Model-Based Background Subtraction System
Application Domain: Pedestrian Tracking

Aalborg University, Esbjerg
Department of Software and Media Technology

Mads Lindstrøm       Mohamad Zind

January 2004
Title: Model-Based Background Subtraction System  
Domain: Pedestrian Tracking  
Education: Master of Science in Information Technology  
University: Aalborg University, Esbjerg  
Courses: Computer Vision  

Group: CIS4, 9th and 10th semester  
Members: Mads Lindstrøm, Mohamad Zind  
Supervisor: Volker Krüger  

Start: February 17, 2003  
Delivery: 15 January, 2004  

Abstract: This master thesis describes theory related to background subtraction. Furthermore, it describes our analysis and implementation of the first two parts of a general model-based background subtraction system developed for object tracking. The chosen domain for this thesis is pedestrian tracking.

The first part of our system, which is about pixel-based background subtraction is presented. The reason for doing pixel background subtraction is described and the problems associated with it are also discussed. Some basic and advanced pixel-based background subtraction models are presented. Furthermore, some general useful techniques related to background subtraction are also described. The implementation of the initial part of our system is also described.

Three noise filtering techniques, which could be used for removing the noise after pixel-based background subtraction, are presented. One of them is invented by ourselves, which we call 'dynamic expanding’ filter. The two others are the mean and the median filters. We developed the 'dynamic expanding’ filter to remove the noise from the output from the initial part of our system. The results of a test of the three filters are presented.

The second part of the system, which is about learning the model of pedestrians, is presented. Furthermore, some theories, methods, and techniques needed for this task are described. These include model types, clustering methods, and distance measures. We also describe our implementation of the model learning part of the system.

The results of the first two parts of our model-based background subtraction system and the noise filtering are presented.
Mads Lindstrøm and Mohamad Zind are responsible for this project.

Mohamad Zind

Mads Lindstrøm
4 The model learning part: Pedestrian representation

4.1 Model requirements

4.2 Model representation

4.2.1 Model types

4.2.1.1 Model type choice

4.2.2 Active contour models

4.2.3 Templates and Exemplars

4.2.4 Our choice

4.3 Extraction of information from background subtracted movies

4.4 Structuring the extracted information

4.4.1 Clustering

4.4.1.1 Clustering methods

4.4.1.2 Algorithm choice

4.4.1.3 Our approach

4.4.1.4 Our choice

4.4.2 Distance measures

4.4.2.1 Normalization of exemplars

4.4.2.2 Requirements for distance measures

4.4.2.3 Hausdorff distance

4.4.2.4 Chamfer distance

4.4.2.5 Moment based distance

4.4.2.6 Our choice

4.5 Motion model

4.6 Interface to recognition part of the system

4.7 Our pedestrian model implementation and experiments

4.7.1 Pedestrian model implementation

4.7.2 Test results of our pedestrian model

4.7.2.1 Movie recording and exemplar extraction

4.7.2.2 Clustering
5 Conclusion

5.1 Summary ................................................. 91
5.2 Future work ............................................. 94
  5.2.1 Improvements for the initial segmentation part of our system .... 94
    5.2.1.1 Improvement of shadow elimination ..................... 94
    5.2.1.2 Looking at the region level ............................. 95
    5.2.1.3 Looking at the frame level ............................. 95
    5.2.1.4 Exploit camera settings ............................... 95
  5.2.2 Improvements of the model learning part of our system ........ 96
    5.2.2.1 Combination of distance measures ....................... 96
    5.2.2.2 Different distance measures at different levels in the hierarchy 96
    5.2.2.3 Textures ........................................... 96
    5.2.2.4 Hierarchy - variable number of children .............. 97
    5.2.2.5 Test of the system with other object types ........... 97
    5.2.2.6 Real-time computation / adaption of model ........... 97
  5.3 Observations ........................................... 98

A Software .................................................. 105

B Examples of the three noise filters .................................. 107
  B.1 Mean filtering example ..................................... 107
  B.2 Median filtering example ................................... 108
  B.3 Dynamic expanding filter example ............................ 108
    B.3.1 Examples of using our filter ............................. 108

C Test results of our pixel-based background subtraction algorithm .... 110
  C.1 Preliminaries ............................................ 110
  C.2 Description of our movies .................................. 111
    C.2.1 Computer generated images ............................... 111
    C.2.2 Indoor footage ........................................ 111
    C.2.3 Outdoor footage ....................................... 112
C.3 Our results and comparison

C.3.1 Cars

C.3.2 Sudden light

C.3.3 Camouage

C.3.4 Camera bump

C.3.5 Slowly moving object

C.3.6 Find bottle minus light

C.3.7 Find bottle plus light

C.3.8 Whiteboard minus light

C.3.9 Whiteboard plus light

C.3.10 Road and grass walk

C.3.11 Birds

C.3.12 Meeting

C.3.13 Grass walk

C.3.14 Path walk near

C.3.15 Wind and illumination change

D Technical implementation description of pixel-based background subtraction

D.1 Executables and prerequisites

D.1.1 Compilation of C++ programs

D.2 Usage of background subtraction programs

D.2.1 npbgs

D.2.2 ShowV4L

D.2.3 recordMovie

D.2.4 basicBGS

D.3 Source code and documentation

E Technical implementation description of model learning software

E.1 Prerequisites

E.1.1 Compilation of C++ Matlab function

E.2 Introduction to usage of the Matlab function
E.3 Source code structure and documentation ........................................ 126
E.4 Technical description of interface to the recognition part ................. 126
   E.4.1 Order.dat ................................................................. 127
   E.4.2 hierarchy.dat .......................................................... 128
   E.4.3 Silhouette****.BMP .................................................... 128
   E.4.4 Outline****.BMP ....................................................... 128
   E.4.5 MeanSilhouette****.BMP .............................................. 128
   E.4.6 maxDistances.dat .................................................... 129

F  Why Chamfer distance is not a metric ........................................ 130

G  Test movies .............................................................................. 132
List of Figures

1.1 Overview of our system ................................................. 4
1.2 Detailed overview of our system ..................................... 5

3.1 Normal distribution example ........................................... 19
3.2 Two dimensional normal distribution example ..................... 21
3.3 Screenshot of a sign .................................................. 33
3.4 Mean filtered screenshot of the sign .................................. 34
3.5 Median filtered screenshot of the sign ................................. 35
3.6 Input image pixels used in average image ........................... 36
3.7 Dynamic expanding filtered screenshot of the sign ................... 37
3.8 The reference image before and after background subtraction ... 38
3.9 Mean filtered reference image after background subtraction ....... 39
3.10 Median filtered reference image after background subtraction ... 39
3.11 Dynamic expanding filtered reference image after background subtraction .. 40

4.1 Pedestrian shapes under walking cycle ................................ 46
4.2 Human stick figure example .......................................... 47
4.3 3D human model example ............................................ 48
4.4 (B-spline) Control points example .................................... 50
4.5 Template and exemplars example ..................................... 51
4.6 Contour and silhouette example ...................................... 53
4.7 An example of our clustering approach ............................... 60
4.8 A special case for exemplar matching ................................. 62
4.9 Comparison of clustering algorithms ................................. 67
4.10 Untransformed image and distance transformed image ............... 68
4.11 An example of our rotation-normalization approach ............... 69
4.12 Centroid comparison of k-means and random clustering .......... 71
4.13 A contour vs. silhouette comparison of the distance measures ... 78
4.14 Crispness comparison of the distance measures ..................... 78
4.15 Order of exemplars ........................................... 80
4.16 Hierarchy with width 3 ......................................... 87
4.17 Hierarchy with width 4 ......................................... 88
4.18 Hierarchy with width 5 ......................................... 89
4.19 Hierarchy with width 6 ......................................... 90

B.1 3-by-3 Mean filtering example .................................. 107
B.2 3-by-3 Median filtering example ................................ 108
B.3 Theoretical example of dynamic expanding filter ................. 109

C.1 Screenshots from the 'cars' movie ................................ 114
C.2 Screenshots from the 'slowly moving object' movie ............... 116
C.3 Screenshots from the 'road and grass walk' movie ............... 118

F.1 Chamfer distance breaking triangular inequality example ........ 130
List of Tables

3.1 Examples of a few implemented systems ........................................ 27
3.2 Example of different chromatic values ........................................... 29
4.1 Advantages and disadvantages of clustering methods ....................... 59
D.1 BGS programs. ............................................................................. 121
D.2 BGS source code and implementation. .......................................... 124
E.1 Matlab packages. .......................................................................... 127
This master thesis is the result of the 10th semester project carried out by group CIS4 in the period from February 2003 to January 2004. It is the final project in the M.Sc. in software engineering, Aalborg University, Esbjerg.

It is an one year project where in the first six months (9th semester) our group consisted of four students, which were responsible for the first part of the project (pixel-based background subtraction). However, in the last six months (10th semester) the group were split in two groups, where we were responsible for the second part of the project (model learning part), and Jakob Anderson and Thomas Prehn were responsible for the third and last part (the recognition and tracking part).

We would like to thank our supervisor, Volker Krüger, for guidance and practical assistance.

Esbjerg, 15 January 2004

Mads Lindstrøm

Mohamad Zind
Chapter 1

Introduction

Even though computers become faster and faster, they are still quite dumb, since they can neither see nor even perform simple reasoning. Obviously, we are not satisfied to just use our computers as we use them today (for computation, text processing, game station/workstation etc.); instead, we expect computers to do more intelligent things that human beings can easily do. For example:

- Computers should be able to identify people by looking at their faces or even their gaits.
- Computers should be able to track movements and distinguish objects from each other.
- Computers should be able to see and learn, and think by themselves.

We believe that these examples are not impossible to achieve and someday they will become a reality. However, for the above mentioned examples to become a reality, we have to deal with a classical computer vision topic known as background subtraction. Background subtraction is a fundamental technique used in many different computer vision applications such as video surveillance, object detection and tracking, etc. The goal is to distinguish the background from the foreground in a scene. This may sound very easy, but we will see in this master thesis that it is actually a complex task, especially if the application should be a general background subtraction system, which can track objects under different situations, in indoor and outdoor environments, and in real-time.

Generally, there are two methods to do background subtraction. The simplest one is based on pixel values and is called pixel-based background subtraction. This method operate at pixel level and it cannot distinguish specific objects in a scene, such as humans, cars, etc. Pixel-based background subtraction method is more likely to classify every new object it sees in the background as foreground object. However, this simple method can be used as a part of a larger vision system. Often it can be used as a preprocessing step in order to minimize the amount of data the system needs to process later on. This is also what we intend to do in this project.

The second method is more sophisticated and it can solve some problems, which the pixel-based background subtraction method cannot. This method is called model-based background subtraction. It is able to use prior knowledge (the model) about the object(s) it is looking for.
Therefore, it can be constructed to recognize specific object(s) in a scene, such as pedestrians or any other objects we are interested in.

This thesis will however cover both methods.

1.1 Motivation and goal

The chosen domain for this project is pedestrian tracking. Because pedestrian is a class of objects with deformable shapes, research on pedestrian representation/modeling and tracking will not only produce a wide range of applications but will also shed light on general object tracking/detection and shape analysis. Therefore, the solution to our background subtraction system can be employed in a wide range of applications such as safe robot navigation, visual surveillance, human-computer interface, performance measurement for athletes and patients with disabilities, virtual reality, and figure animation.

Some example areas involving background subtraction (for pedestrian/human tracking) include:

- **Mobile robot navigation**: A mobile robot with the ability to detect people can work among people more safely and accomplish more tasks, such as following a person or giving a tour without running into people.

- **Visual surveillance**: A computer with the ability to sense people can monitor a security region to check if somebody breaks through and can report this person’s actions.

- **Human detection and tracking for video teleconferencing**: A camera plugged to a computer system, which can detect and track humans under teleconferencing can be useful for different computer supported cooperative work tasks.

- **Self-driving motor vehicles**: For a self-driving motor vehicle, it is required to detect different objects such as cats, dogs, other vehicles, people, etc., for avoiding collision with these objects.

- **Virtual reality**: In order to create a presence in a virtual space one needs first to recover the body pose in the physical space. Applications lie in interactive virtual worlds, with the Internet as possible communication medium. Such applications could be games, virtual studios, motion capture for character animations.

- **Motion analysis**: This area could eliminate the need of humans to browse through a large data sets, such as video footages for analyzing human behavior. Other applications in this area lie under personalized training systems for various sports. These systems will observe the skills and make suggestions for improvements.

The main goal of this project is the design, implementation, and validation of reliable background subtraction method, which can be used for object tracking.
1.2 Application context: Model-based background subtraction system

The theoretical goal of this project is to study and compare different background subtraction approaches, applied to both black/white and color video sequences. Furthermore, we investigate different approaches to build a model of the objects of interest. This model should be useful for recognition and tracking of the object of interest.

The practical goal of this project is to implement several of the studied algorithms, and try out approaches which could improve their performance and efficiency. Ideally we would develop a background subtraction system, which can be used for different purposes, such as pedestrian tracking. The system will be briefly described in the next section.

1.2 Application context: Model-based background subtraction system

While there are many areas to which background subtraction can be applied, the chosen domain for this master thesis is pedestrian tracking. This means that our system includes a model of a class of objects, which is a model of pedestrians. Therefore our system is called a model-based background subtraction system. Despite the specific domain of the chosen application area (pedestrian class of objects), the underlying ideas and technologies presented in this master thesis are readily applicable to the tasks described in section 1.1 and to tracking other objects. Theoretically, it should be possible to just replace the model with another model, and the system should be able to detect and track the class of objects represented in the new model.

Generally, the system is designed to subtract and track the object of interest (in this case pedestrian) from the rest of the scene in various shapes, sizes, partial occlusion, and clothing using a normal digital video camera.

The system consists of three parts as shown in figure 1.1:

- **Initial segmentation**: This is the pixel-based segmentation part of the system for finding the regions of interest. Here the foreground will be subtracted from the background based on pixel information.

- **Model learning**: This is the second part of the system, which is about representing the object of interest by developing an object model, in this context the object model will represent pedestrians.

- **Recognition**: This is the third and final part of the system, which is about recognition and tracking of the foreground (the object of interest).

The system takes the input from our pixel-based background subtraction algorithm. The input is used for both the model learning part and the recognition part of the system. However, there is a difference of the input. For the model learning part, we need “perfect” shapes (silhouettes) of the object of interest (in this case pedestrians), since the model learning should be made
of some perfect shapes (silhouettes). However, like we will see in section 3.1 there is no perfect pixel-based background subtraction algorithm, however what we want is to find good shapes with minimal errors and noise, and for this we will get the input from a camera under controlled environment. This means that the place, light, people, etc. will be chosen to get the best possible contours. We will also use our “dynamic expanding” noise filter (see section 3.3.3) to remove noise after the initial segmentation is done.

The model learning part of the system takes different inputs of different people for training and developing a pedestrian model. This part delivers the training results to the third and last part of the system, which is the recognition part.

The recognition part of the system takes two inputs. The first is a normal input from our pixel-based background subtraction algorithm (the initial segmentation part), which is not under a controlled environment. The second input is from the model learning part of the system, where then the normal silhouette will be matched with the training results of pedestrian shapes.

In figure 1.2 the system is shown in more details (only system part 1 and 2) for pedestrian tracking domain. These details (components) of the system will be described through chapter 3 and 4.
1.3 Problem definition

The next two sections will define the problem for this project. We will start by describing some challenges for the project, then we will list some important questions, which we will answer through the master thesis.

1.3.1 Challenges and solutions

Why is it difficult to develop a background subtraction system, which can distinguish pedestrians (or other specified objects) from other objects effectively? Human can easily detect and track pedestrians and estimate their locations from a single image, these problems are inherently difficult for a computer. The difficulties stem from the number of degrees of freedom in the human body, self and partial occlusion, appearance variation due to clothing, and the ambiguities in the projection of a 3D human shape onto the image plane. Shape features (the silhouettes of objects) are commonly used to overcome variable texture and illumination. However, to extract perfect and complete shapes of objects from a cluttered scene is very difficult. Although many contour extraction methods have been proposed [50], they tend to have errors and are distracted by scene clutter. To locate the joints of a person is even harder, because these are hidden by muscle, skin, and clothing. Segmentation or contour extraction errors also pose a significant challenge to accurate joint localization. Although a great number of object recognition and tracking methods have been proposed [5], none of them can handle the above challenges very well.

The context for model-based background subtraction (subtracting specific object(s)) can be seen as a general object detection problem. There are two approaches to object detection. One
approach is to search the whole image at multi-scales for objects. This is a time consuming procedure and may result in multiple responses from a single object. Another approach is to first segment foreground objects from the background and then classify each segmented object as foreground or background. Classifying only segmented objects rather than whole images significantly reduces computational complexity. Several methods such as pixel-based background subtraction, or frame differencing can be employed to separate foreground objects from the background.

Object modeling and recognition/tracking by shape is difficult mainly because:

- Shapes of a class of objects can be distorted by sensor noise or digitization.

- Shapes of a class of objects can differ due to varying viewpoint.

- Shapes of a class of objects can be deformed non-rigidly. For example, an object such as a human may have moving parts and may be flexible.

- Some parts of a shape may not be visible. These partially occluded shapes need to be classified correctly.

Pedestrian tracking is even more difficult because of shape variation due to clothing. The texture and shape variances among dresses make even the same person appear significantly different when wearing different dresses. This makes the problem more interesting to investigate, and therefore we chose pedestrians tracking as the domain of this thesis.

1.3.1.1 Our solution

Our solution was in developing a system of three parts (see section 1.2). We developed a pixel-based background subtraction algorithm, which is heavily inspired of the pixel-based background subtraction implementation made at University of Maryland (UMD-BGS) [20], to make the initial segmentation of the foreground from the background.

Then our object (pedestrian) recognition part of the system should classify a previously segmented object as pedestrian or non-pedestrian. This is an object classification problem. Usually an object classifier consists of an object model and a recognizer [63]. Therefore, our recognition/tracking part of the system was split into two parts.

In this master thesis we will focus on the initial segmentation part (the pixel-based background subtraction) and the model learning part (pedestrian representation) of the system. The recognition part will be made by another group, therefore we will only discuss the elements of the last part of the system, which are relevant for constructing the model. However, a good object model should allow the recognition of objects independent of their positions, orientations, sizes, and articulation for articulated objects. It should also accommodate variations among the instances of an object class and should be insensitive to objects with partially missing parts (occlusion).
1.3 Problem definition

1.3.2 Important questions

Several different questions will be answered in this project, which are related to the first and the second parts of the system:

The initial segmentation part of the system: pixel-based background subtraction

- What is pixel-based background subtraction?
  - Which general approaches are traditionally used for pixel-based background subtraction?
  - Which basic algorithm is used for pixel-based background subtraction?
  - What are its limits?
  - How should a good background subtraction algorithm perform?

- Which basic general models are related to and used for pixel-based background subtraction? Are they effective?

- Which advanced general models do exist for pixel-based background subtraction? Are they effective?

- Which general techniques are used in different general pixel-based background subtraction models? Are these effective?

- Which filtering techniques can be used to improve the result of background subtraction? Are these effective?

The model learning part of the system: Object representation

- Which requirements should our object model possess?

- How can we represent the object of interest (pedestrian)?
  - What model type can we use?
  - How can we describe a single stance?

- How can we extract information to construct the model from a video sequence with minimal human intervention?

- How can we structure the extracted information to make it easily and efficiently accessible?

- How can we describe the likelihood of going from one stance to another?

- How should the model be delivered to the recognition part of the system?

This master thesis presents the first and the second part of a model-based background subtraction system.

In the next section we will describe the hardware used under the project.
1.4 Hardware

During the project various hardware is going to aid the development of the model-based background subtraction implementation. While the used hardware is mostly a question about availability, software can generally be selected amongst several alternatives.

This section presents the hardware used during the project, and also describes their usage in the project.

The software used is described in appendix A. The hardware, which was available is the following:

- **Computer**: A PC equipped with a 1.8GHz Intel Pentium 4 processor and 512MB RAM is at our disposal during the project. The PC is also equipped with a Geforce 4 MX440 with 128 MB RAM and a large hard drive.

- **Cameras**: Three different cameras are available during the project. The first is a very cheap non-configurable USB web-cam (Logitech QuickCam 256), with low resolution and somewhat distorted image. This camera is not usable for the background subtraction implementation, but it might turn out handy for testing purposes. The second camera is a Sony DFW-VL500 camera with an IEEE1394 interface. This is a quality camera which is highly configurable and supports several resolutions. For a complete description of the camera specifications see [19]. This camera will be used for the model-based background subtraction implementation. The camera comes with a nice tripod, which allows steady capture. The third and final camera is a Sony DSR-PD140P camera. It can record onto a digital tape and it is therefore easy to record movies in different places. However, it is has one major drawback, namely that it can only record $12\frac{1}{2}$ frames per second.

1.5 Thesis overview

Chapter 1 introduces the project and includes the problem definition, which sets the boundaries for this project.

Chapter 2 describes some previous related work done by different people at different places.

Chapter 3 describes the theory related to the first part of the system and presents our pixel-based background subtraction algorithm.

- Section 3.1 investigates background subtraction theory. This includes a description of the basic approach, the problems associated with it, and how a good background subtraction algorithm should perform. The section also describes some basic general theories and models related to background subtraction, and some more advanced general models used for background subtraction.

- Section 3.2 presents some general techniques used for pixel-based background subtraction. The techniques are chromatic colors, incremental update, and selective update.
- Section 3.3 describes and compares some noise filtering techniques. This includes the mean, median, and our dynamic expanding filter.

- Section 3.4 describes our pixel-based background subtraction implementation and our experiments with it. This include the comparison of our implementation with the pixel-based background subtraction implementation made at the University of Maryland.

Chapter 4 describes the theory related to the second part of the system and presents our developed model of pedestrians.

- Section 4.1 describes some important requirements for developing a good object model.

- Section 4.2 presents different models types for representation of objects. It also describes and discusses some different modeling techniques, which can be used for constructing the model. Furthermore, it describes the model type and the model technique that we will use to represent the object of interest (pedestrian).

- Section 4.3 describes how we extract the exemplars from the background subtracted movies.

- Section 4.4.1 describes clustering methods. It also discusses different clustering techniques and compare them. Furthermore, it describes our clustering technique.

- Section 4.4.2 describes some popular distance measures, which will be used for clustering. A comparison of these distance measures will also be described in this section.

- Section 4.5 describes the motion model we used for calculating the probability of a stance following another.

- Section 4.6 describes very briefly the interface to the last part of the system, which is the recognition and tracking part.

- Section 4.7 describes our implementation of the model learning part of the system and our experiments with it.

Chapter 5 concludes the project.

