_A	124	19.73	0.20	3	79.75
ITC_20060928_B	124	32.14	0.11	2	67.54
UKA_20060912_A	214	99.67	3.79	1	-3.52
UKA_20060912_B	119	99.81	1.10	0	-0.91
UKA_20061116_A	184	76.79	9.71	18	12.19
UKA_20061116_B	159	70.41	4.71	9	24.24
UKA_20061120_A	121	55.94	8.31	10	34.43
UKA_20061120_B	143	53.29	2.55	6	43.67
UKA_20061207_A	179	44.15	13.44	25	40.61
UKA_20061207_B	158	56.25	6.34	19	36.35
UPC_20060620_A	141	51.12	5.64	12	42.37
UPC_20060620_B	132	41.48	6.18	21	50.98
UPC_20060713_A	125	59.21	1.05	10	39.08
UPC_20060713_B	124	47.06	1.63	6	50.92
UPC_20060720_A	125	50.97	2.26	17	45.68
UPC_20060720_B	163	49.07	3.65	19	46.02
UPC_20060720B_A	106	25.98	2.56	7	70.81
UPC_20060720B_B	140	28.00	10.38	26	59.30
Overall	145	43.87	5.57	334	49.80

Table D.5: *CLEAR test results for the combined foreground and feature point tracker shown individually for all CLEAR videos together with the overall score.*

Appendix E

EUSIPCO 2010 Paper

Our foreground tracking system has been described in a paper entitled "Three-Dimensional Adaptive Sensing of People in a Multi-Camera Setup", which has recently been accepted for the 2010 EUSIPCO conference [28]. This is attached in this appendix. Note that this is a draft, and that the final version might include minor changes based on the reviews.

THREE-DIMENSIONAL ADAPTIVE SENSING OF PEOPLE IN A MULTI-CAMERA SETUP

M. Andersen[1], R. S. Andersen[1], N. Katsarakis[1,2], A. Pnevmatikakis[2], and Z.-H. Tan[1]

(1) Department of Electronic Systems,
Aalborg University
Fredrik Bajers Vej 7 B, 9220 Aalborg, Denmark
{martina, rasmusan, zt}@es.aau.dk

(2) Autonomic and Grid Computing,
Athens Information Technology
P.O. Box 64, Markopoulou Ave., 19002 Peania, Greece
{nkat, apne}@ait.edu.gr

ABSTRACT

Sensing the presence and state of people is of paramount importance in assistive living environments. In this paper we utilise a set of fixed, calibrated cameras to model the bodies of people directly in three dimensions. An adaptive foreground segmentation algorithm is run per camera, providing evidence to be collected in 3D body blobs. A particle filter tracker allows monitoring the modelled bodies across time, offering estimations of their state by using hot-spots and body posture. We apply our system on fall detection and activity monitoring for the elderly, addressing both emergency and cognitive care.

1. INTRODUCTION

Much interest has in resent years been directed at sensing the presence and state of people. The possible applications include surveillance [1], assistive living environments [2, 3], and human-machine interfaces [4, 5]. In this paper we build a system for tracking the position and posture of human bodies in 3D in real-time. For this, a set of 5 fixed and calibrated cameras is utilized. An adaptive foreground segmentation algorithm runs per camera. The detected 2D foreground masks for each camera are combined into one set of 3D foreground cubes using a hierarchical approach based on octrees [6]. Segmentation separates the cubes into a number of bodies, giving indications of the number and position of persons in the scene. A particle filtering tracker allows monitoring the modeled bodies in time, offering estimations of their state.

We apply our system on fall detection and activity monitoring for the elderly, addressing both emergency and cognitive care. To do so, the state estimations from the tracker are used to detect abrupt height changes and position persistence. The former are classified as "person siting down" or "person falling" and the later are compared against predefined hot-spots, to reason on possible activities like "person at dinner table", "at kitchen", or "by the TV". Also multiple human tracks indicate visits, again classified as "for dinner", "for tea", etc.