- Section 5.1 gives a summary of our work.

- Section 5.2 gives an outlook to what is next. This includes ideas for improvements to our pixel-based background subtraction algorithm and to our pedestrian modeling.

- Section 5.3 concludes the project with our observations on background subtraction subject.
Appendix A describes the software used during the project.

Appendix B illustrates one example for each of the three noise filters we experimented with.

Appendix C describes a detailed test of the initial part (our pixel-based background subtraction algorithm) of our system.

Appendix D describes some technical aspects of our implementation of the initial part of the system.

Appendix E describes the technical aspects of the model learning implementation, and explain how to use the software found on CDROM one attached to this thesis.

Appendix F describes why Chamfer distance is not a metric.

Appendix G shows which test movies we have used for the pixel-based background subtraction test, and where they are located on CDROM one.
Chapter 2

Related Work

A huge amount of work has been done in the area of pedestrian recognition and tracking. The goal of pedestrian tracking is especially hard because of a wide range of possible pedestrian appearances and noisy backgrounds. Most of the approaches use background subtraction techniques to solve the problem. However, because the background is not stationary and because of the other problems described in section 3.1.1.2, simple pixel-based techniques of background subtraction cannot be used alone for getting the foreground objects effectively. Therefore, most of the current approaches to the task of pedestrian tracking have treated this as a model-based background subtraction task, where a model of object(s) is involved.

This means that what is needed is a background subtraction system, which uses object recognition and tracking techniques that use the cues provided by an object model.

2.1 Pixel-based background subtraction

Some approaches on pixel-based background subtraction such as [27] and [73] have been developed, for example to compensate for small, or gradual changes in the scene. However, they cannot deal with, for instance, large sudden change in the background (see section 3.1.1.2 for more problems). Independent motion detection techniques can help [46] [60], but they are difficult to develop and are not feasible for non-rigid object extraction since different body parts move differently. A common drawback with all these approaches is the assumption that the background is almost static. This limits the generalization and application of these techniques.

2.2 Model-based background subtraction

More sophisticated pedestrian recognition and tracking techniques have a two-step process: foreground detection followed by recognition step to verify if the target object is a pedestrian or not. These techniques can be called for model-based background subtraction approaches, and they are all application specific, where the model of the object(s) that must be subtracted
is known beforehand by the system. The foreground detection step is normally done by relatively simple pixel-based background subtraction algorithms. The recognition step is normally from two parts 1) model-learning and 2) classification. Generally, these approaches is about segmenting the objects of interest and nothing else.

The model-based approaches can be divided broadly into those that exploit only general properties of some specific object, and those that use explicit models of humans/pedestrians (or their projections). The advantage of the former approach is that the range of pedestrians that can be subtracted is not restricted by the set of available models. However, the use of explicit models can normally improve the reliability of segmenting and tracking pedestrians or other objects.

2.2.1 Approaches with models of object general properties

The first line of model-based approaches are relatively simple and involve shifting windows of various sizes over the image at different resolutions, extracting properties/features, such as size, color, and position, and using some standard pattern classification techniques to determine the presence of objects. This line of model-based techniques is not applicable for pedestrian recognition, this is because it is difficult to find general properties about pedestrians, since they have different shapes, height, width, and dresses in different colors. However, it has been used for recognizing of human faces, hands, and other different objects. For example, in [76] they identify human faces out of the following properties: eyes, eyebrows and mouth. In [51] they used blobs for representing different objects, and they used the blobs properties size, color, and position for multiple object tracking. For applications involving the human hand, the region of interest is typically obtained by background image subtraction or skin color detection. This is followed by some filters or morphological operations to remove noise. These extracted features are based on hand shape, movement, and/or location of the interest region, such as in [72] and [31].

2.2.2 Approaches with explicit object shape models

Explicit model-based approaches for pedestrian recognition and tracking, such as [41] and [64] try to solve the problem by recognizing and segmenting pedestrians in single images, hence taking care of both moving and stationary pedestrians. Both approaches use handcrafted human models for pedestrian recognition. The biggest and most interesting challenge to this approach is the modeling of the huge amount of variation in shapes, pose, size and appearance of the objects of interest from different directions.

For example, in [37] they extract edges and use distance measures to compare with a hierarchy of templates of human shapes. However this system have to search the whole image at multiscales for pedestrians. This is a computationally expensive procedure and single targets might give multiple responses.
2.3 Stereo vision

A powerful technique used to establish regions of interest is stereo vision. It is used in many different approaches, such as [26] and [71] in combination with texture-based pattern classification. All these techniques have a common drawback that they use multiple cameras, and hence the results depend highly on camera calibration. The setups are usually quite fragile and require calibration periodically.

2.4 Sophisticated video sensors

Lately, there has been increased interest in advanced video sensors, such as in [23] and [75], which operate outside the visible spectrum e.g. infrared, because of their reduced prices and increasing easy of use. The fact that humans appear as bright blobs in infrared videos, due to heat that their bodies emit, helps establish robust regions of interest. But pedestrians are not the only source of heat (e.g. bulbs, lamps, cars etc.), hence pattern recognition techniques are required to classify them correctly. Also, due to the low quality of the video provided by cheap infrared cameras, and the fact that different body parts appear or not, depending on the clothes, time of day, pose and other factors makes the task quite challenging.

2.5 Summary

The pixel-based background subtraction have many problems, which we will discuss later in chapter [3] therefore today the focus has shifted to the model-based background subtraction, which can provide more accurate results. However, the pixel-based background subtraction is still a useful technique that can be used as an initial step for many different applications, such as a model-based background subtraction system.

Our approach can be classified under model-based background subtraction approaches. It is mostly related to the approaches, which use explicit models of object(s) shape (see section 2.2.2). The choice of this model type will be discussed under section 4.2.1.1.
Chapter 3

The initial segmentation part: Pixel-based background subtraction

Pixel-based background subtraction is probably the oldest technique used in computer vision. Separating dynamic objects, such as people, from a relatively static background scene is one of the first steps in many computer/machine vision applications. Accurate and efficient background removal can be important for:

- Interactive systems/games [55]
- Person or object recognition and tracking [27]
- Graphical special effects [43]

In this chapter we will describe the first part of our application, which we call the initial segmentation part, which is about removing the background and keeping the foreground as input for the model learning and the recognition parts of the system (see section 1.2).

We will start by discussing some relevant theory related to pixel-based background subtraction. Furthermore, we will describe some relevant techniques for pixel-based background subtraction. Since noise is an important problem for our system, and it is in general a challenging problem for computer vision applications. We have developed a noise filter for use after the initial segmentation of the foreground, which will remove some salt/pepper noise and may also remove smaller clumps of falsely categorized pixels from the images, which will result in getting better pedestrian shapes. Therefore, we will next discuss noise filtering in general and describe and test the performance of our filter. In the end of this chapter, we will describe the implementation of the initial segmentation part of the system and our experiments with it.

3.1 Pixel-based background subtraction

There are many previous attempts at segmenting objects from a known background, many algorithms use change detection as their primary segmentation criterion. The positions and
shape of the moving object is detected from the frame difference of the learned background scene. These attempts seem to have taken one of three pixel-based background subtraction approaches, which are described below. There is however an indication that the new and more efficient model-based techniques is already beginning to replace the pixel-based background subtraction techniques. Therefore, we have chosen in this project to move further and implement a model-based background subtraction system, and chose pedestrian as the target class of objects to be detected and segmented from the background. However, the focus in this chapter is only on pixel-based background subtraction.

### 3.1 Pixel-based background subtraction approaches

The first and the simplest approach uses statistical pixel properties of the background observed over an extended period of time to construct a model of the background, and use this model to decide which pixels in an input image do not fall into the background class. The fundamental assumption of this approach is that the background is approximatively static in all aspects: geometry, reflectance, and illumination. The articles [68] and [27] describe this approach.

The second approach is based upon image motion, and only presume that the background is stationary or at most slowly varying and that the foreground objects are moving. In this approach no detailed model of the background is required. Of course, this method is only appropriate for the direct interpretation of motion; if a person stops moving, no signal remains to be processed. This method also requires constant or at most slowly varying reflectance and illumination. The systems described in [58], [21], and [30] use this approach.

The third, and the final approach we found, is based upon geometry. This approach employ special stereo hardware to compute dense depth maps in real-time. When provided with a background disparity value, this approach can perform real-time depth segmentation. The only assumption of this approach is that the geometry of the background does not vary. However, the computational burden of computing dense, robust, real-time stereo maps, requires great computational power [78]. This approach is used in [78] and [54].

The reader should have noticed now that each approach has its limits. There is not a perfect approach for everything. There is still no algorithm that performs very good background subtraction in all conditions, however some works fine as expected (under controlled environments).

A lot of people are still working on this subject and a few have developed some relatively good algorithms. In the following section we will briefly describe the most basic pixel-based background subtraction algorithm.

#### 3.1.1 Basic approach for pixel-based background subtraction

Pixel-based background subtraction basically involves subtraction of live image from a reference image. Typically, the approach consists of three basic steps:
- Threshold selection: determines appropriate threshold values used in the subtraction operation to obtain a desired detection rate. These values are found experimentally by the developers and set as a tuning parameter.

- Background modeling: where a reference image representing the background is constructed.

- Subtraction operation or pixel classification: determines whether the pixel is a part of the ordinary stationary background or if it is a foreground object.

3.1.1.2 Problems with the basic approach

Many applications have used the basic approach for pixel-based background subtraction, and did not address the following important problems [58][46]:

- Moving shadows: They can cause serious problems in pixel-based background subtraction, since they differ from the background image and are therefore identified as parts of the moving object(s) and become a part of the foreground. However, we think that this is an application specific problem. Because we might in some system make use of shadows, such as a surveillance system that can catch an intruder by his shadow, if he/she was hiding behind something.

- Slow-moving or stationary objects: These objects can also be a problem; the background image becomes corrupted by the objects themselves. For example a parked car in a street can become a part of the background. This means that a foreground object that becomes motionless cannot be distinguished from a background object that moves and then becomes motionless. This is again an application specific issue, for example, it is very important for a surveillance system to be able to track an intruder, even if he/she moves very slowly or was motionless and begins to move.

- Non-stationary background: The background can contain moving objects, for example, in an outside scene, the wind can make tree branches and bushes move. This kind of background motion causes the color pixel intensity values to vary significantly with time. For example, one pixel can be part of a leaf at one frame, a tree branch at another frame, a part of the sky on a third frame and some mixture subsequently.

- Camouflage: A foreground object’s pixel characteristics may be subsumed by the modeled background. This means, that a foreground object with some of the same colors as the background produces holes in the computed foreground object.

- Time of day: Illumination change after learning the model of the background. For example, shadows from slowly moving clouds or other variable lighting conditions cause inclusion of background elements in the computed foreground.

- Light switch: Sudden and big changes in illumination and other scene parameters alter the appearance of the background.
- Moving camera: If it is a moving camera that is used for surveillance or if the camera moves because of wind or other reasons, everything will become foreground. Small movements were addressed by only one of the approaches [20] we investigated, no one addressed large movements. However, we think it is an important problem due to the different uses of motor-driven cameras in for example surveillance and motion picture film.

An ideal background subtraction system should be able to avoid the problems listed above. Unfortunately, there does not exist any system that can deal with all the described problems. Most pixel-based background subtraction approaches look at the pixel level and some also at the region level, however very few did look at the frame level (the different levels are described next).

### 3.1.2 The different levels of pixel-based background subtraction

What we mean by the three levels we mentioned above is the following:

- The pixel level: This means that the system maintains models of the background for each pixel. Pixel level processing makes the classifications of what is foreground and what is background and also handles adaptation to changing backgrounds. The pixel level helps to avoid the following problems: 'non-stationary background’, 'time of day’, and ‘camouflage’. Pixel level processing happens at each pixel independently and ignores information observed at other pixels.

- The region level: At this level the system considers the relationship between a pixel and its neighborhood, this means that the system make use of the information observed at other pixels in the neighborhood, which might help to refine the raw classification of the pixel level. This might help avoiding the 'slow moving or stationary objects’ problem, the ‘non-stationary background’ and may help in observing the ‘moving camera’ problem.

- The frame level: This level watches for sudden changes in large parts of the frame. If a big change happens, the system can switch between some alternate background models, which can process the new observed background. This can address the ’light switch’ and the ’moving camera’ problems.

In [46] they have developed an algorithm named Wallower, which can deal with many of the problems listed above. Their success was in making an algorithm that can handle images at all the three levels. However, the results was not perfect, and they did not address the ’moving shadows’ and the ’moving camera’ problems.
3.1.3 Basic models for pixel-based background subtraction

This section starts by presenting important theories related to the task of performing pixel-based background subtraction. It contains an introduction to the normal distribution, which is a very important result from probability theory. This introduction is relevant, since pixel intensity values are often modeled as normally distributed over time. Also the univariate- and multivariate density functions are presented, with an explanation of how they relate to pixel-based background subtraction. Lastly basic models for gray scale and color background subtraction are presented.

3.1.3.1 Normal distribution

Mathematicians, statisticians and business people have long been aware, that many phenomena are not as random and difficult to understand and describe as they may seem at first glance. In many cases a phenomena conforms to the so called normal distribution, in which case it is more straightforward to deal with in terms of statistics. This is because the data of a normally distributed phenomena is predictable, and therefore also much more appealing for statistical analysis.

The predictability of normally distributed data is due to several properties. One of these properties is often referred to as the empirical rule. This property states, that when observing a normally distributed phenomena, 68 percent of the data samples will fall within one standard deviation of the mean. Likewise 95.4 percent will fall within two standard deviations, and almost all data samples (99.7 percent) will fall within three standard deviations \[63\]. Once reaching four standard deviations 99.99 percent is included.

Other important properties of a normally distributed density function are, that it is symmetrical around the mean value, and the area under the function curve always equals to one (as it is for all density functions).

These facts make the normal distribution statistically so appealing, that it is often used to model phenomena with data samples that are only roughly normally distributed. Good examples of this can be found in the area of human physiology, and more precisely in the heights of an adult population. This physiological feature can reasonably be modeled as normally distributed. In order to get convinced (if needed), one can measure the heights of a crowd of people, and arrange the measured results in a histogram. Given the population is big enough, the distribution will be almost symmetrical around the mean height. The distribution of adult heights is of cause only approximately normally distributed. That is, the percentage of people x cm shorter than the average height, will not always precisely equal the percentage of people x cm taller than the average. Likewise only approximately 95 percent of the adult heights fall within two standard deviations of the mean.

The normal distribution can be modeled using a univariate density function, which can be thought of as a "bell-shaped" curve. For a single measured feature (one dimension) the function is defined by the following equation:
Equation (3.1) has two numerical parameters, which is the mean value of the observed values ($\mu$), and the standard deviation ($\sigma$) respectively. The mean value constitutes the peak of the density function, while the standard deviation determines the actual shape of the density function.

The properties of the normal distribution is also usable in the area of computer vision. When trying to perform a robust and effective pixel-based background subtraction in a sequence of video frames, a good model of what constitutes the background is essential. The intensity value of a pixel can generally be modeled reasonably well as normally distributed over time.

An example density function for the distribution of the intensity values of a single pixel is shown in figure 3.1. In this example the mean value of the pixel is 130 ($\mu = 130$) and the standard deviation is 30 ($\sigma = 30$).

The measured intensity could be either a gray-scale value, or one of the three color components of the RGB color space (or some other relevant color space). In both cases the intensity value is known to be in the range 0-255, that is, a single byte is used to represent the intensity value. It is assumed that the likelihood of a pixel belonging to the background of a stationary scene, is distributed evenly around the mean pixel intensity value $\mu$. The validity of this assumption naturally depends on the specific scenario in which the pixel-based background subtraction is to be performed. Imagine a dark room where all the background pixel values are distributed inside a low intensity range. Also imagine, that in this room a door is sometimes opened a few seconds, as a person enters or exists the room. As the door is being opened, the room is
The initial segmentation part: Pixel-based background subtraction

gradually illuminated by light from the neighboring room. Because there is no corresponding decrease in intensity value, an intensity-histogram of the background pixels would not be symmetrical. In this specific scenario the background would not be modeled particular well as normally distributed. Another issue which might break the full symmetry of the pixel intensity distribution is camera related. A camera is only able to differentiate pixel intensities which are inside some limited range. The size of this range depend on the specifications of the camera. If the captured scenario contains pixel intensity values outside the range of the camera, they will be registered as border values (0 or 255). Such a situation will result in a unrealistic high number of border value occurrences in a histogram. Pixel intensity values which are actually quite different end up being registered as border values. Resolving this issue is a matter of choosing a camera with the right specifications, and configuring it appropriately for use in the scenario at hand.

In general however, the univariate density function is a fair model choice for pixel-based background subtraction without prior knowledge of the scenario and specific subtraction task, when assuming static background.

Multivariate density: Often more than one sample feature is available for measurement. As an example of this, fish moving along on a conveyor belt is typically categorized by combining several measurements of the fish. Examples of such measurements, which are useful for differentiation, is the height, length and lightness of the fish. By combining the right measurements, fish can be categorized with a fairly low error rate.

In general, a samples with \( d \) measured features can be modeled using the multivariate density function given in equation 3.2.

\[
p(x) = \frac{1}{(2\pi)^{d/2}\Sigma^{1/2}} \exp \left[ -\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right]
\]  

(3.2)

In this equation \( x \) is a \( d \)-component column vector (the features), and \( \mu \) is a \( d \)-component mean vector. \( \Sigma \) is a \( d \)-by-\( d \) covariance matrix, and \( |\Sigma| \) and \( \Sigma^{-1} \) are its determinant and inverse respectively. We denote the transpose of \( x - \mu \) as \( (x - \mu)^T \).

The multivariate density function is also relevant as a model for color pixel-based background subtraction. Here three different measurements (intensity values) exist for each sample (pixel). These are typically the color components R, G and B, but can also be the components of other color spaces. In this case the multivariate density function is used with an input vector of size 3 (\( d=3 \)).

Because calculating a covariance matrix is computational expensive, many pixel-based background subtraction approaches assume independence between the available color components. A good example of an approach making this assumption can be found in [20]. Figure 3.2 displays the combined normal distribution of two color components (with different mean and variance values) under the assumption that they are completely independent. The probability densities for each of the color component intensity values can then simply be multiplied. It
is important to realize, that a color independence assumption is generally quite unrealistic. In most scenarios the measured values of the color components will be varying together to some degree. For example this is the case when the lighting conditions of the scenario changes. Then the intensity of the color components will increase or decrease together.

**Figure 3.2: Two dimensional normal distribution example.**

### 3.1.3.2 Basic probabilistic models

This section presents basic pixel-based background subtraction models for gray scale and color input.

These models are rather simplistic, since the background is assumed to be static. All pixels are simply modeled using a single univariate density function (gray scale), or a multivariate density function (color). In the gray scale case, a new pixel captured from the camera is compared to an average of the same pixel calculated from previous image frames. The pixel intensity values are assumed to be normally distributed. If a pixel is within a certain number of standard deviations from the mean value, the pixel is marked as belonging to the background. Otherwise the pixel is too far away from the model, and will therefore be modeled as foreground. The same procedure is used in the color case, only this time based on the multivariate density function. Here the standard deviation is replaced with a threshold, which is used to determine the status of new pixels. The threshold is a tuning parameter. In both models, a binary image can be constructed with white as foreground and black as background (intensities 255 and 0).

The mean and variance (covariance) values in both the univariate and multivariate density function has to be updated between images frames. This is not computational expensive in the univariate case, since the values can be updated iteratively. As mentioned earlier it is quite expensive to update the covariance matrix of the multivariate density function iteratively.
In the implementation, both models can optionally be expanded to incorporate exponential forgetting. This enables the user to control how long past pixel values influences the model of the background. Exponential forgetting is calculated using equation \[3.3\]

\[
B(x,y,t) = (1 - \alpha)B(x,y,t-1) + \alpha I(x,y,t)
\]

In equation \[3.3\] the pixel intensity value of the background at image coordinate \((x,y)\) in time \(t\) is written \(B(x,y,t)\). The pixel intensity value of the current image at the same coordinates is written \(I(x,y,t)\). The higher the value of \(\alpha\), the faster the past background model is forgotten and vice versa.

The obvious problem with the pixel-based background subtraction models is, that they do a poor job at handling non-stationary backgrounds. A pixel is likely to be falsely marked as foreground, even though it is really just belonging to more than one background category. Also the models do not incorporate more advanced features such as shadow removal. These issues are addressed in the next section.

### 3.1.4 Advanced pixel-based background subtraction models

In the last section we described some basic models for pixel-based background subtraction. All these models had in common that they could not handle non-stationary backgrounds. In this section we will describe models which do handle non-stationary backgrounds. These models are still not perfect, but they are better than the ones described in the previous section. However we will compare the models and describe their advantages and disadvantages in different situations and environments. We will also compare the models to the ones described in the previous section.

As described in section \[3.1.1.2\] there are several difficulties when doing pixel-based background subtraction. One of these is that the background is not always stationary. This means that one very difficult part of background subtraction is the maintenance of the background model. Therefore the non-stationary background problem is among the most important problems to solve. One large cause of non-stationary backgrounds is the weather. If the weather is windy some objects will move, for example trees. This will mean that the same pixel, at different times, could be showing part of a branch, the front of a leaf, and the back of a leaf. In this section we will examine different ways of handling this problem.

One way to classify approaches in solving this problem, is to divide between those approaches which look at each pixel in isolation, and those who take a neighborhood of pixels into account. In the following we will discuss each of these approaches.

#### 3.1.4.1 Neighborhood of pixels

The reason for looking at a neighborhood of pixels is due to the assumption, that often small movements in the background will occur. For example a branch may move. The branch will
now occupy another pixel than before. This new pixel is likely to be in the vicinity of the old pixel. Therefore when testing whether a pixel, in a newly observed image, belongs to the background or foreground, one could not only look at the same location in the background model, but also at locations nearby. If any of these pixels fit the newly observed one, we will deem it to belong to the background.

There are however some problems with this model. Firstly only small displacements are discovered, as we only look at the vicinity. We see this problem as negligible, as there are limits to how far background objects will move. However it may be a problem if the weather is very windy.

The second problem is that true foreground pixels, may accidently be deemed background. However if a pixel is likely to have moved, other pixels, connected to this pixel, are also likely to have moved. Therefore one can look at connected pixels, if these also seem to have moved (in the same direction) it is likely to be true movement. However if they have not, it is not likely to be true movement.

Thirdly, not all pixels may come from the surrounding neighborhood. Pixels could come from previously obscured parts of the frame. For example an object could have been behind a wall. Another possibility is that an object could have two sides where each side is shown at different times, like a leaf could show its front and backside. This model does not take the last problem into consideration, however the single pixel models do.

This technique for pixel-based background subtraction is described in more detail in [20].

3.1.4.2 Single pixel solutions

Another way to solve the problem of non-stationary backgrounds, is to use more traditional statistical approaches. We will describe both non-parametric and mixture model approaches. Mixture models uses multiple normal distributions to model the same pixels, whereas a non-parametric approach uses each pixel in each frame as its own normal distributions (explained below).

In both approaches a set of pixels from the same location are observed at different times. These pixels are assumed to model the background.

**Mixture model:** In this model an algorithm is used to figure out which pixels belongs to which normal distributions. The number of normal distributions is normally set as a parameter from the beginning, however some algorithms also estimate this number [3].

These normal distributions are then used to form a combined probability estimate. The equation for a multivariate mixture model is shown below in [3.4]

\[
p(x) = \sum_{j=1}^{K} \frac{w_j}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} \exp \left[ -\frac{1}{2} (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j) \right]
\]  

(3.4)
Where $K$ is the number of distributions, and $d$ the number of color channels. $w_j$ is a weight used to adjust the influence of each distribution, $\mu_j$ is the mean, and $\Sigma_j$ is the covariance for the $j$th distribution. $x$ is the color vector that is to be compared to the background model.

If we compare this with equation 3.2 for the multivariate model described in section 3.1.3.1 we see that the only difference is the summation sign at the beginning and the weights.

A mixture model could also be used for single variate density functions as shown below.

$$p(x) = \sum_{j=1}^{K} \frac{w_j}{\sigma \sqrt{2\pi}} \exp \left[ -\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2 \right] \quad (3.5)$$

Again $K$ is the number of distributions, $w_j$ is the weight, $\mu$ is the mean, $\sigma$ the standard deviation, and $x$ is a gray scale value, which is to be compared to the background model.

The two most common approaches to estimate which pixels belong to which distribution are expectation maximization and K-means clustering (see section 4.4.1.3 for description of the K-means algorithm). These methods are described in [2].

**Non-parametric model:** In [20] a non-parametric method is described. This model does not estimate the number of distributions, nor does it use the number of distributions as a parameter. It simply uses the value of each pixel as the mean parameter for a normal distribution. That is, there are as many normal distributions, for each pixel, as there are frames in the background model. New pixels are compared to each normal distribution, for the pixel’s location, and the density values are added to the total. If the combined value exceeds a threshold, it is considered to be a background pixel. Otherwise it is deemed a foreground pixel. For clarity a formula is shown below:

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (x-x_i)^T \Sigma^{-1} (x-x_i) \right] \quad (3.6)$$

Where $n$ is the number of train frames, $x_i$ is the pixel in train frame $i$ and rest of the variables are as described in section 3.1.3.1.

In [20] independence between the different color channels are assumed and the function reduces to:

$$p(x) = \frac{1}{n} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi} \sigma_j} e^{-\frac{1}{2} \left( \frac{x_j-x_{ij}}{\sigma_j} \right)^2} \quad (3.7)$$

Where $n$ is the number of train frames, $\sigma_j$ is the standard deviation for the $j$’th dimension, $x_j$ is the $j$’th dimension for $x$ and $x_{ij}$ is the $j$’th dimension of the pixel in train frame $i$. 

---

The initial segmentation part: Pixel-based background subtraction
3.1 Pixel-based background subtraction

The standard deviation for the normal distribution needs to be estimated, but the normal approach shown in equation 3.8 cannot be used.

\[
\sigma = \sqrt{\frac{\sum_{i=0}^{n} (x_i - \mu)^2}{n}}
\]  

(3.8)

The reason for this is that we do not know the number of distributions. If we use elements from different distributions, the standard deviation will be too high, as the differences between distributions are higher than within distributions. However, a way to estimate the standard deviation is to use the median of the differences between pixels taken from consecutive frames. It is assumed that the likelihood that these pixels belong to the same distribution is high, as these pixels are captured close in time. A small number of these differences will be high, as the pixels will be from different distributions. However, this will not be a problem as the median and not the mean is used.

**Problems with single pixel model:** The single pixel models work when all objects that can occupy a location has been incorporated in the background model. However, if an object rarely occupies a location it may not become part of the background model. This could happen when larger than ordinary movements happen, for example when a particular large blow of wind occurs. In this case the single pixel models would falsely classify the location as foreground. The above described neighborhood solution does not have this problem.

**Non-parametric versus mixture model:** In general it is best to use non-parametric methods when the data sample is small (e.g. \( n < 100 \)) and we do not know the distribution type (e.g. a normal distribution). Here normal parametric methods may become unreliable as these expect the data to be of a certain distribution type and they need a large number of samples to be robust to randomness [15]. Though, parametric (mixture) models are more precise than their non-parametric counterparts when the data quality is good, and are therefore preferred in these situations:

“On the other hand, nonparametric statistics are less statistically powerful (sensitive) than their parametric counterparts.” [15]

We are not sure of the distribution type, though it does not seem to far fetched to expect it to be normal. However we do not know this. Above 100 samples are by no means unrealistic, though some use as little as 50 samples [20]. Therefore it is unclear if it is best to use a parametric or non-parametric approach.

3.1.4.3 Combination of multi pixel and neighborhood model

As both single pixel and neighborhood models have advantages and disadvantages, we think it is best to use a combination of both models. This is exactly what is done in [20]. This gets
us both the advantages of single pixel models and neighborhood models. We will be able to recognize pixels as background, both if they have moved from a nearby location, and if the same pixel over time has different intensity values.

### 3.1.4.4 Related literature

In table 3.1, we have listed some of the most known approaches we have investigated, which use the models we described in this and the previous section. The table shows which system use which model and which problems each system addresses, of the problems we have described under section 3.1.1.2. It also shows which levels (see section 3.1.2) each system uses to perform the background segmentation.

The table shows that the first and the last systems have addressed many problems. The first system operates at two levels, the pixel and the region levels. It uses the non-parametric model, while the last system in the table operates at three levels: the pixel, region, and the frame levels.

It uses the probabilistic model at the first level and some other techniques, which we have not described, at the second and the third levels. The region level allows the last system to address the 'non-stationary background’ problem, where the frame level helps in addressing the 'light switch’ problem, due to the fact that the system considers big sudden changes in the frames. However, the last approach do not address the 'moving shadows’ problem while the first approach does.

The worst system is the one which uses the basic pixel-based background subtraction model, which we have described under the section 3.1.1.1. They use some techniques that can address, to some degree, the 'moving shadows’ and the 'time of day’ problems, however it seems that the system performs badly.

The second and the third systems performs better than the worst system, however they do not match the first and the last systems. We think that it is because they just perform at one level, which is of course the pixel level.

The table shows that even if one use the basic probabilistic model for pixel-based background subtraction, one can still address many different problems by using additional techniques. That what makes the comparison between the models very hard.

### 3.2 Techniques for pixel-based background subtraction

In the following sections we will present a few techniques used in pixel-based background subtraction. These techniques can be used in most if not all of the different models used to make pixel-based background subtraction, see section 3.1.3 and 3.1.4 for descriptions of different models. The techniques presented below are chromatic colors, incremental update, selective update, and fast pixel estimation.
### 3.2 Techniques for pixel-based background subtraction

<table>
<thead>
<tr>
<th>Systems</th>
<th>Used model</th>
<th>Adressed problems</th>
<th>Levels</th>
</tr>
</thead>
</table>
| [20] | Non-parametric model | - Non-stationary background  
- Moving shadows  
- Slow-moving or stationary objects  
- Camouflage  
- Time of day | Pixel and region levels |
| [73] | Mixture model | - Moving shadows  
- Non-stationary background  
- Camouflage  
- Time of day | Pixel level |
| [34] | Probabilistic & Mixture model | - Moving shadow  
- Slow-moving or stationary objects  
- Non-stationary background | Pixel level |
| [68] | Basic pixel-based background subtraction model | - Moving shadows  
- Time of day | Pixel level |
| [46] | Basic probabilistic model & additional techniques not explained in our thesis | - Slow moving or stationary objects  
- Non-stationary background  
- Camouflage  
- Time of day  
- Light switch | Pixel level, region, and frame level |

[20]: Non-parametric Model for Background Subtraction  
[73]: Adaptive background mixture models for real-time tracking  
[34]: Image segmentation in video sequences: A probabilistic approach  
[68]: A statistical Approach for Real-Time Robust Background Subtraction and Shadow Detection  
[46]: Principles and practice of background maintenance

Table 3.1: Implemented systems and the models they use and which problems they address.
3.2.1 Chromatic colors

Changing illumination can have great influence when judging whether a pixel belongs to the foreground or the background, as higher or lower intensity will affect all the different color channels. This can be very unfortunate, as a cloud which gets in front of the sun can turn all pixels into foreground pixels. Furthermore foreground objects casts shadows. The shadowed area will get a lower intensity and therefore turn into foreground. Likewise for higher light intensity, for example if someone turns on the light, the whole frame can get a higher intensity and therefore become foreground.

To avoid this chromatic colors can be used, which are independent of light intensity. These are calculated by:

\[ r = \frac{R}{R+G+B} \quad (3.9) \]

\[ g = \frac{G}{R+G+B} \quad (3.10) \]

\[ b = \frac{B}{R+G+B} \quad (3.11) \]

That is, if all color channels gain or lower their intensity by the same amount (percentage wise), the chromatic colors will stay the same. This technique are used by many different pixel-based background subtraction implementations, such as [62][20].

This technique is not perfect, and in the next section we will describe some of its shortcomings.

3.2.1.1 Problems with chromatic colors

Normally color values are represented between 0 and 255 (both inclusive). Some color intensities are outside the range covered by the camera, and are therefore incorrectly assigned a border value (0 or 255). Therefore, pixels which should have been assigned a value below 0 or above 255 will make the chromatic colors stop functioning properly. To illustrate the problem we have made an example below, with a pixel which has a green intensity value of 300. We lower the intensity with 20% to illustrate the problem:

As it can be seen in the tables in figure 3.2 the chromatic values are not the same. The reason for this is that the green value was above 255 in the first table. When it drops in value, it does not seem as significantly to the computer as the other colors. The reason for this is that the computer did not represent the green color as 300 before the intensity drop, only 255. It should be noted that this problem is not only a problem with chromatic colors, as it is a general problem with cameras (the same problem also exist with non chromatic colors).
### 3.2 Techniques for pixel-based background subtraction

<table>
<thead>
<tr>
<th>Real intensity</th>
<th>Computer intensity</th>
<th>Chromatic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red 120</td>
<td>120</td>
<td>120/455=0.26</td>
</tr>
<tr>
<td>Green 300</td>
<td>255</td>
<td>255/455=0.56</td>
</tr>
<tr>
<td>Blue 80</td>
<td>80</td>
<td>80/455=0.18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real intensity</th>
<th>Computer intensity</th>
<th>Chromatic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red 120*0.8=96</td>
<td>96</td>
<td>96/404=0.24</td>
</tr>
<tr>
<td>Green 300*0.8=244</td>
<td>244</td>
<td>244/404=0.60</td>
</tr>
<tr>
<td>Blue 80*0.8=64</td>
<td>64</td>
<td>64/404=0.16</td>
</tr>
</tbody>
</table>

Table 3.2: Different chromatic values.

Another problem with chromatic values are that they are sometimes far from accurate, even with no presence of borderline values. The real reason for this is unknown to us, though our practical experience shows us this.

For the reasons mentioned above, chromatic colors are not a perfect solution to varying light intensity. However, the technique is usable to a certain degree and, as mentioned above, it is used by many pixel-based background subtraction implementations.

#### 3.2.2 Incremental update

When doing background subtraction it is desirable that the background subtraction models adapts to changes in the background. For example, when a new object has been in the scene for some time it can be desirable to begin to consider it part of the background and no longer foreground. Time of day will also affect the lighting conditions and it will be advantageous if the background adapts to this. In outdoor scenes changing weather conditions, such as heavier or weaker winds, should also change the background model. Therefore, it is an advantage to not just have a stable set of background frames, but instead to update the background while performing background subtraction. However, incremental update also has problems. Some of these are described below.

#### 3.2.2.1 Problems with incremental update

If the background adapts too quickly we will risk that foreground objects falsely become background. This could happen when an object comes to a complete stop for some time or when an object moves very slowly. Therefore we do not want the background to adapt too quickly.

On the other hand if the background adapts too slowly, changes will not be reflected in due time and the result will be false foreground.
As seen above, both too fast and too slow adaptions are a problem and it is therefore important to strike a good balance between the need for adaption and not adapting foreground objects too quickly. Another way to deal with this problem, known as selective update, is presented below.

### 3.2.3 Selective update

Instead of blindly updating the background, we can limit ourselves to only updating the parts of the screen that are deemed background. In this way foreground objects, which are temporarily stopped will not falsely become part of the background model.

However, we may want to limit our selective update for two reasons:

- If pixels have falsely been judged foreground, we will keep seeing them as such if we use selective update. It may be acceptable that this occurs for some time, however it is not acceptable that it occurs in the long run.

- If an object is recorded by the camera over an extended period of time, it will often be desirable to make it part of the background. However, with the selective update it can stay part of the foreground forever.

We can limit the selective update by only performing it in some percentage of the new frames. For example we could perform selective update in nine out of ten newly recorded frames, while blindly updating all pixels in the single remaining frame.

Other models use more elaborate schemes to avoid the problems of updating the background, see [20].

### 3.2.4 Fast pixel estimation

Estimating every pixel to see if it is foreground or not is a very expensive task. Therefore it is an advantage to skip the estimation of some regions if it is possible. In the following we will explain how this is done in our implementation.

The basic idea is that a foreground object must come from somewhere near its current location. Therefore if the neighbor regions of a region in an image do not contain a foreground object, it is likely that the region itself will also not contain a foreground object in the next image. How this idea is implemented is explained in more detail below.

As the first step when estimating a new frame, before normal estimation, the fast pixel estimation is performed. This is first done by calculating an average image as explained in section 3.3.3. The average image is calculated from the result of the image estimated just before the current image. The square of size N is a tuning parameter, which we normally set to four.

Secondly, for each square in the average image we look at the square itself and all its neighbors. If they are all foreground we need to estimate the center as we would normally estimate a pixel
(without 'fast pixel estimation'). The coordinates of the center are calculated (relative to the current square) as shown in equation \[3.12\]

\[
\text{Center}_X = \text{Center}_Y = \left\lfloor \frac{N}{2} \right\rfloor
\]

If the center pixel value is also foreground we deem the hole square foreground.

In case all the squares are background, and the center pixel is background, the hole square is deemed background. If none of the above is true we just estimate all the pixels in the square as normal.

Fast pixel estimation is not without problems, these are explained in the next section.

**3.2.4.1 Problems with fast pixel estimation**

If objects move very fast they may skip several regions. This will result in objects not coming from a neighbor region and the basic idea stops working properly. We will then have to rely on the estimation of the center pixel. This is however not a perfect solution, as this estimation is build upon statistics and as such is insecure. Since we only estimate a single pixel it is not very unlikely that we will get wrong results.

Furthermore the foreground object may come from a region that was previously invisible. For example the object could come from behind another object or it could come from outside the camera view (for example from the left of a leftmost region). Again we will have to rely on the estimation of the center pixel.

To remedy the above mentioned problems we could stop estimating borderline regions, and we could do more than one check estimation (as with the center pixel). However this will lead to a lot more pixel estimation and therefore a less efficient implementation.

**3.3 Noise filtering**

With any image, whether it is received by a camera or perceived by the human eye itself, there will more than likely be distortion or 'noise' of some sort. In some systems such as those found at production lines, this is not usually a problem as the conditions for imaging are carefully controlled with special lighting. However in other applications, where conditions are not so tightly controlled, images will inevitably be distorted. As humans we give no thought to filtering out noise to make sense of our surroundings. On a foggy day for example, our vision will certainly be impaired, however what we can see we can still understand - driving on a motorway for example and seeing the hazy image of car tail-lights in front of us, we instinctively know to react to the situation by slowing down. We still recognize the image as another car even though we can not see it properly.
With human eyesight we can still make sense of our surroundings despite factors distorting our view. For machine/computer vision however, distortion often needs to be removed or ‘filtered’, before anything can be done to process the image itself.

The definition of noise in computer vision is:

“...noise may refer to any entity, in images, data or intermediate, that is not interesting for the purposes of the main computation.” [65]

Here it is important to make clear that noise filtering is a large topic, which is very classic in both the signal and image processing fields [65]. Different types of noise are countered by different techniques, depending on the nature of the noise and it’s characteristics. However, in the following sections we will only describe the simplest and the most used filters in computer vision applications. We will also describe a simple home made filter, which we call “dynamic expanding” filter.

There is however a major side effect connected to filtering. Filtering removes noise, but may also remove important data from the images. Some filters remove edges, while other may make some parts in the image more round. Therefore, one always has to choose and configure the right filter for the specific scenario at hand.

Removal of noise is normally done by a series of mathematical functions or algorithms such as the ‘median’ filter.

Once distortion has been removed, we can go on to processing the image. For instance, we can extract features from the image for later use or just use the filtered images as input for the background subtraction algorithm.

Noise in our case is everything that is not pedestrain, since the system should only recognize pedestrians and nothing else. It is however very difficult to develop a filter, which can recognize a class of objects (in our case pedestrian) and remove everything else, without prior knowledge about the the class of objects. This problem is about object recognition/classification rather than noise filtering (see section [1.3.1] and chapter [4]. Therefore the noise filters, which will be described in the next sections, are only usefull for removing salt/pepper noise and smaller clumps of falsely detected pixels. In the following sections we will describe and compare two simple and popular salt/pepper noise filters, which are used in both image processing and computer vision fields. These filters are named mean and median filters. Next we will describe our “dynamic expanding” filter, which is the one used by our pixel-based background subtraction algorithm. The filter is home made and have been given the name “dynamic expanding” filter by ourselves. We will then compare the two popular filters (mean and median filters) with our filter (dynamic expanding filter).

To make it possible for the reader to visually compare the described filters, figure [3.3] of a sign will be used as reference. This screenshot contains several elements, which should make it valuable for comparison.
3.3 Noise filtering

3.3.1 Mean filtering

Mean filtering is used to reduce noise in images. It is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It replaces the center value in the window (for example a 3-by-3 pixel matrix of an image) with the average (mean) of all the pixel values in the window. Larger windows of the neighborhood (e.g. 5-by-5 squares) can be used for more severe smoothing.

The window is usually square but can be any shape. An example of mean filtering can be seen in appendix B.

Figure 3.4 shows the result of applying the mean filter on the reference screenshot of the sign. Notice how the pixels around the edges of the sign letters are blurred compared to the reference screenshot. Also notice the differences in the grassy background.

3.3.2 Median filtering

Like the mean filter, the median filter is normally used to reduce noise in an image. However, it often does a better job than the mean filter of preserving useful details in the image such as keeping the shape of the edges intact.

The median filter considers each pixel in the image in turn, by looking at its nearby neighbors, and deciding whether or not it is representative of its surroundings. It is actually equal to the mean filter in this respect. However, instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order, and then replacing the pixel being considered with the pixel value of the middle element.
of the ordered list. If the neighborhood contains an even number of pixels, the average of the two middle pixel values is used. An example of median filtering is shown in appendix B.

The following illustrates two advantages of median filter over the mean filter [14] [65]:

- The median is a more robust than the mean as a single very unrepresentative pixel in a neighborhood will not affect the median value significantly.

- Since the median value must actually be the value of one of the pixels in the neighborhood (this is only if the neighborhood contains an odd number of pixels), the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter. A median filter will remove noise spikes from the image without significantly blurring the edges. Very fine detail, such as sharp corners, will be removed by the mean filter.

The screenshot in figure 3.5 illustrates the effects of applying the median filter on the reference sign. Notice how the edges of the letters have become round and curved. Also notice that the images is not as blurred as the mean filtered screenshot.

For the reasons described above, we have implemented the median filter in C++ and tested it with the basic pixel-based background subtraction algorithm (see section 3.1.1.1), which is also implemented in C++, see appendix D.2.4.

We also tried using the mean filter in Matlab, but due to the disadvantages listed above, we have not tested it with the pixel-based basic background subtraction algorithm.
3.3 Noise filtering

Figure 3.5: Median filtered screenshot of the sign.

3.3.3 Dynamic expanding filter

The purpose of this filter is to remove noise from the image, without harming the foreground object by removing parts of it or smoothing the objects edges too much. To ensure this, the filter starts with a small square size (see below for explanation of square size) in which choices can be made with higher confidence. It then uses larger and larger square sizes, where it becomes less and less confident about its choices. However, the filter will then make the less confident choices based on a good foundation, as the smaller square sizes has already been used.

Compared to the mean and median filters, this filter is conservative in its decision makings. That is, where the mean and median filter almost always changes the image, the dynamic expanding filter requires larger evidence to change a pixel. Therefore we can use larger square sizes without risking heavy damage to the foreground object.

The main disadvantage of the filter is its complexity, which makes it computational expensive compared to the mean and median filters. Further disadvantages are described below in section 3.3.4.

3.3.3.1 How it works

The filter works by first creating a smaller average image from the input image.

The average image is created by moving a N-by-N sized square around the original input image. The value of N is variable, and set by hand as opposed to calculated automatically. The dimensions (in pixels) of the average image is calculated using equations [3.13] and [3.14]

\[
X_{\text{average}} = \left\lfloor \frac{X_{\text{original}}}{N} \right\rfloor \tag{3.13}
\]
The initial segmentation part: Pixel-based background subtraction

\[ Y_{\text{average}} = \left[ \frac{Y_{\text{original}}}{N} \right] \]  

(3.14)

The values in the average image are calculated as the mean of all the values covered by the area of the N-by-N square. This is illustrated in figure 3.6 where each letter in the average image corresponds to the mean of the area of the same letter in the original image. In this case the value of N is set to two.

Once a mean value has been calculated, the N-by-N square is moved N pixels to the side or N pixels down if the end of the current line is reached. Once the square has been moved, another mean value is calculated for the average image. This procedure repeats itself until the entire input image has been covered.

If the \( X_{\text{original}} \) or \( Y_{\text{original}} \) values does not divide with N to a integer value, the remaining lines of the input image will not be taken into account.

After an average image has been created, it is traversed pixel-wise starting from pixel coordinates (1,1). Note that the top left corner has the coordinates (0,0). All eight neighboring pixels are examined to find the pixel with the highest (maximum) and lowest (minimum) value. Note that we do not start at coordinates (0,0) since this pixel does not have eight neighbors. Also note that the average image must at least be 3-by-3 pixels large for this operation to work. The side effect of starting at location (1,1) is a small border around the filtered image with unaffected pixels. However, this border can easily be truncated without much harm to the result. The same side effect also exist with mean and median filtering.