The novelty of the proposed system lies partly in the efficient combination of 2D foreground masks into 3D foreground bodies and partly in the utilization of a body measurement likelihood function within the particle filtering framework. From the 3D foreground representation, projections onto the floor plan are obtained by summing the body evidence at all heights for the given position. The resulting 2.5D representation is used to evaluate the measurement likelihood function of the proposed particle filter tracker.

This paper is organized as follows: In Section 2 the proposed tracking system is detailed. Test results of the implemented system are presented in Section 3, based on test video from the setup at the AIT. The performance of the system is evaluated and concluded upon in Section 4.

2. TRACKING SYSTEM

In this section our method for foreground detection in 3D, target management and tracking is detailed. Tracking is done using a particle filter based on an effective likelihood function, and the tracking results are interpreted to determine immobile bodies located near hot spots and the posture of each body.

2.1 Body Detection

Body detection is carried out in three stages: First foreground evidence is collected in 2D per camera. This is then combined to 3D foreground, which is finally used to model 3D bodies. Foreground detection in 2D utilises the per pixel Gaussian Mixture modelling inspired by Stauffer et. al. [7]. The performance of the algorithm at the start-up phase is improved by increasing the learning rate according to a window based approach, inspired by [8]. Robustness of foreground blobs is increased by removing shadows as in [9].

2.1.1 Modelling the space in 3D

The purpose of using several cameras for tracking is partly to be able to track in 3 dimensions, but also to filter out noise. Noise in the 2D foreground exists no matter the method used. By combining information from a number of cameras, this noise can be reduced significantly, thus increasing the robustness of the body detection.

The 2D space for each camera is spanned per pixel, i.e. discretely and, the 3D space can be spanned either discretely or as a continuous space. The latter can work e.g. either by combining information per-pixel for all cameras or by estimating the position of persons in each camera, and use epipolar geometry to estimate their positions in 3D. Combining information per-pixel for all cameras is, however, computationally expensive, and estimating the position of persons in each camera will not fully utilise the noise removal capability of having multiple cameras. Another well known approach is to model the 3D space discretely by spanning a grid of small cubes [10, 11, 12]. Information from the different cameras can then be combined for each cube, instead of for each person or region.

The novelty of our 3D body detection system is the speed improvement by using a hierarchy of cubes of different sizes, and the efficient implementation using distance transform , both described in the following.

1. Span the room with a grid of cubes on N hierarchical levels.
2. Project the centre and corners of each 3D cube on all levels to the image plane of each camera. Use the corners to determine an enclosing circle C.
3. Let the set S consist of all cubes on the highest hierarchical level.
4. **For** each cube in S:
   (a) **For** each camera:
      - Test foreground mask for foreground evidence within the enclosing circle C.
   (b) **If** enough cameras detect significant foreground:
      - **If** the cube has any children, **then** repeat 4 with S consisting of all children of the cube. **else** mark the cube as a foreground cube.

Figure 1: Recursive algorithm for converting the 2D foreground masks to a 3D grid of foreground cubes using distance transforms.

2.1.2 Hierarchical Grid Structure

Foreground in the 3D space will mostly be structured in coherent volumes that indicate the presence of persons. Large areas of the space will be completely without foreground. By dividing the space into hierarchies of cubes, these areas can be ruled out efficiently by only testing very large cubes for foreground. Only if a large cube contains foreground, is it necessary to test smaller cubes it contains (its children) to improve the resolution of the model.