After both the maximum and minimum values from the neighborhood has been found, they are used to check if any pixels inside a N-by-N square in the original image needs to be changed. The pixels in question in the original image corresponds to the ones used to calculate the mean.

If we observe figure 3.6 the first and outermost pixels that may be replaced are those in the input image with label e, as pixel e in the average image is the only one which is surrounded by eight neighbor pixels.

The observed pixel in the N-by-N square can now either be left unchanged, or replaced by the maximum or minimum value. The decision is made taking the current value of the pixel into account.
account. If the observed pixel value is lower than the found minimum, the pixel is set to the
minimum, and vice versa if the observed pixel value is higher than the found maximum. Using
this strategy, a single high value in areas with lower values is reduced to the highest average
value observed, and likewise for a single low value. If the observed pixel value is between the
maximum and minimum values no change is made.

Once all the pixels in the input image has been checked, the first run is completed. Now the
value N is increased and the process of generating an average image, checking and replacing
pixels starts over. This is repeated until N reaches a specified end value. The N value does not
necessarily have to be increased by one for each iteration. The fact that the filter expands to a
bigger and bigger area, is why we have termed it as dynamically expanding.

The screenshot in figure 3.7 illustrates the effects of applying our filter on the reference sign.
As the reader can see, the result is not impressive. This is because our dynamic expanding filter
is not designed to filter noise of natural images (as they are captured from the camera), but it is
designed to filter out noise from the result of background subtraction, as will be shown in the
next section.

Figure 3.7: Dynamic expanding filtered screenshot of the sign.

It may be a bit tricky to understand the way the filter works from the description alone. Therefore a theoretical example is presented in appendix B.

3.3.4 Test results of the three filters on the initial segmentation output

The purpose of a filter is to remove noise without damaging the foreground object(s). In this
section we present tests of how well different filters perform when filtering noise from the
output of our own background subtraction algorithm. The filters tested are the three filters
described above, namely mean, median and our own dynamic expanding filter. We would
expect our own filter to work the best, as it is constructed especially for removing noise from
the output of a background subtraction algorithm, whereas the other filters are more generally useful.

For the test, we have used Matlab’s built-in functions for mean and median filters. For the dynamic expanding filter we have implemented a version in Matlab and C++ (in the Matlab version it is called ‘recursive filter’).

In figure the original image is shown to the left (before background subtraction) and the result of using our background subtraction algorithm is shown to the right. As one can see the background subtracted image has a lot of noise. The reason for the huge amount of noise, is that our background subtraction algorithm does not perform well when it encounters moving objects in the background, such as the trees seen in the image. The trees are moving due to windy weather.

As the careful reader may have noticed, the right image is a gray scale image. Therefore the filters have been applied to gray scale values (as opposed to binary values). This is contrary to the output of the filters which are shown as binary, even though these are also in gray scale. The reason for showing the output of the filters as binary, is that it will show the difference between the filters more clearly. Also it is because the binary values are often what is used for further processing. Furthermore, binary values are often what is presented to the user in background subtraction applications.

The conversion from gray scale to binary were done by making all values of 127 or less correspond to 0 and values between 128 and 256 correspond to 255.

![Figure 3.8: The reference image before (left) and after (right) background subtraction.](image)

The next two images in figure show the results of applying a mean filter to the output of our background subtraction algorithm. The window size used is 3-by-3 in the left image and 5-by-5 in the right image.

The left image does a poor job of removing the noise and still manages to damage the foreground objects somewhat. The damage is seen as a rounding of the edges.

The right image clearly does a better job (although still poor), of removing noise from the
3.3 Noise filtering

image. However the foreground objects are even more damaged. The reader should especially
look at the feet and legs of the two persons in the image.

![Figure 3.9: Mean 3-by-3 (left) and 5-by-5 (right) filtering of the reference image after back-
ground subtraction.](image)

In figure 3.10 the results of applying a median filter to the output of our background subtraction
algorithm are shown. The window size used is 3-by-3 in the left image and 5-by-5 in the right
image.

The left image does a poor job of removing noise, even worse than the mean filter. Though it
also does not damage the foreground objects as much as the mean filter. The damage is, like in
the mean filter, seen as a rounding of the edges.

The right does a clearly better job (although still poor), of removing noise from the image.
However the foreground objects are very damaged. The reader should especially look at the
feet and legs of the two persons in the image.

![Figure 3.10: Median 3-by-3 (left) and 5-by-5 (right) filtering of the reference image after back-
ground subtraction.](image)
Finally in figure 3.11 the results of applying our own dynamic expanding filter are shown. We have used a square size of 1-by-1, 2-by-2, 3-by-3 and 4-by-4 in the left image (see section 3.3.3 for explanation of square size). In the right image we have done exactly as in the left, plus applying the filter three times with square size 5-by-5.

The left image does a reasonable job of removing noise, better than both the mean and median filters, though it is still not very impressive. However it does very little damage to the foreground object. It does not round the foreground objects as the mean and median filters.

The right does a clearly better job of removing noise from the image and still leaves the foreground objects intact. The reader should especially look at the feet and legs of the two persons, and compare these to the original output of the background subtraction algorithm, and the mean and median filter results.

![Dynamic expanding size 1,2,3,4 (left) and size 1,2,3,4,5,5,5 (right) filtering of the reference image after background subtraction.](image)

As shown our filter does a clearly better job of removing noise without damaging the foreground objects, compared to the mean and median filters. However, there are problems with our filter. First, as seen in figure 3.11 it does a poor job of removing noise along the borders of the image. Second, very small foreground objects (for example a 4-by-4 box object) will be completely deleted by our filter. This will not happen with the mean and median filters if a small window size is used such as 3-by-3. Small objects can appear in a scene, if the objects are truly small like a bird or a person which is far away from the camera. Also objects can appear small, when they have been partly occluded by a background object, for example if a person is partly occluded behind a tree.

We do not see the first problem as significant, since it is not unrealistic that a program removes a 2-4 pixel sized border from an image before presenting it to the user. Secondly, with further thought, it may be possible to expand this algorithm to also include borderline pixels.

The second problem is more severe. Since it is inherent in our filter, that it will remove very small foreground objects. Therefore it is basically a tradeoff between how badly one wants to remove noise and how badly one wants to preserve small objects.
3.4 Our pixel-based background subtraction implementations and experiments

In the following sections we will describe our pixel-based background subtraction implementation and our experiments. We will start by listing the algorithms we have implemented, and then describe our most prominent implementation, which we also used for our experiments. Finally we will present our experiments, where we compare our own pixel-based background subtraction algorithm with the one done at University of Maryland[20]. The description of how to execute the pixel-based background subtraction program can be seen in appendix [D].

3.4.1 Our pixel-based background subtraction implementations

To achieve our goals we made a lot of experimental code. This has for the most part been implemented in Matlab, which were used as a prototyping environment. Developing in Matlab is relatively easy and fast, and so ideas can quickly be implemented. However, for reasons of efficiency, we have also implemented approaches in C++. Below the implemented algorithms are listed.

- Single channel (gray scale) Gaussian model, see section 3.1.3.1.
- Multi channel (true color) Gaussian model, with the assumption that color channels are independent, see section 3.1.3.1.
- Multi channel (true color) Gaussian model, with calculation of dependency between color channels, see section 3.1.3.1.
- Multi channel (true color) non-parametric model, with the assumption that color channels are independent, see section 3.1.4.2.

In additions to these models we have tried different techniques, with the models:

- Incremental update of training frames, see section 3.2.2. Used instead of having a static set of training frames that do not change over time.
- Exponential forgetting, see section 3.1.3.2.
- Chromatic colors, see section 3.2.1.
- Selective update. That is, only background pixels are updated not foreground pixels, see section 3.2.3.
- Fast pixel estimation. That is, only some regions are fully estimated. See section 3.2.4.
We have also tried some different filters, which have been applied both to the input images and to the result of pixel-based segmentation.

Mean and median filters were applied to both input images and the result of segmentation. The dynamic expanding filter were only applied to the result of segmentation, as it was not designed for ordinary images (as captured by a camera).

Finally we have implemented the possibility of reading from a live video stream. We can read from both a Video4Linux webcam and from a IEEE1394 based camera (Firewire). Furthermore we can read a sequence of images from the hard drive.

Our pixel-based background subtraction implementation, based on the non-parametric model, will be described in more detail below.

### 3.4.2 Non-parametric model

Our own implementation, which we used for testing purposes, was a non-parametric algorithm, see section 3.1.4.2 about the non-parametric model. Though in the implementation we do not estimate a standard deviation for each pixel, instead we set a standard deviation for all pixels as a tuning parameter.

Our system uses chromatic colors, incremental update, and selective update. However, the background is not always selectively updated, sometimes we use blind updating. The decision of which frames are used to update the background selectively, and which are used to blindly update the background, are made by random. That is, for each new frame 'a dice is thrown' and it selects how the background will be updated using this frame. How likely it is that a frame will be used for selective update is set as a tuning parameter. In our test we update 15% of the frames blindly.

We use the dynamic expanding filter (see section 3.3.3) to remove noise in the pixel-based background subtracted image. Which filter sizes are used can be set when the initial segmentation part of the system is started (tuning parameter). Furthermore the dynamic expanding filter can be changed to use the second largest and second smallest found neighbor values in the average image. This will help remove unwanted pixels on the border between foreground and background. In our test we used filter sizes of 1, 2, 3, 4, 5, 5, and 5 for each frame, that is the filter were run on each frame seven times with the filter sizes mentioned above. Furthermore, a dynamic expanding filter which used the second largest/smallest square, with square size one, was used after the other filters.

We also used 'fast pixel estimation', as described in section 3.2.4. We used a square size of four.

We used 50 training frames for our tests, the threshold was set at 0.04 and the standard deviation was set at 2.
3.4.3 Test results of our pixel-based background subtraction algorithm

In this section we present some results of our test of our pixel-based background subtraction implementation. The whole test is described in details in appendix C.

From our experiments we can conclude that we definitely have a serious problem with dark areas which needs to be solved to avoid too much false foreground detection. We are also positive that this will improve our background subtraction quite a bit, as we will be more likely to discover actual foreground objects which is now covered by the false noise.

All the test movies more or less demonstrates pixel-based background subtraction issues mentioned in section 3.1.1.2. Our movies show these problems quite good. Just to mention a few, we can see the problem of the object disappearing from the 'camouflage' and 'slowly moving objects' movies (see appendix C). Many of our movies also show shadows as foreground objects, even though we use chromatic colors to try and remedy this issue.

In general our pixel-based background subtraction, compared to the results of the UMD-BGS [20], has less holes in the detected objects, which we regard as an advantage. However, the UMD-BGS does a better job of actually detecting which parts should be foreground. This can for example be seen in many of the outdoor movies, where we have more false detections and later less or missing actual foreground detections. The 'road and grass walk' movie where a bottle is being thrown in the air illustrates this (see appendix C).

With most of the indoor movies we feel that we are able to keep up with the UMD-BGS and some movies even provide a better result. Once we move outdoor however, the UMD-BGS does a better job with the difficult scenarios. However, our pixel-based background subtraction program is applicable for the use as a preprocessing step for the modeling part of our system, since we have to control the environment, under the movie shooting, to get best results possible.

The next chapter will describe the second part of our application, which is the modeling learning part.
Chapter 4

The model learning part: Pedestrian representation

Our system needs a model of the object of interest to make the tracking possible. Therefore, we will in this chapter describe the model learning part of the system, which is about modeling our object of interest, which is a pedestrian.

Generally, constructing a model of pedestrians, for the use in a tracking system, is a very difficult task. Since people dress in very different colors that sometimes blend with the background, they wear hats or carry bags, and stand, walk and change direction unpredictably. Different people have different shapes, heights, and size. The background is also very various, containing buildings, trees, moving or parked cars, cycles, street signs, signals etc, which makes the tracking very hard. Moreover, sudden changes in the backgrounds makes the problem, of developing such a system, even more complex. This complexity encourages further research in the field.

Therefore, constructing a good model of pedestrians will help eliminating a lot of the above mentioned problems.

We will start this chapter by describing the requirements for the model. Next we will discuss which model type is most applicable for our system. After we have chosen the model type, we will describe some representation techniques which we can use to model our object of interest (pedestrian). The techniques we will describe are active contours, templates and exemplars. Then we will describe some methods related to the chosen representation technique. These are exemplar extraction, clustering, distance measuring, and probability calculation of one stance following another (motion model). Then the interface to the recognition part of the system will be described. The chapter will end by describing the implementation and experiments of the modeling part of the system.

4.1 Model requirements

Pedestrian (or any other object) modeling is an essential part of our model-based background subtraction system. Although a great number of pedestrian/human models have been proposed
in the literature, few of them are appropriate for our purpose, which is pedestrian tracking. Most models are developed for other purposes than for object tracking, such as object detection [22] [56] or figure animation [47]. These models are either too complicated to be practical for efficient pedestrian/human recognition and tracking, or can just be used to detect a particular person rather than all instances of humans. The common drawbacks with previous human models are:

- The modeling of pedestrian shapes are not invariant to geometric deformations, thus, they can only recognize people of a fixed size or orientation.

- The models are usually specific to a particular person, and do not model the statistical variance among individuals.

- Most models only represent the shape of a human body, but cannot handle the shape variation due to clothing.

- Some models can handle certain global shape variance, but they have difficulty dealing with large articulated motion and partial occlusion.

For our system to be effective, our object (pedestrian) model should overcome these drawbacks. It should satisfy all the following requirements of a good model for object classification and segmentation:

1. It should not depend on scale, orientation, and position of objects. This means it should handle view-dependent shape variation and planar rotation (in the image plane). These variations should also be allowed among individuals.

2. It should be robust towards noise resulting from digitization noise and foreground/background segmentation errors.

3. It should be robust to partial occlusions of an object.

4. It should allow for articulated moving parts.

5. It should be efficient towards space, the model should use as little disc/memory space as possible.

6. It should support efficient shape recognition/classification for both speed and disc/memory space.

4.2 Model representation

Model-based vision is firmly established as a robust approach to recognizing and locating known rigid objects in the presence of noise, clutter, and occlusion. It is more problematic
to apply model-based methods to images of objects whose appearance can vary, though a number of solutions based on the use of different techniques can be used.

What we wish in this project is to derive a model that is able to represent the shapes of pedestrians under their walking cycle, as shown in figure [4.1]. Different pedestrians have sufficiently different shapes, depending on the height, size, clothes, etc. This means that a rigid model would not be appropriate.

![Figure 4.1: Pedestrian shapes under walking cycle.](image)

Our aim is to build a model, which describes both typical pedestrian shapes and their typical variability. We will in the next sections describe and compare different techniques, which can be used for representation of objects. However, before that we have to decide what model type is most applicable for our project. Therefore, we will start by describing and comparing different model types used for object representation.

### 4.2.1 Model types

In computer vision three types of models have been used for representing objects. These types of models are:

- **2D models based on object general properties:** Some approaches based on general object properties have been mentioned under section [2.2.1]. These approaches [76][51][72][31] are based on 2D models. We did not mention any approaches to pedestrian/human representation using this type of models under that section. There are however a few general approaches to human movement. It is about describing human movement in terms of simple low-level, 2D features from a region of interest. Models for human action are then described in statistical terms derived from these low-level features or by simple heuristics [61]. However, this type of model have not been used successfully for human modeling. This is because of the many degrees of freedom the human body possesses (see section [2.2.1]).

  This type of 2D model (without explicit shape models) has been especially popular for applications of hand pose estimation in sign language recognition and gesture-based dialogue management [66].

- **2D explicit models of object shape:** We have mentioned some approaches based on this type of 2D model under section [2.2.2]. These approaches [41][64][37] discuss work
which use explicit a priori knowledge of how objects such as the human body, face, hand, or other objects appear in 2D, taking essentially a model-based approach to segment, track, and label objects parts. Since occlusion makes the problem quite hard for arbitrary movements, many systems assume a priori knowledge of the type of movement or the viewpoint under which it is observed. However, human/pedestrian is the most interesting object for this type of model-based approaches, since the human body is a very complex object and pedestrian detection and tracking is a very common problem in the computer vision field, and its typically segmented by background subtraction, assuming a slowly changing or stationary background and a fixed camera. The models used are usually human shapes modeled by:

- Active contours
- Templates and Exemplars

The type of the model and the techniques used for the representation of the object, strongly influences what features are used for tracking.

**3D models of object shape:** 3D models is mostly used for recovering 3D articulated pose over time. A typical approach to pose recovery is by formulating the problem as a search problem which entails finding the pose parameters of a graphical human model whose synthesized appearance is most similar to the actual appearance of the real human. Then an appropriate measure between model and scene is required [35].

The general problem of 3D motion recovery from 2D images is quite difficult. In the case of 3D pedestrian tracking, however, one can take advantage of the large available a priori knowledge about the kinematic and shape properties of the human body to make the problem tractable. Tracking also is well supported by the use of a 3D shape model which can predict events such as (self) occlusion and (self) collision.

3D models for the human body generally consist of two components:

- A *representation for the skeletal structure (the “stich figure”)*: The stick figure is simply a collection of segments and joint angels with various degree of freedom at the articulation sites, where each human body part is represented by a stick and the sticks are connected by joints (see figure 4.2).

- A *representation for the flesh surrounding it*: The representation for the flesh can either be surface-based (e.g., using polygons) or volumetric (e.g., using cylinders, see figure 4.3). There is a trade-off between the accuracy of representation and the number of parameters used in the model. Many highly accurate surface models have been used in the field of graphics to model the human body [57], often containing thousands of polygons obtained from actual body scans. However, human models used for computer vision do not have to meet the standard of being highly realistic and natural looking as long as their shape approximates the real human shape well enough to support image segmentation.

As a general rule, the more complex the 3D human body model, the more precise the tracking results. But on the other hand, complex 3D human body model leads to extra computational cost.
4.2.1 Model type choice

The choice of whether to pursue a 2D or 3D approach is largely application dependent.

A 2D approach is effective for applications where precise pose recovery is not needed or possible due to low image resolution (e.g. tracking pedestrians in a surveillance setting). A 2D approach also represents the easiest and best solution for applications with a single human involving constrained movement and single viewpoint.

A 3D approach makes more sense for applications in indoor environments where one desires a high level of discrimination between various unconstrained and complex (multiple) human movements (e.g. humans wandering around, making different gestures while walking and turn-
It is unlikely that this can be achieved by a purely 2D approach; a 3D approach leads to a more accurate representation of physical space which allows a better prediction and handling of occlusion and collision. It leads to meaningful features for action recognition, which are directly linked to body pose. Furthermore, 3D recovery is often required for virtual reality applications.

However, we think that it is fair to say that the results of vision-based 3D tracking are still limited at this point. Few examples of 3D pose recovery exist in the literature and most of these introduce simplifications (e.g. constrained movement, segmentation) or limitations (e.g. processing speed) that still require improvement with respect to robustness. Robust 3D tracking results have been particularly scarce for approaches using only one camera. The benefit of using stereo vision or multiple cameras to achieve tighter 3D pose recovery has been quite evident [35][47]; body poses and movements that are ambiguous from one view (by occlusion or depth) can be disambiguated from another view. Stereo vision also aids judging the distance to a point, as it makes triangulation possible.

We chose in this project to use a 2D model, because we think that a 2D model is sufficient for our project. Another reason is that we do not want our system to use some special and expensive hardware and cameras, which 3D models may need to work. Furthermore, it is very difficult to construct a 3D model automatically.

It is now clear for us that a 2D explicit model will be used for this project. The reason for using an explicit model rather than a feature-based model, is that the feature-based model can not handle objects, which do not have general properties and have many degrees of freedom, such as the human body. It is now relevant and interesting to describe and compare the 2D techniques listed under the section 4.2.1 which we can make use of to build an explicit model of our object of interest, which in this case is pedestrian. The techniques will be described in the following sections.

4.2.2 Active contour models

Flexible shape models have been shown to be useful for applications in tracking and image interpretation [48], [77]. These tasks are made easier by restricting the solution space of allowed shape deformations. For instance, we could incorporate some knowledge of the object in question.

There is a substantial literature describing the use of flexible models or deformable templates to aid image interpretation. Such models usually have a number of parameters to control the shape and pose of all or parts of the model. These models are typically called “active contours” or “snakes” [48].

In [48] they describe active contour models (“snakes”) which are attracted to image features. These include spline curves, which are modeled as having stiffness and elasticity and are attracted toward features such as lines and edges. Constraints can be applied to ensure that they remain smooth and to limit the degree to which they can be bent. Snakes can be considered as
parameterized models, the parameters being the spline control points. They are usually free to take almost any smooth boundary with few constraints on their overall shapes.

In order to model a shape (in this case pedestrian), we can for example represent it by B-spline control points (B-splines have been used for tracking image contours [25]). For example points around the boundary, as seen in figure 4.4. This must be done for each shape in the training set.

![Control points around the foreground object boundary.](image)

For instance, using the shapes in figure 4.1 as a training set, we can build a shape model by representing each example as a set of labeled points, calculating the mean positions of the points and the main ways in which the points from each example tend to vary from the mean.

As an example, in [29] they describe a point distribution model in which a set of labeled points is hand generated from a set of training images of a particular object in a variety of positions. The shapes are aligned and the deviations from the mean are analyzed. The most significant modes of variation give a compact representation of the generic object shape (based on the training set) whilst the other modes are ignored as they contribute little to the overall shape.

The main problem with the active contour approach, is that the control points for every shape is acquired with human intervention, where fixed points must be selected by eye from example images. This is time consuming and impractical in many cases. Several approaches, such as [40] and [49], try to automate this process of finding the “landmark” of control points. However, these approaches are for specific objects, and it is unlikely that they will work successfully in general.
4.2.3 Templates and Exemplars

Templates and exemplars have been used in computer vision field for representation of different objects.

Template or exemplar matching is a typical approach to object recognition and tracking. For example, consider the noisy human shapes shown in figure 4.5. These noisy shapes are called 'exemplars'. The noise-free version shown at the left in figure 4.5 can be used as a 'template'. To classify one of the noisy shapes (exemplars), we can simply compare it to the template. This can be done in a couple of equivalent ways:

- The maximum correlation approach: Count the number of agreements (black matching black and white matching white). Pick the class that has the maximum number of agreements.

- The minimum error approach: Count the number of disagreements (black where white should be or white where black should be). Pick the class with the minimum number of disagreements.

Template and exemplar matching works well when the variations within a class are due to "additive noise." Clearly, it works for this example because there are no other distortions of the characters: translation, rotation, shearing, warping, expansion, contraction or occlusion. It will not work on all problems, but when it is appropriate it is very effective. However, some different more sophisticated approaches have been developed to deal with these problems.

In [53] human body parts are detected by locating limbs by template matching, using two separate templates for the torso and the limbs, each of which emphasizes long regions that are bright along the axis and darker closer to the boundaries and have edges of an appropriate orientation on either side. The body part locations are found by convolving the image with a template in a range of orientations.

Toyama and Blake [70] use 2D exemplars to track people in video sequences.

The differences between the terms exemplars and templates are often emphasized, even though the differences are minor.

The dictionary [6] gives the following definition for a template:
“A pattern or gauge, such as a thin metal plate with a cut pattern, used as a guide in making something accurately, as in woodworking or the carving of architectural profiles.”

The same dictionary [6] gives the following definition for an exemplar:

“A model, original, or pattern, to be copied or imitated; a specimen; sometimes; an ideal model or type, as that which an artist conceives.”

This means that a template must be ‘perfect’ to be used for accurate matching. Therefore templates are usually done by hand. However, to automatize this process, we could generate a good template out of a set of minor good exemplars. Exemplars can contain all image noise and distortions while in a template, these are usually deleted.

However, what we use in our project is more related to exemplars than templates. This is because none of the shapes we have are ‘perfect’, and there will always be some noise in the images, even though the images were taken under a controlled environment (see figure 1.1 under section 1.2). Because of that we will from now on only use the term exemplar.

4.2.4 Our choice

Our goal in this project is to develop a general system which can be used for different objects where the model can be generated automatically or at least with minimal human intervention.

There are many problems to address for object modeling. First of all, we need to consider what features the recognition part of the system needs for recognizing and tracking of pedestrians. Contours (the outline of objects) or silhouettes are commonly used to overcome variable texture. Therefore, we employ the pedestrian contours and the silhouettes of the human body as features to recognize pedestrians. By contour we mean the outline of the shape, as shown in figure 4.6. By silhouette we mean an outline that is solidly colored in, as shown in figure 4.6.

We have chosen exemplars to represent the shapes of pedestrians in different view angels, since other techniques involve specialized equipment and/or can not be constructed with minimal human intervention. Exemplars can be constructed automatically from an image sequence, as captured by an ordinary camera. This sequence can be background subtracted and the exemplars can then be extracted. These exemplars can then be clustered and put into a hierarchy (see section 4.4.1.3) for easy lookup. Instead of background subtraction, other techniques are also available, such as edge detection.

One advantage when using exemplars in a hierarchy is that comparing query with all shapes is unnecessary. For instance, if a query is distant from a pedestrian, it will be distant from all pedestrians. In context of all shapes pedestrians will be tightly clustered where each category is represented by a few exemplars. Normally indexing results are excellent using a large database with a lot of shapes (exemplars) clustered under fewer shapes. This can be done by using a hierarchical representation.
4.3 Extraction of information from background subtracted movies

The information (shape of pedestrians) will be represented by exemplars. The exemplars are needed to construct the pedestrian model. Since we want to automatize the construction of the model, we need a way to extract these exemplars from a video sequence with as little human intervention as possible. We also want the system to be able to handle other kinds of objects than pedestrians. However, there will always be some kind of human intervention to construct these exemplars as:

- Recording the video is in itself a human activity.
- Not all of the movie will contain usable exemplars, and a human must somehow tell the system what is the desired object and what is not.
- Heuristics could be used to distinguish between pedestrians and other objects. However this is just another kind of human intervention and these heuristics will not work for other kinds of objects.

One could create a program to examine old video sequences from a TV station or from the Internet. This would remove the need for oneself to record these movies. However, this would enlarge the need for human intervention when distinguishing between different objects, as one can be sure that the system would not only examine video sequences of pedestrians.

Our goal is to minimize the burden put upon the user. We do this by both using general heuristics, that apply to all objects and by asking the user questions when necessary.

We have chosen to record our own movies, as it seemed too great a task to construct a program, which finds videos from the Internet and extracts exemplars of these. To get silhouettes of
pedestrians we background subtract the movies we record. This is what the initial segmentation part used for in our system.

Before turning the result from background subtraction into exemplars we filter the images. We filter with the dynamic expanding filter, with the n-connected filter, and with the median filter. See section [3.3.3] for information about the dynamic expanding filter. The n-connected filter works by first turning any region of background pixels with less than \( n \) elements into foreground. Then it turns any region of foreground with less than \( n \) elements into background. We use the first filter as it is very capable at removing noise without hurting the exemplar. The second filter as it will clean up remaining noise (again without hurting the exemplar) and as it works near the edges of the images, which the dynamic expanding filter does not. We use the median filter as it is good at smoothing the edges of the exemplar. This will also hurt the sharp corners, however the advantage of smoothing out the exemplar outweighs unwanted smoothing (such as sharp corners).

Then these frames (silhouettes) are handed off to a program which removes outliers. For example, if we want to model pedestrians and a car appears, we do not want the car to be a part of the model. How this is done is described below:

1. Locates exemplars in each frame.

2. Each exemplar is tagged with a frame and a series number. A new series is started whenever a new movie is recorded, or when no exemplars are extracted from a frame for some reason, or when the exemplar changes direction. In this way we can later discover which exemplars come from consecutive frames. This is important as a lot of exemplars may be cut away. The direction is also used to mirror normalize the exemplars. If the direction is from left to right, nothing happens. But if the direction is from right to left, the exemplars are mirrored around the center vertical axes. This means that we do not have to record exemplars walking from both directions.

3. The exemplars are then removed if they are closer to a border than a constant named \( \text{MinDistanceFromBorder} \).

4. The exemplars are thereafter removed if they are part of series with fewer than a constant named \( \text{MinNumberOfFramesInSeries} \) members.

5. The exemplars are then normalized as described in section [4.4.2.1].

6. In the next step user evaluation is introduced. But first the concept of object types need to be introduced. An object type consist of:

   - a name \( (O^n_{\text{name}}) \)
   - a number of human approved exemplars \( (O^n_{\text{human}_m}) \)
   - a number of machine approved exemplars \( (O^n_{\text{machine}_m}) \)

where \( n \) is the object type number and \( m \) is an exemplar number. Human approved exemplars have been assigned to this object type by the user. The human approved exemplars
are used to approve the machine approved exemplars automatically. When the algorithm starts only one object type exist, namely 'trash' where all exemplars of unknown type goes. Exemplars are assigned to object types in the following way:

(a) For each exemplar the distance to each object type, with non empty $O_{human}$, is calculated as:

$$d(a, O^n) = \arg \min_{e \in O_{human}} d(a, e)$$

where $d(a, e)$ is the symmetric Chamfer (silhouette) distance between exemplar $a$ and $e$, see section 4.4.2.4 for description of Chamfer distance.

(b) If the closest object type number $n$ is below a constant named $MaxDistanceFromRepresentative$ then the exemplar is assigned to $O^n_{machine}$. Else the user is asked. He can either assign it to an existing object type or write the name of a new object type, which will then be constructed. In both cases the exemplar is assigned to both $O_{human}$ and $O_{machine}$ for the chosen object type.

7. All object types machine approved exemplars are then examined. An exemplar from $O_{machine}$ is removed if it is part of a chain with fewer than $MinNumberOfFramesInSeries$ members. By a chain we mean consecutive exemplars where the series number is the same and the frame number get one higher for each exemplar in the chain. There can be jumps in frame number, without a new series number, as some exemplars could have been assigned to another object type by the user or the system.

The extracted exemplars, to be used in further processing, can now be found in each object types $O_{Machine}$ except for the 'trash' type.

This algorithm will may remove most, if not all, unwanted exemplars without removing too many wanted exemplars. Furthermore we hope to minimize the need for user input.
4.4 Structuring the extracted information

Extracting information, from the pixel-based background subtracted movies, will result in many different exemplars representing different pedestrian shapes. What we need now is a technique for grouping or indexing these exemplars in a way that makes it easy for the recognition part of the system to access them. Therefore, we will in the next sections investigate different clustering techniques and different distance measures, which are used by the clustering algorithms.

4.4.1 Clustering

Generally, the purpose of clustering is to place objects into groups or clusters suggested by the data, not defined a priori, such that objects in a given cluster tend to be similar to each other in some sense, and objects in different clusters tend to be dissimilar. Thus a cluster is defined as a set of similar objects. Each object could be described by some metrics. The differences in these metrics are used to calculate a distance (see section 4.4.2) between the objects, and the distance determines the clusters.

The main goal of clustering is maximizing the differences between clusters relative to variation within clusters.

A vast number of clustering methods have been developed in several different fields, with different definitions of clusters and similarity among objects.

The representations of objects to be clustered can take many forms. The most common are:

- A similarity matrix, in which both row and columns correspond to the objects to be clustered.
- A coordinate matrix, in which the rows are observations and the columns are variables. The observations, the variables, or both may be clustered.

Clustering algorithms attempt to find natural groups of objects (or data) based on some similarity. They can also find the centroid of a group of objects or data sets. The centroid of a cluster is a point (or a object) whose parameter values are the mean of the parameter values of all the points (objects) in the clusters. To determine cluster membership, most algorithms evaluate the distance between a point (an object) and the cluster centroids. The output from a clustering algorithm is basically a statistical description of the cluster centroids with the number of objects in each cluster.

Generally, the distance between two points or objects (in our case exemplars) is found by a distance measure taken as a common metric to assess the similarity among the components of a population. Some popular distance measures are described in section 4.4.2.
4.4 Structuring the extracted information

4.4.1 Clustering methods

The various clustering concepts available can be grouped into two broad categories:

- Hierarchical clustering method: This method includes those techniques where the input data (in our case exemplars) are not partitioned into the desired number of classes in a single step. Instead, a series of successive fusions of data (exemplars) are performed until the final number of clusters is obtained.

An example of a hierarchical clustering algorithm is the minimal spanning tree [59]. This hierarchical algorithm starts by considering each component of the population to be a cluster. Next, the two clusters with the minimum distance between them are fused to form a single cluster. This process is repeated until all components are grouped into the final required number of clusters.

There are two different approaches to hierarchical clustering:

- Divisive clustering: This is a top down approach. It starts with all objects in a single set, and then it repeatedly splits set(s) into subsets. The approach stops when the subsets are sufficiently uniform, or each object is in its own subset.

- Agglomerative clustering: This is a bottom up approach. It starts with each object as its own set, and then it repeatedly merges similar sets. The approach stops when everything is in a single set.

Given an input set of objects or data, the result of hierarchical clustering is producing a hierarchy (tree) in which nodes represent subsets of the input set, and the whole hierarchy represents the structure found in the whole set. Therefore, hierarchical clustering produces a hierarchy with the following properties:

- The root cluster contains the whole input set of objects.
- The leaves are the individual objects of the input set (there can be more than one object in a leaf).
- The internal nodes are defined as the union of their children.
- Clusters are organized so that one cluster may be entirely contained within another cluster, but no other kind of overlap between clusters is allowed.
- Each level of the tree represents a partition of the input objects (data) into several (nested) clusters or groups.

- Nonhierarchical (partitioning) method: This method includes those techniques in which a desired number of clusters is assumed at the start. These does not involve a hierarchical (tree-like) representation of the objects. Objects are allocated among clusters so that a particular clustering criterion is optimized. A possible criterion is the minimization of the variability within clusters, as the mean distance from members to cluster ‘centroid’.

An example of an nonhierarchical clustering algorithm is the popular K-means algorithm (see section 4.4.1.3). It examines each object in the population and assigns it to one of
the clusters depending on the minimum distance. The centroids position is recalculated every time an object is added to the cluster and this continues until all the objects are grouped into the final required number of clusters.

Given an input set of objects (or data) and a fixed number of clusters K, the result of nonhierarchical clustering is returning a partition of the set into K subsets. The nonhierarchical clustering properties are as follows:

- The number of clusters are fixed and must be specified from the beginning.
- Each object is placed in one and only one cluster.

### 4.4.1.2 Algorithm choice

The choice of whether to use hierarchical or nonhierarchical clustering is not an easy task. When a priori knowledge about the data (objects) is not available or insufficient, it may be desirable to try different algorithms to explore the data (objects) and get meaningful clustering results through comparisons. Finding the best algorithm is also a difficult task. The hardest problem in comparing different clustering algorithms is to find an algorithm independent measure to evaluate the quality of the clusters. The quality of a clustering algorithm is dependent on the distance measure (see section 4.4.2) used in the algorithm. Therefore, we approach this problem by investigating different clustering algorithms and different distance measures (also the same clustering algorithm with different distance measures) and compare the results. However, since there are a vast number of clustering and distance measure algorithms, we will only experiment with the most known and used algorithms.

We have listed in table 4.4.1.2 some advantages and disadvantages for the two different clustering methods described earlier. Some of the advantages and disadvantages are not proved to be true, different researchers have different experiences. However, we believe that these are the most common properties.

The table shows that the hierarchical clustering techniques are more preferable for detailed analysis and are more efficient for small datasets. However, these techniques can be inefficient to compute when the datasets become very large. These techniques could compute clusters that are not optimal, this could happen because they do not allow branches to split and then merge back (divisive (top down) algorithms always split and do not merge back, whereas agglomerative (bottom up) algorithms always merge and do not split). The hierarchical method of clustering is rigid, which means that we cannot correct errors made earlier in the clustering process, this means that the results will be very bad if the algorithm make errors at the very top or the very bottom of the hierarchy, depending on the algorithm type (divisive or agglomerative).
### 4.4 Structuring the extracted information

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hierarchical</strong></td>
<td>- Indicates more complex relationships&lt;br&gt;- Efficient for small datasets</td>
<td>- Expensive to compute for very large datasets&lt;br&gt;- The clusters may not be optimal&lt;br&gt;- Rigid, cannot correct errors made at the top/bottom of the tree</td>
</tr>
<tr>
<td><strong>Nonhierarchical</strong></td>
<td>- Simple to understand and implement&lt;br&gt;- Relatively cheap to compute</td>
<td>- Some algorithms can not be used for some types of data&lt;br&gt;- User needs to specify the number of the clusters and when the algorithm should stop</td>
</tr>
</tbody>
</table>

Table 4.1: The advantages and disadvantages of clustering methods.

The same table also shows that the nonhierarchical clustering techniques are simple and easy to implement. This method of clustering is usually cheaper to compute than the hierarchical method and is generally preferable for efficiency, especially for large datasets (the efficiency depends on the used distance measure). However, some nonhierarchical clustering techniques (e.g. K-means, see section 4.4.1.3) do not make sense for some types of data, such as names, where it will not make sense to compute the mean of the names. For nonhierarchical algorithms, the user needs to specify the number of clusters and when the algorithms can stop (there are some algorithms for estimating the number of clusters automatically, such as MDL [39], BIC [33], and SIC [67]).

#### 4.4.1.3 Our approach

While there is no proof of which clustering method performs better, a combination of the two methods may give the best results. Therefore, we will in the following sections describe our approach, which is heavily inspired from [36], which uses this kind of combination.

When possible we use a similarity matrix, in which both row and columns correspond to the exemplars to be clustered. We do this as a similarity matrix is more efficient, since we only calculate the distance between objects once. Furthermore, a similarity matrix can be reused if we need to cluster the same exemplars (or a subset of them) again. It is possible to use a similarity matrix for alternative K-means (see below) and simulated annealing (see below) algorithms. For algorithms, such as K-means, it is not possible to compute a similarity matrix beforehand, as it constructs new exemplars, namely the means.

**The hierarchy:** For our system we need a hierarchical representation (clustering) of the exemplars (pedestrians shapes) to provide the recognition part of the system with fast searching and checking abilities. This means that when doing the recognition, one can just follow the tree
path through the clusters and do not have to check all the exemplars. We use a top down (divisive) approach (see figure 4.7). Figure 4.7 illustrates how our clustering algorithm works. The first level is the root of the hierarchy (tree), which contains all the exemplars (one big cluster). Then the second level will be constructed by using a nonhierarchical clustering technique (e.g., K-means) to cluster the exemplars from the root. The third level will be constructed by clustering each cluster from the second level by the same nonhierarchical technique. This procedure will proceed in the same manner until a specified number of levels are reached or there are only some few exemplars left in a cluster.

![Figure 4.7: An example of our clustering approach: 30 exemplars with K = 3 and the algorithm stops after reaching 3 levels.](image)

This way the nonhierarchical clustering determines the cluster membership, which will make it possible to compute the probability of which exemplar follows another exemplar (see section 4.5). These probabilities are very useful for the recognition and tracking part of our system.

We used a normal hierarchical top down approach to construct a tree. We could have tried a bottom up approach, which we believe will speed up the algorithm if there will be many levels in the tree, however we think for our system the top down approach is more suitable, because it will obtain the main structure of the exemplars, since the focus will be on the upper levels of the hierarchy (tree), and the speed of the algorithm is not that important for our system, since the modeling part of the system is going to be done offline and not in realtime.

For the recognition part of the system, the traversing of the tree (hierarchy) is important. Since each new exemplar (the extracted exemplars from the input images, which are delivered from the initial part of the system) has to be compared with the exemplars in the tree for the best match. This could be done by comparing the new exemplar with the clusters’ centroids.

We want to traverse the tree so that we a) find the best matching exemplar and b) do it as efficiently as possible. This may result in choosing multiple paths (branches) at times and only one branch at other times.
4.4 Structuring the extracted information

When traversing down the tree one should not always choose the greedy approach, which would be to only choose the path of the closest centroid, as this will not always result in finding the closest exemplar (the best match). This is illustrated in figure 4.8. The figure shows the case where the new exemplar is closer to the centroid of cluster A than the centroid of cluster B, but the new exemplar is closest to an exemplar which lies in cluster B than to the other exemplars, which lie in the closest centroid’s cluster. In this case the closest centroid is not the best choice. Because the new exemplar will match best with the exemplar from cluster B.

The figure shows only two dimensions (degrees of freedom), however the case will not be improved with more dimensions.

Therefore, we must use another way of traversing the tree. This is done by calculating the minimal and the centroid distance from the new exemplar to each cluster.

By centroid distance we mean the distance from the new exemplar to the centroid.

By minimal distance we mean that from all exemplars in a cluster to the new exemplar the distance is at least some value. For example, the distance from each member of a cluster to the new exemplar is minimum 40. We use the triangular inequality (see below) to calculate the minimal distance.

By using the minimal and the centroid distances we can rule out hole branches, when traversing the tree. For example, if the minimal distance to cluster A is 40 and the centroid distance to cluster B is 20, we can rule out cluster A all together, as no exemplar in cluster A can be closer to the new exemplar than the centroid of cluster B. However, we can not always rule out all branches except one. For example, if the minimum distance to cluster A is 50 and to cluster B is 70, and the centroid distance to cluster A is 80 and to cluster B is 90, we cannot rule out any of the clusters (branches), as both clusters can contain the exemplar with the smallest distance to the new exemplar. We will then have to investigate the children of both cluster A and cluster B. These children will then be compared to each other.

Therefore, the recognition part should make use of the triangle inequality when traversing the tree (hierarchy). This will be done as follows:

1. \( d(a, b) + d(b, c) \geq d(a, c) \) (symmetry)
2. \( d(a, b) \geq d(a, c) - d(b, c) \) (symmetry)
3. \( d(a, b) \geq d(a, c) - d(c, b) \) (symmetry)

In other words, the distance from \( a \) to \( c \) via \( b \) is longer or equal to the distance from \( a \) to \( c \). The distance from \( a \) to \( b \) is longer or equal to the distance from \( a \) to \( c \) minus the distance from \( b \) to \( c \). The distance from \( a \) to \( b \) is longer or equal to the distance from \( a \) to \( c \) minus the distance from \( c \) to \( b \).

Let \( a \) be the new exemplar that must be compared with the clusters in the hierarchy, let \( c \) be a centroid for cluster \( C \), and let \( b \) be the exemplar in cluster \( C \) which is longest from the centroid \( c \).
Let $|ac|$ be the distance from $a$ to $c$, and let $|cb|$ be the distance from $c$ to $b$, and let $|ab|$ be the distance from $a$ to $b$.

Now let us for example say that $|ac| = 100$ (centroid distance) and $|cb| = 40$ (maximal distance from centroid to the exemplar which is farthest away from the centroid). Then from the third rule, $|ab| \geq 100 - 40 = 60$. This means that all the exemplars in cluster C are 60 or longer from the new exemplar $a$.

![Figure 4.8: An example of a special case for exemplar matching.](image)

The recognition part will often have a limit on how far away a match must be. That is, it should be no farther away than $x$. Here the triangular inequality can also be used to rule out some clusters, if the minimal distance is greater than $x$.

The interesting part of our clustering algorithm is the nonhierarchical approach, since the clusters have to be in good quality to get good results. We have experimented with the following nonhierarchical algorithms:

- K-Means
- Alternative K-Means
- Simulated Annealing
Popular nonhierarchical clustering algorithms: We will here describe the above listed algorithms and our experience with them using the different distance measures described in section 4.4.2.

- K-means clustering: K-means is a well-known nonhierarchical clustering (partitioning) method. Objects are classified as belonging to one of K groups, K chosen a priori. This means that in our case the algorithm starts with an initial partition of the objects (exemplars) into K clusters. Then subsequent steps modify the partition to reduce the sum of the distances from each exemplar to its ‘centroid’. The modification consists of allocating each object (exemplar) to the nearest of the K means of the previous partition. This leads to a new partition for which the sum of distances is strictly smaller than before. The improvement step is repeated until the improvement is smaller than a certain threshold. This approach is very fast and minimizes the overall within cluster scatter by the iterative reallocation of cluster members. However, the speed heavily depends upon the speed of computing the means and distances between the objects (exemplars). There is a possibility that the improvement step leads to fewer than K partitions. In this situation one of the partitions (generally the one with the largest sum of distances from the mean) is divided into two or more parts to reach the required number of K partitions. The algorithm can be rerun with different randomly generated starting partitions to reduce the chances of the heuristic producing a poor solution. Generally the number of ”true” clusters in the data is not known. Therefore, it is a good idea to run the algorithm with different values for K that are near the number of clusters one expects from the data to see how the sum of distances reduces with increasing values of K.

The properties of the K-means algorithm are the following:

- There are always K clusters.
- There is always at least one item in each cluster.
- The clusters are non-hierarchical and they do not overlap.
- Every member of a cluster is closer to its cluster centroid than any other cluster centroid.

The algorithm’s process is as follows:

- The dataset is partitioned into K clusters and the data points are randomly assigned to the clusters resulting in clusters that have roughly the same number of data points.
- The mean (centroid) of each cluster is calculated, where each object is assigned to one of the new K means. This is called the mean calculation step.
- Then for each data object:
  - Calculate the distance from the object to each cluster centroid.
  - If the object is closest to its own cluster centroid, leave it where it is. If not move it into the closest cluster centroid.