An efficient way to construct hierarchies is to use octrees; that is to divide every cube on a particular hierarchical level into 8 cubes on a lower level [6]. For the test results in this paper we use a 4-level octree with the following cube widths: 40 cm, 20 cm, 10 cm and 5 cm. Only if a parent cube contains foreground are its children cubes tested for foreground. A problem for this approach arises in the border areas of the 3D space of interest, where the larger cubes might not fit very well. If a cube is partly outside the 3D space but with its centre inside the space, it is used directly. If the centre is outside the room, it cannot be tested for foreground, and must therefore be omitted. Instead, the border region is filled directly with smaller cubes (that have centres inside the 3D space). The algorithm for converting the 2D foreground masks into a grid of foreground cubes is summed up as pseudo-code in Figure 1.

For our system, all of the cameras are stationary. This causes the projection of cubes to the image plane of each camera to be identical for all frames. Therefore, the items 1 and 2 in Figure 1 can be carried out off-line, leaving 3D foreground testing as the only potentially computationally heavy part.

The hierarchical algorithm is a speed optimization of the non-hierarchical version, and has been tested to reduce the time consummation of the algorithm by around 80 % when a distance transform is used to combine the 2D foreground masks into 3D foreground as described in the following section.

2.1.3 Efficient Combination of 2D Foreground Masks into 3D Foreground

To test whether a cube projected to the image plane of a camera contains foreground, all foreground mask pixels located in that projected cube should ideally be tested. The percentage of pixels with foreground can then either be compared with a threshold for significant foreground, or used as a non-boolean indication for foreground. This is, however, computationally intensive, since pixels in the 2D foreground masks are included in many cubes, and will thus be tested many times.

The speed of the foreground testing can be increased by making certain simplifications. In many cases, the centre pixel of the projected cube indicates correctly if the cube contains foreground. To give some resistance to noise, a blurring kernel can be applied before testing. The hierarchical grid structure causes, however, the cubes to be of very different sizes, which again causes the optimal kernel size to be very different. Therefore it is chosen to apply a distance transform instead, where each pixel gets a value corresponding to the distance to the nearest pixel with foreground. After the distance transform has been applied, the centre pixel can be tested and compared to the radius of the enclosing circle, C, of the projected cube, calculated off-line. This reduces item 4a in Figure 1 to testing one pixel and comparing to the radius of C. To minimize the computation time of the distance transform an approximating 3×3 kernel is applied following the approach in [13]. This causes the calculated distances to be slightly imprecise, but also enables the time consumption to be comparable to a 3×3 blur kernel. In our implementation, the optimal values 0.95509 and 1.36930 are used for the horizontal/vertical and diagonal entries in the kernel, respectively.

It is worth noting that the results of the hierarchical and non-hierarchical algorithms when based on distance transforms are not completely identical. In some cases, perspective and camera distortion can cause the enclosing circle of a child cube (C_{child}) to contain an area not included in the enclosing circle of its parent cube (C_{parent}). If foreground is present in this area, but *not* in the rest of C_{parent}, this will cause the hierarchical structure to sort out the child cube, even though foreground exists within its enclosing circle. Minor tests have indicated that around 0.1% of the foreground cubes are sorted out for this reason. The issue could easily be avoided by using a circle slightly larger than C_{parent} for parent cubes. However, since this only happens when there is foreground inside the enclosing circle of a child cubes but *not* inside the cube itself, there is no actual reason to prevent it.

2.2 Target Management

Target management includes detection and initialization of new targets and destruction of older targets.

2.2.1 Detection of Targets

Target detection is necessary for initialization of new targets. A simple and fast approach which in many cases will work is to do 3D blob analysis of the detected foreground cubes. For our system, additional measures are taken in an attempt to utilise the typical structure of the 3D foreground. These are illustrated in Figure 2. When two individuals are positioned close together, their detections will easily be connected near

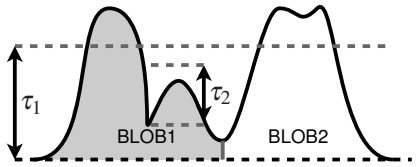


Figure 2: Top/down detection of targets.