This is called the clustering step.
- Repeat the mean calculation and the clustering steps until a complete pass through all the objects results in very small movements of objects or no objects moving from one cluster to another. At this point the clusters are stable and the clustering process ends.

The idea of the algorithm is to place the objects (in our case the exemplars) into different clusters to minimize variability within clusters and maximize variability between clusters. However, a common problem, in the algorithm partitioning, is that if the randomly chosen initial partitions was very wrong then the computation will have a high chance of converging to a local minimum rather than the global minimum solution. The initialisation step is therefore very important. To combat this problem it might be a good idea to run the algorithm several times with different initializations and choose the best clustering. This clustering will have a higher chance of being the global minimum. Even if it is not the global minimum it is very likely that it will be closer to the global minimum, than if the algorithm was run only once. If the results converge to the same partition then it is likely that a global minimum has been reached. This, however, has the drawback of being computationally expensive.

- Alternative K-means clustering: As mentioned in section 4.4.1.2 it is not always possible to compute the mean of objects. Therefore, one can use an alternative version of K-means that does not compute means.

The algorithm works exactly like ordinary K-means (see above), except that it calculates the centroid by first computing the distance from each exemplar $a$ to the one exemplar that is farthest away from $a$ (from hereon called a semi-centroid). This distance is called the maximum distance for $a$. The semi-centroid is the exemplar with the smallest maximum distance, or put more formally:

$$m(C) \leftarrow \arg\min_{c \in C} \arg\max_{c' \in C} d(c, c')$$

where $C$ is a set (cluster) of objects, and $d(a, b)$ is the distance from object $a$ to object $b$. This is also called a minimax solution. We could have chosen other solutions, such as min-i-mean. However, as effective traversing of the tree (see above), relies on a low distance from the semi-centroid to the object, which is farthest away from the semi-centroid, this seems like the best solution.

Like K-means the algorithm may converge to a local minimum rather than a global one. Therefore this algorithm should also be run multiply times.

- Simulated Annealing: Simulated annealing (SA) uses the metropolis algorithm to cluster objects into $n$ clusters, where $n$ is chosen a priori. The algorithm simulates the process of annealing, which is used to cool materials. By cooling them slowly one improves the chance that the materials internal energy will be a global minimum and not just a local minimum.

Other algorithms, like K-means, always go downhill. That is, they always go from one state to another state with lower energy (within cluster scatter). SA can both go uphill and downhill. It does therefore not have as great a risk of being stuck in a local minima.
SA is an iterative algorithm. In the first iteration the chance of moving uphill is large. As the algorithm proceeds the chance of going uphill gets smaller and smaller. This is due to a temperature parameter which gets smaller and smaller. The chance of going from one state to another is calculated as:

\[
\begin{align*}
E &- E' \geq 0 & \quad 1 \\
E - E' < 0 & \quad \exp\left(\frac{E - E'}{T}\right)
\end{align*}
\]

where \(E'\) is the new states energy, \(E\) is old states energy and \(T\) is the temperature.

The problem when using this algorithm is choosing a good cooling schedule and choosing a way of making random based permuting of the clusters. For the permutations, we simply choose a cluster centroid randomly and assign a non-centroid in its place. We do this for two reason, as it is simple and as when we have chosen the centroids we can calculate the best clusters (lowest within cluster scatter), given these centroids, by simply choosing for each exemplar the cluster where it is closest to the centroid. For the cooling schedule we have chosen a linear one, as it is very simple.

We use the exemplars that are to be clustered as cluster centroids in our implementation of SA, and are therefore not using true centroids but semi-centroids (like alternative K-means, see above). Below we have shown the algorithm for our version of SA. \(T_{\text{max}}\) is the starting temperature, \(T_{\text{min}}\) is the ending temperature, and \(T_{\text{step}}\) is the number of iterations.

1. Let \(C\) be a set of \(n\) cluster, where the cluster semi-centroids are chosen randomly.
2. Let \(T = T_{\text{max}}\)
3. Let \(C' = C\)
4. Randomly choose a cluster semi-centroid in \(C'\). Randomly choose an exemplar which is not a cluster semi-centroid in \(C'\). Swap them.
5. In \(C\) assign each exemplar to the cluster semi-centroid it is closest to.
6. In \(C'\) assign each exemplar to the cluster semi-centroid it is closest to.
7. Let \(E\) be within cluster scatter for \(C\).
8. Let \(E'\) be within cluster scatter for \(C'\).
9. Let \(C = C'\) with the following probability:

\[
\begin{align*}
E &- E' \geq 0 & \quad 1 \\
E - E' < 0 & \quad \exp\left(\frac{E - E'}{T}\right)
\end{align*}
\]

10. If \(T \leq T_{\text{min}}\) stop and return \(C\) as the result else let \(T = T_{\text{step}}\) and proceed to step 3
4.4.1.4 Our choice

Above we described different clustering algorithms, namely K-means, alternative K-means, and simulated annealing. We will now compare them and see what is best for our purpose.

The test is based on calculating the within cluster scatter and the between cluster dispersion. We define within cluster scatter as the distance from the semi-centroid to the exemplar, which is farthest away from the semi-centroid, or more formally:

$$S(C_i) = \max_{c \in C_i} d(c_i, c)$$

where $C_i$ is a cluster and $c_i'$ is the semi-centroid for $C_i$. As we want to compute the scatter for a set of clusters, we compute the weighted mean of the scatter for single clusters:

$$S(C) = \frac{1}{\sum_{i=1}^{\left| C \right|}} \sum_{i=1}^{\left| C \right|} S(C_i) * |C_i|$$

where $C$ is a set of clusters, $|C|$ is the number of clusters in $C$, $C_i$ is cluster $i$ in $C$, and $|C_i|$ is the number of members (exemplars) in cluster $i$.

Dispersion for a set of clusters is the weighted mean of the distance from each semi-centroid to its nearest neighbor semi-centroid, or more formally:

$$D(C) = \frac{1}{\sum_{i=1}^{\left| C \right|}} \sum_{i=1}^{\left| C \right|} \min_{c' \in C - c_i} d(c_i', c') * |C_i|$$

where $C'$ is the set of semi-centroids for the clusters in $C$.

Finally to compute quality of a set of clusters we divide dispersion with scatter, or more formally:

$$Q(C) = \frac{D(C)}{S(C)}$$

We have chosen this measure as it directly promote efficient traversing of the tree (by the recognition part), see section 4.4.1.3. A short distance between semi-centroid and the exemplar, which is farthest away from the semi-centroid, will increase the minimal distance (see section 4.4.1.3) and thereby make it more likely that a branch will be removed (that is, no need to descend into that branch). Long distances between semi-centroids and their closest neighbor will decrease the risk that a new exemplar is close to two branches and thereby decrease the risk that one needs to descend into two or more branches.

In figure 4.9 we show the comparison of the three clustering algorithms. Both alternative K-means and ordinary K-means were run with 100 iterations. Simulated annealing had a starting temperature of 200, ending temperature of 0.001, and ran through a 30000 iterations.
The exemplars used in this test, is the same as the ones that is used to construct our final model (see section \[4.7.2.1\]).

As can be seen in the figure, simulated annealing is the worst, regardless of the number of clusters. Alternative K-means is the best when the number of clusters is below five. When the number of clusters is five and above, alternative K-means and ordinary K-means are equally good. Therefore, alternative K-means performs best overall.

By this kind of comparison we only judge the top level of the hierarchy. We do not see this as a huge problem, as the top level is the most important level. It is most important as a problem here will result in problems throughout the hierarchy.

### 4.4.2 Distance measures

In order to cluster the exemplars we need to compare them, see section \[4.4.1\] For this we need a distance measure. The recognition part of the system also needs a distance measure, to find the exemplars in the background subtracted image.

We have investigated the following different distance measures:

- Hausdorff distance measure
- Chamfer distance measure
The model learning part: Pedestrian representation

- Moments-based distance measure

We have chosen the two first as they have successfully been used to compare pedestrian exemplars [70] [36] and the last one as we felt it would be productive to compare the first two measures, which are based on a point to point comparison, to a measure based on statistics.

Before we compare the exemplars we have chosen to normalized them to make them easier to compare. We normalize with regard to planar rotations (in image plane), position, size and mirroring. We normalize the same way for each distance measure, as it will make the measures easier to compare and as it will ease the implementation of the system.

The Hausdorff and Chamfer distance compares distances from points in one exemplar to points in another exemplar. To do this efficiently one uses the distance transform [28]. The distance transform of image $A$ is simply an image with the same size as $A$, and where each point denotes the distance to the nearest feature in $A$. We use the Euclidean distance. By feature is meant a pixel that is on, that is a pixel which equals one. In figure [4.10] are shown an image and its distance transform:

![Image](image.png)

**Figure 4.10:** Untransformed image(left), distance transformed image(right)

In the distance transformed image a point is darker when it is closer to a point in the untransformed image.

When computing the distances from a point $x, y$ in an image $B$ to the closest point in image $A$, one can simply use point $x, y$ of $A$’s distance transform.

In the next section we will describe how the normalization is carried out. Thereafter we will describe and compare the three above listed distance measures.

4.4.2.1 Normalization of exemplars

We normalize with regard to in-planar rotations. Before we explain how this is done the reader should know the following three moments [42]:

$$Centroid_x = \frac{1}{|A|} \sum_{a \in A} a_x * a_d$$
4.4 Structuring the extracted information

\[
\text{Centroid}_y = \frac{1}{|A|} \sum_{a \in A} a_y \ast a_d
\]

\[
\text{Orientation}(A) = \frac{1}{|A|} \sum_{a \in A} (a_x - \text{centroid}_x) \ast (a_y - \text{centroid}_y) \ast a_d
\]

where \(a_d\) is the degree to which point \(a\) is present, \(a_x\) is the x coordinate for \(a\), \(a_y\) is the y coordinate for \(a\) and \(|A| = \sum_{a \in A} a_d\). See section 4.4.2.2 for why a point can be more or less present. These symbols will be used throughout the distance measure section.

If the x-axes grows rightwards and the y-axes grows downwards, \(\text{Orientation}(A) > 0\) will mean that the exemplar primarily goes from top left to bottom right. This means that the pedestrian would have a tendency to fall counter clock wise. \(\text{Orientation}(A) < 0\) means that the exemplar goes from top right to bottom left. This means that it would have a tendency to fall clock wise. When \(\text{Orientation}(A) = 0\) the pedestrian goes neither way, that it is in a stable position. See figure 4.11 for illustration.

![Figure 4.11](image)

Figure 4.11: \(\text{Orientation}(A) < 0\) (left), \(\text{Orientation}(A) > 0\) (center), \(\text{Orientation}(A) = 0\) (right).

We will rotate all exemplars to a stable position. We will achieve this by rotating \(d\) degrees around the centroid. Therefore, we will solve the equation below. For simplicity we write \(\cos(d), \sin(d)\) and \(\tan(d)\) as \(\cos, \sin\) and \(\tan\), respectively. Also for simplicity, we assume \(\text{centroid}_x = \text{centroid}_y = 0\) without loosing generality.

\[
\frac{1}{|A|} \sum_{a \in A} \left( \cos(d) a_x - \sin(d) a_y \right) \left( \sin(d) a_x + \cos(d) a_y \right) a_d = 0 \quad \checkmark
\]
\[
\sum_{a \in A} \left( \cos a_x - \sin a_y \right) \left( \sin a_x + \cos a_y \right) a_d = 0 \quad \checkmark \text{ (as } |A| \ast 0 = 0)\]
\[
\sum_{a \in A} \left( \cos \left( a_x^2 - a_y^2 \right) + a_x a_y \left( \cos^2 - \sin^2 \right) \right) a_d = 0 \quad \checkmark
\]
\[
\cos \sin \sum_{a \in A} \left( a_x^2 - a_y^2 \right) a_d + \left( \cos^2 - \sin^2 \right) \sum_{a \in A} a_x a_y a_d = 0 \quad \checkmark
\]
\[
\frac{\sum_{a \in A} \left( a_x^2 - a_y^2 \right) a_d}{\sum_{a \in A} a_x a_y a_d} = 0 \quad \checkmark
\]
\[
\frac{\sum_{a \in A} \left( a_x^2 - a_y^2 \right) a_d}{\sum_{a \in A} a_x a_y a_d} = \frac{-\cos^2 + \sin^2}{\cos \sin} \quad \checkmark
\]

We let \(b = \frac{\sum_{a \in A} \left( a_x^2 - a_y^2 \right) a_d}{\sum_{a \in A} a_x a_y a_d}\) then:
\[ b = \cos^2 + \sin^2 \]
\[ b = \cos \sin \cos \]
\[ b = \sin^2 \tan \]
\[ 0 = \tan^2 - b \tan - 1 \]

Now we set \( x = \tan \). We then have an ordinary quadratic equation of the form \( ax^2 + bx + c \), where \( a = 1 \), \( b = b \), and \( c = -1 \). We can then calculate the discriminant \( d \) as:

\[ d = b^2 - 4ac = b^2 - 4 \times (-1) = b^2 + 4 > 0 \]

As \( d > 0 \) we know that there will always be two solutions and these are:

\[ \frac{-b \pm \sqrt{d}}{2a} = \frac{-b \pm \sqrt{b^2 + 4}}{2} \]

We therefore have that:

\[ x = \frac{-b \pm \sqrt{b^2 + 4}}{2} \]
\[ \tan(d) = \frac{-b \pm \sqrt{b^2 + 4}}{2} \]
\[ d = \arctan \left( \frac{-b \pm \sqrt{b^2 + 4}}{2} \right) \]

As the quadratic equation has two solutions and as arcus tangens also has two solutions, we have \( 2 \times 2 = 4 \) solutions. These four will be located 90 degrees apart. We choose the one which requires the least rotation as it is unlikely that large rotations have occurred for pedestrian objects.

To normalize with regard to size, we scale the exemplars so that they all have the same height. We scale, in such a way, that the images keep their aspect ratio. The width is however different for each exemplar. When the distance measure requires the exemplars to be of equal width, we fill the smallest with blackspace (zeros) so that it matches the largest width.

We could have chosen to scale independently of the axes. This would handle some view dependent shape variation [24], and could therefore lead to fewer necessary exemplars in the model. Furthermore, axes independent scaling could handle some variations among different persons, such as height and width. However, if the learning part scales independent of the axes then the recognition part must also do so. If this is not the case the recognition part will see the model as missing some exemplars.

Therefore we have chosen to not scale independently of the axes. Instead we can either use more exemplars to handle view dependent shape variations or we can use a scale independent distance measure. Scale independent distance measures can for example be \( \text{centroid}_x / \text{breadth} \) or \( \text{centroid}_y / \text{height} \). However, scale independent distance measures will suffer from the same problems as scale independent normalization, namely that the recognition part also must use a scale independent distance measure.
To normalize with regards to position we translates the exemplar images so that they all have their centroids located over the same point. This may not always be the location that minimizes the distance between two exemplars. However, the more similar the exemplars are the closer their centroids will be. Dissimilar exemplars may have centroids far from each other, which could result in that two exemplars in stead of being far apart, they will be very far apart. We do not see this as a problem, as they in neither case go in the same cluster. Therefore, it seems reasonable to use this point for position normalization.

To validate that similar exemplars in fact have close centroids, we have clustered the exemplars both with the alternative K-means (see section 4.4.1.3) and by an algorithm, which randomly allocates exemplars to clusters. For each clustering, we compute the variance of the centroids within each cluster. This variances are then used to compute a weighted average (based on the number of elements in each cluster). In figure 4.12 is shown the results of these computations. We use the silhouette Chamfer distance for the alternative K-means clustering.

The exemplars used in this test, are the same as the ones that are used to construct our final model (see section 4.7.2.1).

As seen the K-means clustering produces clusters with a lot more similar centroids than the random clustering, and therefore our hypotheses seems to be correct. Furthermore, the larger the number of clusters the smaller the variance of the centroids (for K-means), which also seems to suggest that our hypotheses is correct as a larger number of clusters will lead to clusters where the exemplars are more similar. This does not happen for the random clustering as its clusters will not have more similar exemplars when the number of clusters increases.

The mirror normalization is handled early when the exemplar is extracted. See section 4.3 for further explanation.

As the normalization handles in-planar rotations, position, and scaling (which keeps aspect ratio) we will, in the following sections, not require that our distance measure does the same.
4.4.2.2 Requirements for distance measures

In section 4.1 we identified a number of requirements that our model should handle. We will compare the listed distance measures in section 4.4.2 to these requirements, except for the following requirements:

1. The model should handle articulated moving parts.
2. The model should handle variations among individuals.
3. The model should handle planar rotation (in image plane).
4. The model should handle scaling which keeps aspect ratio.
5. The model should handle position.

The first as a consequence of choosing exemplars to represent the model we cannot handle articulated body parts within one exemplar, this must be handled by having an exemplar for each stance. The second as shape variations among individuals are handled by the clustering algorithm. The third, fourth, and fifth as these are handled by the normalization, see section 4.4.2.1.

We will still look at handling of scaling independently of the axes, as the distance measure should not handle this kind of scaling perfectly, see section 4.4.2.1. If our distance measures did handle axes independent scaling, then it will require that the recognition parts distance measure (or their normalization) also handle axes-independent scaling. Our distance measure may therefore not handle axes independent scaling. We will handle this kind of scaling by using multiple exemplars for the same stance (but different views and ratios of breadth/width of objects).

We have further identified a number of requirements unique to the distance measure:

- True distance measure, that is it should be a metric.
- Same distance measure in both the model learning and the recognition part of the system.
- Efficient - inexpensive to calculate.
- Handle fuzzy data.
- Works with both silhouettes and contours.

A metric means that the distance measure should satisfy the four following rules [70]:

1. \( d(A, B) \geq 0 \)
2. \( d(A, B) = 0 \) if and only if \( A = B \)
3. \( d(A, B) = d(B, A) \) (symmetry)

4. \( d(A, B) + d(B, C) \geq d(A, C) \) (triangle inequality)

These rules comprise what is intuitively understood as a distance measure. That is, a distance cannot be negative and is only zero when the distance is to and from the same spot. Furthermore, the distance from \( A \) to \( B \) and from \( B \) to \( A \) is the same. Finally, going from a \( A \) to \( C \) is no longer than going from \( A \) to \( C \) via \( B \).

These are more than intuitively good requirements, as at least one technique requires the two first metric rules to ensure its performance [70]. Furthermore, when traversing a hierarchy of exemplars it will be a great advantage if the last metric rule is fulfilled, see section 4.4.1.3.

It would be an advantage to use the *same distance measure* in both the model learning and recognition part of the system. When constructing clusters in the model learning part of the system, a cluster mean is chosen. This cluster mean should, also in the recognition part of the system, be close to all the exemplars in the cluster. This can be guaranteed if we use the same distance measure in both parts. This means that requirements, for the distance measure, that may not seem important for the model learning part of the system, should still be considered as we want to use the same distance measure for both parts of the system.

The measure should be *efficient* both when comparing two exemplars in the model learning part of the system, and when matching an exemplar to a hole image in the recognition part of the system. It is however, most important for recognition part of the system as this part will be running in real time.

Exemplars are usually represented binary. Either a point is there or it is not. However, sometimes it will be necessary to compute the mean of two or more exemplars. For example K-means (see section 4.4.1.3) and many other clustering algorithm computes the mean of two or more objects.

We compute the mean exemplar as:

\[
M_{x,y} = \frac{1}{|A|} \sum_{i=1}^{|A|} A_{i,x,y}
\]

where \( M_{x,y} \) is point \( x,y \) for the mean exemplar, \( A \) is the set of exemplars to average, \(|A|\) is the number of exemplars in \( A \), \( A_{i,x,y} \) is the value of the \( i \)’th exemplars \( x,y \) pixel.

We do not compute the distance transform (see section 4.4.2) from this mean, in stead we compute an average of each exemplars distance transform, as

\[
D_{x,y} = \frac{1}{|D|} \sum_{i=1}^{|D|} D_{i,x,y}
\]
where \( D_{x,y} \) is point \( x,y \) for the distance transformed mean image, \( D \) is the set of distance transformed images, \(|D|\) is the number of distance transformed images in \( D \), \( D_{i,x,y} \) is the value of the \( i \)’th distance transformed image \( x,y \) pixel.

As can be seen this will lead to points being present to a degree between zero and one inclusive. Therefore it would be an advantage if the distance measure could handle fuzzy exemplars.

Note that it only makes sense to calculate the mean for silhouettes, as quite similar contours can have very few points in common due to the low number of points they consist of. Therefore, we do not make mean exemplars of contours.

We would like our distance measures to be able to both work with silhouettes and contours, as this added flexibility will give us a wider choice when implementing our system. We would also like to compare and see what works best.

We will, in the following sections, investigate the above listed distance measures and see how they compare to the above mentioned requirements.

### 4.4.2.3 Hausdorff distance

The Hausdorff distance has previously been used to locate exemplars in images before, see [44] [70] for uses of Hausdorff distance and [38] gives a nice introduction to this distance measure. We have also considered this distance measure for our system. In this section we will first describe the Hausdorff distance in general terms. Then we will describe how it is used when comparing exemplars. Finally, we will describe how the Hausdorff distance meets the above mentioned requirements.

The Hausdorff distance is used to compare two sets. The comparison is done by the following formula:

\[
h(A, B) = \max_{a \in A} \left\{ \min_{b \in B} \{ d(a, b) \} \right\}
\]

where \( d \) is a distance function between members of set \( A \) and set \( B \). That is, for each member in \( A \) we find the closest member in \( B \). Then we choose the longest of these distances.

However, as this measure is not symmetric and therefore not a metric, a symmetric Hausdorff distance is often used which is defined as:

\[
H(A, B) = \max\left( h(A, B), h(B, A) \right)
\]

The Hausdorff distance is a true metric and can be used with both silhouettes and contours. It is not robust towards view dependent shape variations which is good, see section [4.4.2.2], as a single outlier can disturb the measure badly. Furthermore, changes in view can have a dramatic effect on the exemplar image, which also suggest it is not robust towards view dependent shape variations. However, it also has some disadvantages. The measure is not robust towards noise.
or partial occlusion. And we suspect it is quite bad at handling these kinds of robustness, as a single outlier can disturb the measure badly. It can not handle fuzzy exemplars and it is expensive to calculate. We suspect that it will be poor in the third part of the system due to its lack in robustness.