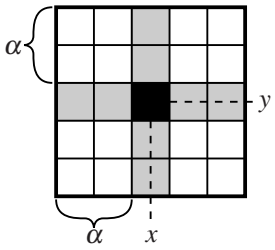


Figure 3: The state space consists of the coordinates x and y on the floor-plan, and the size variable α. The variable α can vary from 0 to the distance between the centre and the boarder of the projection map. Note that the area of a state is given as $(2\alpha+1)^2$.

the ground, e.g. because of shadows. Near their heads they will, however, often be more easily separable, partly because the heads are located farther from the ground, and partly because the head is thinner than the rest of the body. For this reason the height of the connection of two connected blobs are compared with a threshold, τ_1. The blobs are merged only if they are connected above this τ_1. To make the system robust to people sitting down or falling, an additional threshold τ_2 is used. If the height of one of the blobs relative to the connection point are below τ_2, they are always connected.

2.2.2 *Maintenance of Existing Targets*

To determine which targets that have significant supporting evidence in the measurements, the position of all targets are associated with the detected blobs using the Munkres or Hungarian algorithm [14]. Non-associated blobs are used to initialize new targets. A variable M for each target is set to 1 if it is associated, and 0 otherwise. The reliability of targets is updated using a simple IIR-filter:

$$r = r + l(M - r) \tag{1}$$

where r is the reliability and l is the learning rate. By comparing r with two thresholds, it can be determined whether the target should be trusted as an individual and (if not) if it should be destroyed. To allow new targets to become reliable relatively fast if they are associated in each frame, while preserving older targets even if they have been unassociated for some frames, the learning rate is adjusted according to the age (given as the number of consecutive frames that the target has existed). The following equation is used:

$$l = \min(l_{\max}, \tfrac{1}{\text{age}} + l_{\min}) \tag{2}$$

It may occasionally happen, that two targets follow the same individual. Therefore targets that are placed very close to one another consecutively for several frames are merged.

2.3 Tracking

The tracking algorithm used in the system is Particle Filtering (PF) [15, 16]. PF's are able to provide a numerical solution to the recursive Bayesian estimation problem when the system dynamics are not linear and/or the noise models are not Gaussian. They hence provide robust solutions to the tracking problem when the object model or the measurement likelihoods are multimodal. This is offered at the expense of additional computational complexity due to their numerical nature. We build a PF that follows the approach for motion tracking described in [16].

Foreground detection is done in 3D and thus tracking should ideally also be done in 3D. To make the tracking algorithm fast enough to allow real-time tracking, we propose what we call a 2.5D approach. All cubes are projected to the floor, and the number of cubes in each column are used to calculate likelihood. The vertical dimension thus provides some additional data for tracking, without itself being part of the target state, hence the term "half dimension". The states related to position can thus be limited to x and y. Note that the vertical position will provide little extra information for tracking since the difference in height between different persons humans are typically small.

To reduce the dimensionality of the state-space as much as possible, only a single dimension α is used to determine size. The state space $\mathbb{S}$ is therefore 3 dimensional, and the dimensions are illustrated in Figure 3.

2.3.1 *Likelihood Function*

The multi hypothetical nature of the particle filter allows tracking of non-global maxima. However, to achieve robust tracking the likelihood function must in as many situations as possible give local maxima close to the correct location and size of the persons in the scene.

To determine a good likelihood function, a number of values can be taken into consideration (where x and y have been left out as function arguments for simplicity):

Volume: A person is expected to constitute a certain volume, which can be given as a number of cubes $N(\alpha)$.

Density: A person is expected to fill most of the volume, V, inside his bounding box. This amount is expressed as:

$$F(\alpha) = \frac{N(\alpha)}{V} = \frac{N(\alpha)}{h \cdot (2\alpha+1)} \tag{3}$$

where α and h are measured in number of cubes. The height h is set to the maximum number of cubes in a single column in that area.