4.4.2.4 Chamfer distance

The Chamfer distance has been previously used to detect exemplars in images, see [69] [36] for uses of the Chamfer distance. This distance measure is very similar to the Hausdorff distance. It also compares two sets for equality. However, instead of using the maximum it uses the mean. More formally,

\[ c(A, B) = \frac{1}{|A|} \sum_{a \in A} d(a, B) \]

where \(|A|\) denotes the number of elements in set \(A\), and \(d(a, B)\) is the distance from member \(a\) to the closest member in set \(B\). Like the simple Hausdorff distance, this is not a symmetric measure. Also similar to the Hausdorff distance a symmetric measure can be constructed from the non-symmetric one, as:

\[ C(A, B) = \max(c(A, B), c(B, A)) \]

The chamfer distance also works with fuzzy data. Instead of using an ordinary mean, we simply use a weighted mean. The degree to which a point is present, is used as the weight. See formula below:

\[ c(A, B) = \frac{1}{|A|} \sum_{a \in A} d(a, B) \cdot a_d \]

where \(a_d\) is the degree to which point \(a\) is present and \(|A| = \sum_{a \in A} a_d\).

We suspect that the Chamfer distance will be relatively robust towards noise and partial occlusion, as a few outliers will not have a huge effect upon it. We suspect that it will be able to handle some view dependent shape variations, however as this kind of shape variations can dramatically change an exemplar we suspect that it will only handle small changes in view. It is able to work with both silhouettes and contours. And it can be used with fuzzy exemplars. However, it also has some disadvantages. The Chamfer distance is not a true metric as it does not fulfill the triangle inequality, see appendix [F]. It is expensive to calculate. We suspect that it will be reasonably good in the third part of the system, as it can handle some robustness. However, it will be somewhat inefficient when traversing the tree of exemplars as it does not fulfill the triangular inequality, see section 4.4.1.3.
4.4.2.5 Moment based distance

Instead of comparing the points directly we can calculate some statistical measures, called moments [42], for each exemplar and compare these moments instead of comparing the exemplars directly. Below is a list of the moments we use:

- Variance
- Number of pixels

The number of pixels is simply $|A|$. Variance is calculated as:

$$Variance_x = \frac{1}{|A|} \sum_{a \in A} ((a_x - Centroid_x) * a_d)^2$$

$$Variance_y = \frac{1}{|A|} \sum_{a \in A} ((a_y - Centroid_y) * a_d)^2$$

To compute a distance between two exemplars we compute the sum of the absolute differences of the moments, as shown:

$$d(E_1, E_2) = \sum_{i=1}^{\text{moments}} \left| E_1^i - E_2^i \right|$$

where $E_j^i$ is exemplar $j$'s $i$'th moment and $\text{moments}$ is the number of moments. However, as some of these moments vary a lot more than others, some moments will have a too high influence on the result. Therefore, before we calculate the distance, we divide a moment by the variance for this moment. All the exemplars, to be part of the model, are used to calculate this variance. And the formula finally looks like:

$$d(E_1, E_2) = \sum_{i=1}^{\text{moments}} \left| \frac{E_1^i - E_2^i}{\text{var}_i} \right|$$

where $\text{var}_i$ is the variance for moment $i$.

We suspect that the moment based distance will be relatively robust towards noise and partial occlusion. We suspect this as it is based upon statistics, and therefore a few outliers should not have a huge effect upon it. We suspect that it will be able to handle some view dependent shape variations, however as this kind of shape variations can dramatically change an exemplar we suspect that the it will only handle small changes in view. It is able to work with both silhouettes and contours. And it can be used with fuzzy exemplars. It is inexpensive to calculate, as the moments only needs to be calculated once for each exemplar and can then be cheaply compared to other exemplars moments. The moment based distance is not a true metric as it does not
fulfill the \(d(A, B) = 0\) if and only if \(A = B\) rule. We suspect that it will be reasonably good in the third part of the system, as it can handle some robustness and does fulfill the triangular inequality (which is eases traversing the tree of exemplars, see section 4.4.1.3).

### 4.4.2.6 Our choice

We have investigated three different distance measures, and must choose one of them to use for clustering. We have compared the three to the requirements listed in section 4.4.2.2. The Hausdorff distance is the only one which fulfills all the metric rules, however it is not robust as it bases everything on one pixel/comparison (the maximum). We also intuitively feel that this is not good. The moment based distance seems to fulfill these requirements best, as it only have one major drawback, namely that is not a true metric. However, the moment based distance is not a clear winner, as its fulfillment of the requirements is to some degree based upon speculation (for example robustness requirements).

We have therefore made another comparisons based upon the crispness of average exemplars. We cluster the exemplars and compute the mean exemplar (see section 4.4.2.2) in each cluster. We then compute the crispness of a mean exemplar as:

\[
c(M) = \frac{1}{\text{Breadth} \times \text{Height}} \sum_{x=1}^{\text{Breadth}} \sum_{y=1}^{\text{Height}} 1 - 2 \times \begin{cases} M_{x,y} \geq 0.5 & \text{if } M_{x,y} < 0.5 \\ 1 - M_{x,y} & \text{else} \end{cases}
\]

where \(\text{Breadth}\) is the width of the exemplar and \(\text{Height}\) is the height of the exemplar.

When clustering the result is not one cluster, but two or more clusters. We therefore make a weighted mean of the mean exemplars for each clustering as:

\[
C(\text{Clustering}) = \frac{1}{|\text{Exemplars}|} \sum_{i=1}^{|\text{Cluster}|} \frac{c(M_i)}{|\text{Cluster}_i|} \times |\text{Exemplars}_i|
\]

where \(|\text{Exemplars}|\) is the total number of exemplars, \(|\text{Cluster}|\) is the number of clusters, and \(|\text{Exemplars}_i|\) is the number of exemplars in cluster \(i\). We would, of course, like to get as crisp exemplars as possible.

We have made two comparison for each measure to see what works best, the silhouette or the contour. See figure 4.13 for the comparison.

Then when we have compared the three measures. We are using the silhouette for all three. For Hausdorff and moment based -distances we use silhouettes because they got the best results, as seen in figure 4.13. For the Chamfer distance the results are very similar. However, we are using the silhouette for the Chamfer distance, as this will still give us the possibility to use mean exemplars when clustering (we have not defined how to make mean contour exemplars). See figure 4.14 for the comparison.
Figure 4.13: Hausdorff, Chamfer, and Moment Based -distances. A contour(solid) versus silhouette(dash) comparison.

Figure 4.14: Hausdorff(solid), Chamfer(dash), and Moment Based(dot slash) -distances.

The exemplars used in this test, is the same as the ones that is used to construct our final model (see section 4.7.2.1).

There is no guaranty that the measure with the highest crispness, will also be best for clustering. However, it is an indication that it is the best measure for clustering, as it indicates the measure is better at grouping similar exemplars.

We have chosen the Chamfer distance using silhouettes as it does have higher crispness than Hausdorff and moment based -distances, it supports fuzzy exemplars, and as it has some ro-
4.4 Structuring the extracted information

bustness against noise and occlusion. Compared to the other measures it has two drawbacks, namely that it does not fulfill the triangular inequality and it is expensive to calculate. The last point is only an advantage compared to the moment based distance, as the Hausdorff distance is also expensive to calculate.

However, as the model learning part is running off-line it is not a major concern that it is expensive to calculate. Though it could be a problem if the recognition part wants to use the same measure, as this part of the system should run in real-time. But we do not know which measure the recognition part will be using, and if we knew we would probably use the same measure for clustering.

Regarding the triangular inequality, we again would like to point out that we do not know which measure the recognition part is using, and it only makes sense to see this as a major requirement if we are using the same measure. Furthermore, we can still calculate the maximum distance from a cluster mean to one of its members using the Hausdorff and moment based -distances, and include this knowledge in the model.

We do not claim that these drawbacks are insignificant, only that advantages of the Chamfer distance outweighs them.
4.5 Motion model

To make the recognition of exemplars as reliable as possible it can be useful to have information about which stances are likely to follow one another. As this will make it possible for the recognition part to make an informed guess about which stance to expect in the next image in the video (image sequence).

As an example we show five images in figure 4.15, and it is much more likely that they will occur in the order from left to right than in a random order of the images.

![Figure 4.15: Order of exemplars.](image)

We have chosen a simple model, namely a first order Markov model (MM) [74]. We can use this model as the exemplars are already clustered, that is the states (each cluster is a state) are known. It only models the probability of going from one state to another. We could have chosen a more complex model such as hidden Markov model (HMM) [74], that both places the objects in clusters (states) and models the probability of going from one state to another.

But choosing a HMM would require that our objects (exemplars) would be in a vector space. Furthermore, we would have to have explicit knowledge about the dimensions to compute covariance matrices which is needed in the hidden Markov model. Exemplars are not normally in a vector space, nor is there explicit knowledge about the dimensions. However, by using moments (see section 4.4.2.5) to describe the exemplars we would have the exemplars in a vector space and could therefore use most techniques to model the motion, but this would restrict us to using moment based distances. Therefore we have chosen to use a simpler model, with less requirements, namely the first order Markov model.

The MM works by calculating transition probabilities $t_{ij}$ for each state $i, j \in S$, where $S = 1, ..., s$ and $s$ is the number of states. We do this by making a histogram of transitions, by the following algorithm:

1. let $a_{ij} = 0$, for all $i, j \in S$.
2. for each cluster (state) $C_n$ do:
   
   (a) for each exemplar $e \in C_n$, find the exemplar that were recorded right after $e$. We use series and frameNumber to do this, see section 4.3.
   
   (b) if the next recorded exemplar $f$ is found increase $O_{nj}$ by one, where $j$ is the cluster where exemplar $f$ is found. If the next recorded exemplar is not found, do nothing.
3. let $o_i = \sum_{j \in S} o_{ij}$

4. for all $i, j \in S$, let $t_{ij} = \begin{cases} o_i > 0 & \frac{o_{ij}}{o_i} \\ o_i = 0 & -1 \end{cases}$

Now $t_{ij}$ will be the likelihood of going from state $i$ to $j$. If $t_{ij} = -1$ it means that there were no data to calculate the state transition from. It can occur if all exemplars in a cluster were the last in a series, see section 4.3 about extraction of exemplars.

We model each level in the hierarchy as its own MM. It is then the recognition parts choice which MM it wants to use. Using a model close to the root level will have more exemplars in each state and therefore more reliable numbers. On the hand it will not give as specific knowledge about which exemplars are likely to occur, as there are many exemplars in each group. MM closer to the leaf levels will have the advantage of specific knowledge, but it will be less reliable.

We have not tested the quality of the generated motion models, as the recognition part of the system is not implemented yet. We could use the recognition part to validate the motion model, by testing how well the motion model was at predicting which exemplar would appear in the next frame given the exemplar in the current frame.
4.6 Interface to recognition part of the system

Our system is split into three different parts, namely the pixel-based background subtraction, the model learning, and the recognition part. We need to transfer data between the different parts. In this section we will describe what is transferred to the recognition part. How it is transferred is described in appendix E.4.

The following elements are transferred:

- Exemplar silhouettes and contours. All exemplars which has been used in the constructing of the model is stored as both silhouettes and contours.

- Mean exemplars. All means in the hierarchy are stored. Here we do not think about centroids, but true means, see section 4.4.2.2 for how exemplar means are calculated. Only the silhouettes are stored here, as we have defined no way to calculate means of contours.

- Hierarchy structure. That is which exemplars are in which branches.

- Semi-centroids. For each node its semi-centroid are stored.

- Max distances. For each node in the hierarchy the maximal Hausdorff (contour) and the maximal moment based (silhouette) are stored.

- Motion model. For each level in the hierarchy the chance of going from one cluster to another is stored. Furthermore, the amount of data this calculation is build upon is stored.
4.7 Our pedestrian model implementation and experiments

In the following sections we will describe our software to construct a pedestrian model and our experiments with this software. We will start by describing what we have implemented and then describe our testing. The description of how to execute the software can be found in appendix E.

4.7.1 Pedestrian model implementation

To test the different model learning techniques (for example: distance measure, clustering, ...) we have implemented a lot of code. This was done almost exclusively in Matlab, except for a single function which finds the exemplar in a background subtracted image (see CDROM one in directory `matlab/sh/objectDetection/exemplarExtraction/ndFirstPixel.cpp`) which for efficiency reasons were implemented in C++ as a Matlab function. We used Matlab as we find it to be a prototyping environment, where it is possible to develop software fast.

We implemented the following clustering techniques:

- Ordinary K-means
- Modified K-means (see section 4.4.1.3)
- Simulated annealing
- Hierarchical clustering

Furthermore, we implemented the following distance measures:

- Hausdorff distance
- Chamfer distance
- Moment based distance

In additions to this, we developed the following:

- Normalization of exemplars
- Exemplar extraction from a background subtracted video sequence
- Markov modeling for order of clusters
- Similarity matrix construction
- Functions to store the model on a hard disk

Finally, we have developed a number of minor functions to display exemplars, show animation of exemplars, and other useful help functions.
4.7.2 Test results of our pedestrian model

In this section we will present the models, which our work resulted in. First we will explain how we captured and extracted exemplars from these movies. Secondly, we will explain how we used these exemplars to construct our models.

4.7.2.1 Movie recording and exemplar extraction

To extract exemplars we need to record a few movies. In this section we will describe how we did this and the result of extracting exemplars.

We used the Sony DFW-VL500 camera to record these movies, as it was the best camera available to us.

We recorded nine movies of three different pedestrians, that is three movies per pedestrian. One where the person walked along the camera’s optical axes, one where the person walked perpendicular to the camera’s optical axes, and the last movie where the angle between the person’s line of walk and the optical axes were approximately 45 degrees.

For practical reasons (the camera had to be attached to a stationary computer) the movies were recorded from a window. This also means that the movies were recorded from a birds perspective.

The movies were all recorded in the same outdoor scenario. However as they were recorded on different days, two of them has a lot of snow in the background and the last one has a little snow in the background (the snow was only on the ground). The persons walked on a dirt road and on some fairly high grass. In all movies there is very little moving background, which helps in making the movies easier to background subtract.

There were two males and one female in the movies. The pedestrians were all dressed in clothes, which were not too similar to the background. However, the female pedestrian had long hair, which was sometimes similar to the background.

The male pedestrians were dressed in clothes, which did not change their silhouettes dramatically, except for making the silhouettes larger. The female wore a long jacket, which did change her silhouette. The jacket made her legs seem a lot shorter than they actually are. We do not see this as a problem, as the model should reflect different peoples clothes.

These conditions makes it possible to perform pixel-based background subtraction fairly well. The only problem was the female actors hair, which however was only a problem in some frames.

The movies are located on CDROM one in the directory 'movieCaptures/050104' and on CDROM two in the directory 'movieCaptures/080105'.

As explained in section 3 and section 4.3 we background subtracted these movies. The results are on CDROM one in the directories 'movieBackgroundSubtracted/bgs050104' and 'movieBackgroundSubtracted/bgs080104'.
4.7 Our pedestrian model implementation and experiments

We extracted exemplars from the above movies as explained in section 4.3. We got 647 exemplars from these movies. These are the exemplars which are used throughout the thesis for testing and for building our model.

4.7.2.2 Clustering

We have already tested our clustering methods (see section 4.4.1.4) and our distance measures (see section 4.4.2.6). Furthermore, we have chosen which clustering method and which distance measure to use (see above sections). We will in this section describe the result of constructing the final model.

To construct the model we must choose the width (maximal number of children for each node) and depth of our hierarchy. As can be seen from figure 4.9 three exemplars seems to be the optimal width, as it scores the highest dispersion vs. scatter value. However, this is no guarantee that it is in fact the optimal width size for the recognition part. Only an indication thereof. Therefore, we have made hierarchies with widths of 3, 4, 5, and 6.

We choose a high depth for our model, as the recognition part can simply go as deep as they wants to. It should not harm the recognition part that the hierarchy is deeper than it needs to be for their purposes. Particular we use a depth of 7, 6, 5, and 5 for a width of 3, 4, 5, and 6 respectively.

We used the Chamfer distance measure on silhouettes when constructing this model, as we learned from section 4.4.2 that it was the best choice. We used the ‘dispersion versus max scatter’ quality measure to optimize our clustering, as it directly promote efficient traversing of the tree (see section 4.4.1.4).

In figure 4.16, 4.17, 4.18, and 4.19 are shown the four highest levels of the different hierarchies.

For the model with width 3 a single image is shown. It depicts the hierarchy from the root and spans four levels. For the rest of the models, more than one image are shown. For these, the leftmost image is the hierarchy depicted from the root and spans two levels. The non-leftmost images are the hierarchy seen from direct children (not grandchildren) of the root. All the non-leftmost images span three levels.

The created models are stored on CDROM one in the directory 'Models'. All the relevant information is stored, including the hierarchy, the motion model, the exemplar silhouettes / contours, and the mean exemplars. See section 4.6 for what is stored.

We have visually inspected these models and we think that they are reasonable at grouping similar looking exemplars in the same branches (clusters). It also seems that the hierarchies would support traversing by not having exemplars at lower levels, which are very dissimilar to their fathers or grandfathers. Of cause the closer one gets to the top, these similarities seem to lessen, as the means shown represent a lot more exemplars at the top than at the bottom.

However, there are exemplars in the model which should not have been their. This is most clear in the second picture from the left in figure 4.18. Some of these exemplars do not seem to have been mirror normalized properly. The exemplars which do not seem to have been mirror
normalized properly are the five leftmost at the leaf level and their father. We are not sure why these are not been dealt with properly. These types of un-mirror normalized exemplars can also be found at figure 4.19. They are not seen in figure 4.16 and 4.17 this is probably due to the lower number of exemplars these figures show.

The largest problem with these video sequences is their low number. It means that there is a too small number of different clothes and too little differences in body shape. Generally, it means that the variations are too small.

We have made no quantifiable test here, as we have already made quantifiable tests for the clustering and distance measure -algorithms (see section 4.4.1.4 and section 4.4.2.6). Furthermore, the best way to make such a test would probably be to see how well these models served the purpose of pedestrian recognition / tracking. However, this is beyond the scope of this project.
Figure 4.16: Hierarchy with width 3 and depth 7 (only 4 shown)
Figure 4.17: Hierarchy with width 4 and depth 6 (only 4 shown). From left to right. Root (2 levels), 1st child of root (3 levels), 2nd child of root(3 levels), 3rd child of root(3 levels), 4th child of root(3 levels).
Figure 4.18: Hierarchy with width 5 and depth 5 (only 4 shown). From left to right. Root (2 levels), 1st child of root (3 levels), 2nd child of root (3 levels), ..., 5th child of root (3 levels).
Figure 4.19: Hierarchy with width 6 and depth 5 (only 4 shown). From left to right. Root (2 levels), 1st child of root (3 levels), 2nd child of root(3 levels), ..., 6th child of root(3 levels)
Chapter 5

Conclusion

In this chapter we will conclude our work. We will start by summarizing the project by answering the questions we made in the problem statement. The questions will be answered in the same order as they were listed in the problem statement. Then we will describe some ideas for improvements to our work. We will finish this thesis by describing our general observations on background subtraction.

5.1 Summary

The initial segmentation part of the system: pixel-based background subtraction

- What is pixel-based background subtraction? Generally, background subtraction is a fundamental technique in the computer vision field. It is the process of segmenting the background from the foreground in an image. The technique can be used for different applications, and it is often used in the context of people identification and tracking.

There are different approaches to pixel-based background subtraction, some are based on statistical texture properties, some are based upon image motion where the background is assumed to be stationary, and some are based upon geometry. Unfortunately each approach has its limitations. Our pixel-based background subtraction algorithm uses the first approach, and can handle non-stationary background to some degree, since we use the non-parametric model.

The basic approach for pixel-based background subtraction consists of three operations: Threshold selection, background modeling, and subtraction operation or pixel classification. This basic approach is used by many researchers, but several serious problems have not been addressed by most of them. These problems include slow moving or stationary objects, non-stationary backgrounds, moving camera, etc.

The computer vision field has addressed three levels of pixel-based background subtraction: the pixel level, the region level, and the frame level. Some of the implemented background subtraction systems focus only on the pixel level, where other more advanced
algorithms make use of the region and the frame levels, which helps them in addressing some of the difficult problems which belong to the basic approach.

- **Which basic general models are related to and used for background subtraction?**
  The normal distribution is an important result from the field of statistics, which conforms to the sample distribution of many phenomena. Normally distributed phenomena share statistical properties, which makes them attractive for analysis. It is generally reasonable to model pixel intensity values as normally distributed. This can be done using the univariate density function for gray scale pixels, or by using the multivariate density function for color pixels. In both cases a threshold is used to determine which pixels are part of the background and which are not. Exponential forgetting is used to control how fast old pixel values are forgotten from the model of the background. In the implementation, exponential forgetting can be used as a replacement for a simple mean calculation.

- **Which advanced general models do exist for background subtraction?** One of the severest problems with the basic models for background subtraction was that they could not handle non-stationary background. We therefore investigated and found several models which could.

  We investigated both pixel and region based models. The pixel based was the non-parametric and the mixture model. We have also implemented the non-parametric model. We also investigated a model which worked at the region level. This model considered a neighborhood of pixels to see if some part of a background object had moved slightly.

  As so often, our experiments suggested that a mixture of region and pixel-based models was better than just using one of them alone, as they would gain the advantages of both.

- **Which general techniques are used in different general background subtraction models?** A few techniques can successfully be applied to models used for pixel-based background subtraction. One is the use of chromatic colors, which makes the background model less sensitive to sudden illumination changes. Another technique which is advantageous is if the background model is able to adapt itself incrementally over time. This is known as incremental updating and takes care of the problem of slow illumination changes, or more generally, it takes care of changes in the background over time. An example of a slow illumination change is the sunlight during the day. Selective update is a technique used to avoid static or slow moving foreground objects from becoming part of the background. The technique is used in conjunction with incremental update. Instead of updating every pixel, only those which are already marked as background are updated. The technique is not without side effects, but these can be resolved by occasionally giving all pixels a chance of being updated (blind update).

- **How can the shortcomings of traditional and modern algorithms be remedied?** A good background subtraction system should solve all the problems described under section 3.1.1.2. However, some of these problems can not be solved perfectly simultaneously, because some of them needs semantic understanding of motion of foreground and background. Semantic understanding is impossible to accomplish if one has no information about the final purpose. Therefore background subtraction in itself should be goal oriented. A general segmentation system that can identify and classify different objects
in dynamic scenes is today very difficult to build. This is one of the important reasons that model-based background subtraction systems are more efficient than the pixel-based systems, since the model-based applications know beforehand the model or at least some properties of the interesting objects, which should be subtracted from the rest of the scene.