Derivative of density: A good state will be centred close to the centre of a person and include most of that person. This means that $F(\alpha)$ is expected to drop fast if α is increased. This is due to the fact, that most of the area around a person typically is without foreground. The change in $F(\alpha)$ can be measured by its derivative $\frac{\partial F(\alpha)}{\partial \alpha}$, which can be approximated

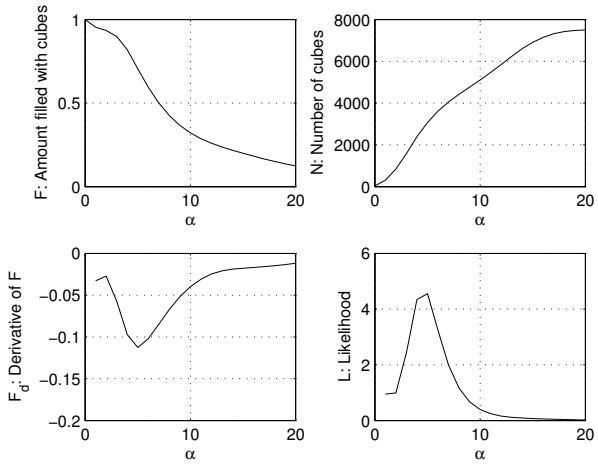


Figure 4: Different values as a function of α.

by:

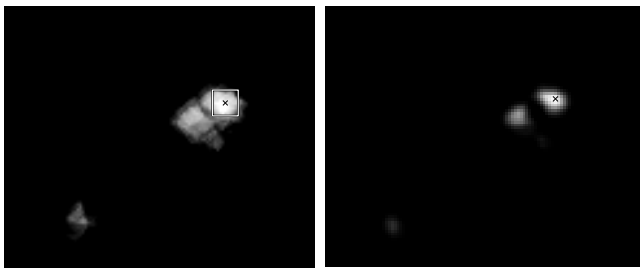
$$\begin{aligned} F_d(\alpha) &= \frac{\Delta F(\alpha)}{\Delta \alpha} \\ &= \frac{F(\alpha+k)-F(\alpha-k)}{2k} \\ &\approx \frac{1}{2kh}\left(\frac{N(\alpha+k)}{(2(\alpha+k)+1)^2}-\frac{N(\alpha-k)}{(2(\alpha-k)+1)^2}\right) \end{aligned} \quad (4)$$

where h is simplified to be the maximum height of the smaller area (which in most cases is identical to that of the larger area).

Figure 4 shows $F(\alpha)$, $F_d(\alpha)$ and $N(\alpha)$ along with the final likelihood function when α is varied in the example shown in Figure 5a. In this example, the maximum of $-F_d(\alpha)$ is located at the $\alpha = 5$, which Figure 5a proves is a good result. This is not sufficient for the likelihood function, however, since F_d reacts equally strongly on few cubes of noise and a real person. To counter this effect, the likelihood function could be chosen to $L(\alpha) = -F_d(\alpha) \cdot N(\alpha)$. $N(\alpha)$ biases towards larger areas. A problem with this approach is apparent by comparing Figure 5a and $N(\alpha)$ in Figure 4. When α grows to include both persons, $N(\alpha)$ just keeps growing. To avoid including multiple persons, $F(\alpha)$ is also included to give the final likelihood function:

$$L(\alpha) = -F(\alpha-k)^2 \cdot \sqrt{N(\alpha+k)} \cdot F_d(\alpha) \quad (5)$$

Instead of $F(\alpha)$ and $N(\alpha)$, $F(\alpha-k)$ and $N(\alpha+k)$ are used to avoid calculating $N(\alpha)$. The functions are weighted by squaring $F(\alpha-k)$ and taking the square root of $N(\alpha+k)$ to bias towards single coherent persons. Figure 5 illustrates the performance of the likelihood for a particular situation, where two persons are located close to one another. The projection of the cubes to the floor is shown in Figure 5b, where a brighter colour correspond to more cubes in the same column. Figure 5a illustrates the likelihood for all values of x and y with α fixed to 4. The set (x_i, y_i) that satisfy $(x_i, y_i) = \text{argmax}_\alpha(L(x, y, 4))$ is marked, and α is adjusted at that location to satisfy $\alpha_i = \text{argmax}_\alpha(L(x_i, y_i, \alpha))$. The state $\mathbb{S}(x_i, y_i, \alpha_i)$ is shown as a box in Figure 5a.



(a) Projection of cubes to the floor. L is optimised with respect to α with fixed x and y. The optimal value is found to be $\alpha = 5$.

(b) Likelihood $L((x, y, 4)|\vec{y})$.

Figure 5: Illustration of likelihood function.

2.3.2 *Body Posture and Hot Spots*

When noise is present in the detected 3D foreground, it is mostly located close to the floor. This is partly due to shadows and partly due to the fact, that other kinds of noise in the 2D foreground detections in most cases are filtered out by the combination of the cameras. This means, that the height of the persons can be accurately estimated by taking the maximum vertical position of the cubes located within 2D-position of the tracked target. By comparing the height with different thresholds, the body posture is identified as either standing, sitting or fallen. FIR-filters are applied to ensure robustness to noise.

People staying near hot spots are detected by analysing the movement of the targets over a predetermined period of time. The variance in the distance from the mean 2D location in the period under consideration is calculated and compared to a threshold.

3. RESULTS

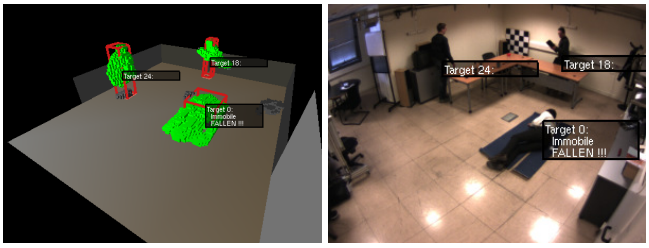
The system is tested on a setup of 5 calibrated cameras available at AIT. Four cameras are placed in the corners of a room and one camera with a fish-eye lens is placed in the ceiling. Using this setup, qualitative tests of the systems ability to detect people falling, sitting, and spending time on hot spots are carried out.

In a test sequence, up to four people move around in the area under surveillance for 6:23 min. At 3 occasions in total a person falls and at 6 occasions a person sits down. All of these events are detected correctly and there are no standing/sitting persons that are falsely detected as fallen. When a person kneels or bows he can be classified as sitting but not as fallen.

An image from one of the corner cameras from the test sequence is shown and compared with the detected foreground in Figure 6 and the complete test videos of both detected foreground and images from the camera are available on our website[1].

With a recorded set of test videos, a single dual-core 2.2 GHz computer is capable of processing a frame in approximately 1/8 second excluding the time required to load the images from the hard drive. To make the system run in real time, a distributed version has been developed, which enables the whole system to run in real-time on five 3.0 GHz

[1] http://kom.aau.dk/~zt/online/3DSensing/



(a) Detected foreground is shown as green cubes, targets are shown using wire frame boxes, and info boxes are shown for each target.

(b) Frame from one of the corner cameras with targets and info superimposed.

Figure 6: Detected 3D foreground are shown in (a) and reasoning results are shown both on top of the foreground and in (b) superimposed on a frame from a corner camera. The person lying in the floor is marked as "Fallen".

dual-core computers when the cameras are recording at 15 fps.

4. DISCUSSION AND CONCLUSION

We have in this paper presented an adaptive 3D approach for sensing people in a multi camera setup. Foreground is found per camera using a per pixel Gaussian Mixture Model and combined to a discrete 3D foreground. For this, a novel approach is used to determine the 3D foreground that combines a hierarchical octree structure with distance transforms to computational cost of the algorithm.

Our tracker is based on a 2.5D particle filter that provides fast tracking with relatively little computational cost. A likelihood function is developed that uses the amount of foreground, the density of the foreground, and the approximate derivative of the density with respect to the size of the target.

The system has been tested on a test video and is able to detect all occasions were a person falls or sits down. It should be noted that the system can fail in detecting a fall if another person is standing close by. This is, however, not critical for monitoring elderly since the fallen can get help from the other person.

It is possible for the system to run in real-time using 5 cameras at 15 fps when distributed to 5 computers. Using 5 computers might not be optimal in a real-life implementation. The major reason is that the computers use USB 2.0, whose bandwidth prevents more cameras from running simultaneously. However, when USB 3.0 gets available one computer will be able to handle a much larger data flow than our test computers.

One major limitation in our system is that tracking is based solely on foreground estimation which again is based solely on motion. Therefore, if a person stays immobile for a long duration, the associated target will eventually be lost. This can be avoided by adding additional modalities to the tracker such as colour, faces, or even sound [11, 16].

ACKNOWLEDGEMENTS

This work has been partly funded by the HERMES Specific Targeted Research Project (Contract No: FP7-216709).

REFERENCES

[1] D. Moellman and P. Matthews, "Video Analysis and Content Extraction." http://videorecognition.com/vt4ns/vace_brochure.pdf.

[2] R. Stiefelhagen, R. Bowers, and J. Fiscus, eds., *Multimodal Technologies for Perception of Humans*. Springer Berlin / Heidelberg, 2008.

[3] "Cognitive Care and Guidance for Active Aging: HERMES Technology." http://www.fp7-hermes.eu/hermes-technology.html.

[4] A. Waibel and R. Stiefelhagen, eds., *Computers in the Human Interaction Loop*. Springer-Verlang London, 2009.

[5] A. Pnevmatikakis, J. Soldatos, F. Talantzis, and L. Polymenakos, "Robust multimodal audio–visual processing for advanced context awareness in smart spaces," *Personal Ubiquitous Comput.*, vol. 13, no. 1, pp. 3–14, 2009.

[6] C. Ericson, *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. Morgan Kaufmann, January 2005.

[7] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 747–757, 2000.

[8] P. Kaewtrakulpong and R. Bowden, "An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection," in *Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems*, September 2001.

[9] T. Horprasert, D. Harwood, and L. S. Davis, "A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection," in *ICCV Frame-Rate WS*, 1999.

[10] N. Katsarakis, A. Pnevmatikakis, and M. C. Nechyba, "3D Tracking of Multiple People Using Their 2D Face Locations," in *AIAI*, pp. 365–373, 2007.

[11] N. Katsarakis, F. Talantzis, A. Pnevmatikakis, and L. Polymenakos, "The AIT 3D Audio / Visual Person Tracker for CLEAR 2007," in *Multimodal Technologies for Perception of Humans*, (Berlin, Heidelberg), pp. 35–46, Springer-Verlag, 2008.

[12] A. Koutsia, T. Semertzidis, K. Dimitropoulos, N. Grammalidis, A. Kantidakis, K. Georgouleas, and P. Violakis, "Traffic Monitoring using Multiple Cameras, Homographies and Multi-Hypothesis Tracking," in *3DTV Conference, 2007*, pp. 1–4, May 2007.

[13] G. Borgefors, "Distance transformations in digital images," *Comput. Vision Graph. Image Process.*, vol. 34, no. 3, pp. 344–371, 1986.

[14] S. S. Blackman, *Multiple-target tracking with radar applications*. Dedham, MA, Artech House, Inc., 1986.

[15] M. Isard and A. Blake, "CONDENSATION - Conditional Density Propagation for Visual Tracking," *International Journal of Computer Vision*, vol. 29, pp. 5–28, 1998.

[16] P. Perez, J. Vermaak, and A. Blake, "Data Fusion for Visual Tracking with Particles," *Proceedings of the IEEE*, vol. 92, pp. 495–513, Mar 2004.