- **Which filtering techniques can be used to improve the result of background subtraction?** We have described three different filters, which are the mean and the median filters, and dynamic expanding filter, which was developed in the context of this thesis. The first two were chosen because of their popularity in computer vision and image processing fields. However, we have developed the dynamic expanding filter, because we it appears to be better at removing noise from the result of the background subtraction process.

**The model learning part of the system: Pedestrian representation**

- **Which requirements should our object model possess?** We investigated different related approaches to object modeling. Our observation is that a good object model must support six requirements: 1) It should not depend on scale, orientation, and position of objects. 2) It should be robust towards noise. 3) It should be robust to partial occlusions of an object. 4) It should allow for articulated moving parts. 5) It should be efficient towards disk and memory space. 6) It should support efficient shape recognition.

- **How can we represent the object of interest (pedestrian)?** We described three different model types, which has been used for object representation in computer vision field: 1) 2D models based on object general properties, 2) 2D explicit models of object shape, and 3) 3D models of object shape. Our observation is that a 2D explicit model of an object shape is the best solution for our system.

We investigated different techniques for representing the object of interest (pedestrian). These techniques are active contours and exemplars. We found out that exemplars are more applicable for our system. Therefore, we describe pedestrians shapes with exemplars.

- **How can we extract information to construct the model from a video sequence with minimal human intervention?** The way we extract the information (exemplars) from a video sequence is as following: 1) We start by filtering the exemplars. We use dynamic expanding filter to remove noise without hurting the exemplars, and we use median filter to clean up remaining noise especially near the edges of the exemplars. We also use n-connected filter to remove the noise from the rest of the images, especially at the corners. 2) The filtered exemplars are then handed over to an algorithm, which is designed to remove the outliers from the filtered exemplars, without removing too many wanted exemplars.

- **How can we structure the extracted information to make it easily and efficiently accessible?** We structure the extracted information (exemplars) by clustering. We investigated two different clustering methods, which are called hierarchical and nonhierarchical
clustering. We chose a combination of these two methods because we need a structure (hierarchy) of the exemplars to make it easier for the recognition part to access them, and we also need the effectiveness of the nonhierarchical methods.

We experimented with three nonhierarchical clustering algorithms to find out which one performs best. These algorithms are K-means, alternative K-means, and simulated annealing. The results show that one algorithm performs best in one situation, whereas another performs better in other situation, however our observation is that alternative K-means performs best overall, and is therefore the chosen algorithm for our system.

We also investigated different distance measures, which clustering algorithms needs for comparison of the exemplars to group them. Before we compare the exemplars we normalize them to make them easier to compare. We normalize with regard to planar rotations (in image plane), position, size and mirroring.

The distance measures we experimented with are: Hausdorff distance measure, Chamfer distance measure, and Moments-based distance measure. We compared these three distance measures, and we observed that there is no clear winner, however we chose the Chamfer distance as it has more advantages than the other distance measures.

- **How can we describe the likelihood of going from one stance to another?** To describe the likelihood of one stance following another, we can use a motion model. We chose a simple motion model called first order Markov model. The model is sufficient to calculate the probability of which stance (exemplar) is likely to occur in the next input image. These probabilities are useful for the recognition part of the system.

- **How should the model be delivered to the recognition part of the system?** Our model learning part of the system transfers six elements to the recognition part of the system. These are: 1) Exemplar silhouettes and contours, 2) Mean exemplars, 3) Hierarchy structure, 4) Semi-centroids, 5) Max distances, and 6) Motion model. These information are stored on the hard disc as ordinary files.

## 5.2 Future work

### 5.2.1 Improvements for the initial segmentation part of our system

Our pixel-based background subtraction algorithm is reasonably good at separating the foreground from the background. However, it could be improved in a number of ways. In the following we will describe some ideas of how this could be accomplished.

#### 5.2.1.1 Improvement of shadow elimination

We see two ways the shadow elimination could be improved. First we could look at other image features than color intensity values, such as edges. Some edges can be detected as being an edge between a shadowed and non-shadowed area. However, edges are not perfect as it can
be hard to differentiate between different types of edges. They can still however help indicate which parts of a frame are shadowed. Other image features may be equally interesting to look at. In [45] multiple ways of detecting shadowed areas is described.

Secondly, we could investigate the RGB color values and see if they increase / decrease proportionally when lighting conditions change. If they do not, and their increase / decrease is systematic, we should be able to use this to improve the shadow elimination model.

5.2.1.2 Looking at the region level

Our current implementation only looks at the pixel level, however we believe that improvements could be made by looking at the region level. In section [3.1.4.1] we described how one could look at a neighborhood of pixels too detects small movements in background objects. We believe that this technique could greatly improve our pixel-based background subtraction program, especially in outdoor scenes.

5.2.1.3 Looking at the frame level

We see several ways one could look at the frame level to improve background subtraction.

First one could extend the neighborhood of pixels method described in section [3.1.4.1] to the whole frame. One could do this by sampling N random pixels from the scene and if x% of those pixels had moved in the same direction it would be an indication that the camera itself had moved and we could use this to adjust the background model.

Second, if x% of the pixels had raised its values it would be an indication that more light globally entered the scene. We could then adjust the background model to the new conditions. Likewise if x% of the pixels had lowered its values. When these big light changes occur we should also be aware of not making many false foreground detections, as big light changes can easily lead to border values which are problematic as described in section [3.2.1.1]

5.2.1.4 Exploit camera settings

The Sony DFW-VL500 camera has a number of settings, such as the shutter and zoom. These settings can be adjusted with the program Coriander[4]. We adjusted the settings before each recording to match the scene’s conditions.

It would be advantageous if our implementation could automatically adjust these settings to match the conditions of the scene, as it would free the user from doing it.

The pixel-based background subtraction program could also adjust these settings continuously in response to scene changes. For example if a cloud covers the sky it could adjust the shutter to accept more light. This would minimize the number of pixel intensities which lay on the border (0 and 255).
Another way to use the camera settings would be to zoom in on objects of interest and thereby get a better view of these objects. However this would only be of limited use with the current camera, since this only supports zooming without human intervention, it can not rotate or otherwise adjust its viewing angle.

5.2.2 Improvements of the model learning part of our system

We did accomplish to make a model from the extracted exemplars. However, it could be improved in a number of ways. In the following we will look at ideas of how this could be done.

5.2.2.1 Combination of distance measures

We found that the Chamfer distance was good at discriminating different pedestrian stances. However, for other objects this may not be true.

Therefore, it would be wise to use different distance measures for different objects. This could be done by testing each distance measure to see which has the best discriminating effect. This could, for example, be done by calculating the crispness of clustering results as it is done in section 4.4.2.6.

Another solution, than using the best distance measure, would be to compute all of the measures and calculate the weighted mean of their results according to their discriminating effect.

5.2.2.2 Different distance measures at different levels in the hierarchy

It is likely that some people would want a model with multiple objects. For example, pedestrians, cars, bicycles, and motorcycles. As the number of objects grow it will become more and more expensive to traverse the resulting hierarchy (by the recognition part). However, the top levels would probably be easy to discriminate as cars and pedestrians are very different.

Therefore, it could be an advantage to use a distance measure which is fast, but with small discriminatory ability at the levels, and only apply the slow and very discriminatory distance measures when needed (at the lower levels). For example, the moment based distance is faster than the Chamfer distance, but not as good at discriminating pedestrians.

5.2.2.3 Textures

We only look at the silhouette / contour of objects and therefore exclude all texture information. We did this as different pedestrians can have different cloths on and as illumination change can disturb the texture. However, for some objects the easiest way to discriminate them seem to be from their texture. For example, a zebra and a horse can be difficult to discriminate by using their silhouettes, but it should be quite easy to discriminate them by using their textures.
Therefore, a texture based distance measure should probably be considered. This could be used in combination with other distance measures, as described in section 5.2.2.1.

### 5.2.2.4 Hierarchy - variable number of children

There is a tradeoff in the hierarchy between small and large width (number of children). A small width is good, as when the recognition part is traversing the hierarchy, it has a small number of children to examine for each node. On the other hand, a small width means that the chance of ruling out entire branches is also small. For a large width the opposite is true.

Currently, we use the same width (number of children) throughout the hierarchy. A better solution would be to use a different number of children for each node, depending on its exemplars, as different sets of exemplars would have different optimal widths. A way to judge the optimal width could be by clustering using different widths, and choosing the one with highest dispersion \( \text{scatter} \), see section 4.4.1.4.

### 5.2.2.5 Test of the system with other object types

We used pedestrians to test our implementation. However, we do believe that our implementation will work with other types of objects, such as dog, horse, or car. We believe this as we have used no special techniques, which only apply to pedestrians.

However, we do not know that the implementation will work with other types of objects, as we have not tested other object types. This kind of testing should be done to validate the generality of the implementation.

### 5.2.2.6 Real-time computation / adaption of model

Currently, the model is computed off-line. This has the advantage that we can spend a lot of CPU cycles computing the model, and thereby making it better. However, it also has some disadvantages. Firstly, a model computed off-line can only encompass a limited number of object types. If, the recognition part, encounters a new object type, it will not be recognized. Secondly, a model computed off-line will probably be bad at adapting to a given object when encountered by the recognition part. For example, a pedestrians cloth will probably not change during a short video sequence, and the texture information could be used to adapt the model to this instance of this particular pedestrian.

These problems could be solved by computing / adapting while performing the recognition part. However, this would require a very fast construction / adaption of the model. This means that we need a fast distance measure and a fast clustering algorithm. Moment based distance is reasonably fast and alternative K-means is also reasonably fast, and they probably could with further optimizations be used to construct models on-line.
5.3 Observations

Our general observation of pixel-based background subtraction is that it performs reasonably well if the background is non-stationary to some degree. However, if there are a lot of motion in the background the pixel-based background subtraction fails to segment the interesting foreground from the background. This means that pixel-based background subtraction is especially useful if it is used under controlled environments. However, the pixel-based background subtraction program, developed in the context of this thesis, which is based on the non-parametric model, performs significantly well for our purpose, since we just use it as a preprocessing step in order to minimize the amount of data the modeling part of the system needs to process later on, and as it is used in an easy scenario.

We observed that our dynamic expanding filter is very useful for removing noise without changing the foreground objects.

Our observation of building an object model, in the context of a model-based background subtraction system, is that, at least from visual inspections, it seems possible to automatically construct a model from exemplars, which can substantially aid recognition and tracking. However, a large amount of input is needed for it to cover the variations among pedestrians and their clothing. The quantifiable tests of clustering and distance measures, makes us more confident that the model is sound.

Our observations, at least when modeling pedestrians, is that the Chamfer distance is better at comparing exemplars than the both Hausdorff and the moment-based distance.

Our observations, at least when modeling pedestrians, suggest that alternative K-means is a little better than ordinary K-means at clustering exemplars. It is a lot better than simulated annealing.

We observed that normalization, made in the context of this thesis, can be used to ease the comparison of exemplars.

Finally, we can conclude that we succeeded in what we set out to do, namely to implement the first two parts of a model-based background subtraction system.
Bibliography

   http://www.oonumerics.org/blitz/.


   http://gscanbus.berlios.de/.

[9] Intel c++ Compiler 7.1 for Linux.

    http://sourceforge.net/projects/libdc1394/.

    http://sourceforge.net/projects/libraw1394/.

[12] Linux1394 homepage.
    http://www.linux1394.org/.

http://www.markschulze.net/java/meanmed.html.


[16] Octave main webpage.  

[17] Scilab main webpage.  


http://bssc.sel.sony.com/Professional/webapp/ModelInfo?id=17519.


[28] Dmitry Chetverikov. Using Distance Transform to Minimize HD.  


http://www.gavrila.net/eccv00.pdf.


Appendix A

Software

- **Operating system:** The PC is available with a GNU/Linux operating system pre-installed (SUSE 8.1 distribution). This widely determines which other software is available for our implementation. In order to get the Linux kernel to support IEEE1394 devices, the guidelines from [12] has been followed. Basically this simply involves loading specific IEEE1394 kernel modules.

- **High-level programming environments:** Several systems are available, that allows for fast development and testing of a model-based background subtraction implementation. Matlab [13] is a commercial product developed by MathWorks, which is widely used for scientific computing. The program is available with very useful statistics- and image-processing toolboxes. These contain implementations of various statistical functions and image filters/operators. Also the available cameras can quite easily be made to work inside Matlab. One major disadvantage of using Matlab is that it is not free. This is however not a problem, since a license is available for our use. Amongst the free alternatives which has been examined, Scilab [17] and Octave [16] seems to be the most promising. Unfortunately these systems currently only offer a subset of the computer vision functionality, which Matlab offers.

- **Programming languages:** The major drawback of using a high-level programming environment, is the performance overhead caused by it being interpreted. As we are going to work with image sequences, there will be a substantial amount of computations where Matlab may fail to be good enough for a real time background subtraction demo. Therefore C++ will be used if it is deemed necessary. We will use the GNU GCC compiler[7]. We may also try a special C++ compiler [9] that has been developed by Intel, which apparently generates code which is very efficient when being executed on Pentium class processors.

- **Utilities:** A few utilities are useful when working with an IEEE1394 based camera. Coriander [4] is useful for configuring the various settings of the camera. For our purpose this is primarily the shutter, gain, focus and zoom of the camera. Without this utility, additional code would have to be developed for controlling the camera. Another tool named gscanbus [8], is used to control devices on the IEEE1394 bus.
- **Programming libraries**: Several programming libraries will be needed during the model-based background subtraction implementation. The functionality of these libraries range from camera control to array manipulation to low-level multimedia library. The libraries will be presented during the description of the background subtraction implementation.
Appendix B

Examples of the three noise filters

We will in this section give an example for each of the noise filtering techniques we described in this thesis.

B.1 Mean filtering example

Mean filtering example of a single 3-by-3 window of values is shown in figure B.1.

Unfiltered values: 2+5+9+3+6+1+7+2+4=39, the mean: \( \frac{39}{9} = 4.33 \).

Mean filtered

Figure B.1: 3-by-3 Mean filtering example.

Mean filtering simply replaces each pixel value in an image with the mean value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. In the figure the center value, which was 6 is replaced by 4.33, which is the mean of all nine values.
B.2 Median filtering example

Median filtering example of a single 3-by-3 window of values is shown in figure [B.2]

<table>
<thead>
<tr>
<th>22</th>
<th>35</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>66</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>48</td>
</tr>
</tbody>
</table>

Unfiltered values: 3, 7, 20, 22, 22, 29, 35, 48, 66 (Sorted values), the median is 22.

<table>
<thead>
<tr>
<th>22</th>
<th>35</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>48</td>
</tr>
</tbody>
</table>

As seen in the figure the center value, which was 66, is replaced by 22, which is the median of all nine values.

Note that for the values of the window in the mean example, the median filter would return a value of 4, since the ordered values are 1, 2, 2, 3, 4, 5, 6, 7, 9. For the values of the window in the median example, the mean filter returns the value 28 since the sum of the nine values in the window is 252 and \( \frac{252}{9} = 28 \).

B.3 Dynamic expanding filter example

A theoretical example of our filter will be presented in this section.

B.3.1 Examples of using our filter

Figure [B.3] shows a small input image and the resulting average image using a N value of two. The width and height of the input image is 9 pixels, which results in a average image with the dimensions \( \frac{9}{2} = 4.5 \), which is reduced to 4. This means that one horizontal and one vertical line in the input image will be ignored by the filter. This is indicated in figure [B.3] with a double line.

The values inside the first 2-by-2 pixel square in the input image (64, 68, 58 and 58) are used to calculate the first mean value. The actual calculation for the first pixel is: \( \frac{64 + 68 + 58 + 58}{4} = \frac{248}{4} = 62 \). Had the division not resulted in an integer the decimals would have been truncated.

After the entire average image has been created it is traversed to find the maximum and minimum neighbors. The first center pixel to find neighbors for is (1,1) with the value 63 in the
average image. Starting the neighbor scan from pixel coordinates (0,0) and first going down the y-axis will compare the following neighbor pixel values: 62, 67, 81, 62, 100, 57, 68 and 78. From these, we find that the maximum value is 100 and the minimum value is 57.

Now these values are used to update the input image. The coordinates for the values that we need to update are (2,2), (2,3), (3,2) and (3,3) as they are the values that were used to find the value 63 in the average image. The values for these coordinates are: 59, 64, 65 and 67. Each of these are compared in turn to discover if any of them are either above the maximum or below the minimum which is not the case here, so the original values are not altered.

The second pixel becoming center in the average image is the one in coordinates (2,1) with the value 68. The neighbors for this pixel are 62, 63, 100, 57, 78, 57, 61 and 77 which yields 57 as the minimum and 100 as maximum. Now these are compared with the values that was used to calculate the mean at (2,1) which are: 66, 64, 71 and 73. Again we observe that all the values are within the maximum and minimum boundaries, so the original values are left unchanged.

Now we move down one line to coordinates (1,2) in the average image to the pixel with a value of 100. We do this because the next pixel in the current row do not have eight neighbor pixels. This places us at a pixel with the following neighbor values: 67, 81, 82, 63, 88, 68, 78, 68 and provides us with 63 as minimum and 88 as maximum values.

Comparing the four values: 200, 62, 78 and 63 in the input image with the maximum and minimum shows that the value 200 is higher than the maximum from the average image and it is therefore reduced to 88. Likewise the value 62 is found to be smaller than the minimum and is therefore raised to 63.

The final comparison in this iteration (N=2) is at coordinates (2,2) in the average image. Here the four values in the input image that corresponds to the one at (2,2) is all within the maximum and minimum boundaries, so no changes are made.

After the first iteration, N is increased to 3 and the whole procedure starts over with the modified input image. This time there will be no truncated pixel at the edge, as \( \frac{9}{3} = 3 \) divides to a integer. Once N reaches the end of the range specified for the filter, the filter stops processing the image.
Appendix C

Test results of our pixel-based background subtraction algorithm

In this section we present some results of our pixel-based background subtraction implementation. During the software development phase, after we got the cameras working within our source code, most testing was done using a live video feed from the camera. However, during the final test we have chosen to use recorded test movies, because this enables us to conduct a better comparisons and include the results on the accompanying CDROM one. Where to find the test movies can be seen in appendix G.

The output results of running the algorithm offline as opposed to online is the same.

C.1 Preliminaries

Before actually testing our pixel-based background subtraction implementation, some initial thoughts needs to be made as to which scenarios we should use.

Initially we will use a computer generated test movie, in which we can control more or less everything about lighting. The camera can also be seen as perfect, since we are not limited by the quality of some lens optics.

Secondly we have some scenes recorded indoor, which we were able to control more tightly than an outdoor scene would allow. Thereby we have a somewhat robust environment, in which we are able to choose a practically stable non-moving background. Also we have different lighting options. The lighting options gives us the ability to experiment with how good the algorithm performs with shadows and changing light intensity levels. As we now use a camera to record our footage, we can expect the recordings to be somewhat noisy. It also appears as if the Sony DFW-VL500 IEEE1394 camera [19] is able to record scenes with less noise if they are well lit. This is usually the case when a scene is lit by ordinary daylight as opposed to the neon lights inside the lab. Adding more light to a scene with help of spotlights also seems to reduce the noise, but introduces heavy shadows as well. We are confident that these noise
problems can be solved using lighting, but we do not feel the need to experiment too much with light settings.

Thirdly we have also recorded a few outdoor scenes, in order to see how our background subtraction performs when it is taken outside the lab. The outdoor scenes are most likely to have non-stationary backgrounds.

The generated movies have a framerate of 25 frames per second. The program we developed (see appendix D.1) for recording movies however, is not able to capture movies at this framerate. The actual framerate varies some, but is expected to be between 10 and 20 frames per second on the computer provided for this project (see section 1.4 for computer specs). The resolution of our captured movies is 320x240 pixels.

Furthermore, we have recorded movies which demonstrate and test the common pixel-based background subtraction issues mentioned in section 3.1.1.2.

**C.2 Description of our movies**

Our test movies have been recorded mainly with the typical background subtraction issues in mind. This section will briefly describe the scenery of our movies, that is what is happening in the scenes and which prior expectations we have to results. We will also mention which of the issues: Moving shadows, slow-moving objects, non-stationary background, camouflage, time of day, light switch and moving camera (elaborated in section 3.1.1.2) are relevant for the given movie.

**C.2.1 Computer generated images**

We have generated two movies using a 3D software package. The first movie called 'plane' is very simple and short (50 frames/2 seconds), and was used as a starting point for our project. It shows a white plane flying through the scene with a very simple house and a large chess brick. We do not see any point in trying out our final algorithm with this movie, as we already had good results with this scene during our Matlab experiments. The movie is included as 'plane' on CDROM one for completeness.

The second generated movie named 'cars', is a bit more complex and also of a longer duration. It shows a few cars driving on a road. Each car casts a shadow, and during the end the illuminations changes a bit. The entire animation is 900 frames long, and should be able to provide good results, though the shadows may be to dark to distinguish from foreground objects.

**C.2.2 Indoor footage**

We have quite a few movies recorded inside the software lab at AUE.
It soon came to our attention that the software lab at AUE is not the best suited location. For example a large single colored wall is not available as background. Also lights for creating different levels of shadows is not available. Despite of this, we did record quite a few movies, proving the previously mentioned common issues of pixel-based background subtraction.

The 'sudden light’ movie consists of 120 frames which illustrate the light switch issue. In the movie, an overhead projector is turned on and off, resulting in an intense illumination change. We expect this intense light change to be too extreme for our implementation to disregard it. Therefore we will probably see the area affected by the light being detected as foreground.

The 'camouage’ movie demonstrate the camouflage issue, in which an object has the same color as the background, and is therefore falsely detected as belonging to the background. In the movie, a coffeeepot is shown as a cup with the same color is placed in front of it. We believe that this movie demonstrate the issue rather well, and that finding a pixel-based background subtraction algorithm solving the issue will be difficult.

The 'camera bump’ movie shows the whiteboard as the camera is suddenly moved. This could happen if someone bumped into the camera, or if an outdoor camera was moved suddenly by a heavy wind blow. This 120 frames movie helps demonstrate the moving camera issue.

The issue with slowly moving objects is illustrated in our movie 'slowly moving object’, in which a computer mouse is dragged slowly past the camera. Here we expect a bad detection of the mouse, since the background model will probably adjust to the foreground object.

The next two movies, 'find bottle minus light’ and 'find bottle plus light’, shows Mohamad walking into the software lab and finding his long lost bottle. The minus light movie is just the lab without any additional lighting, and without really visible shadows. In the plus light movie we have used an overhead projector for illumination of the scene. This produced a heavy shadow on the wall. As the bottle is removed from the scene, we expect the area occupied by it to appear as foreground. Having the same scene recorded with different lighting also helps in testing the moving shadow issue.

The last two movies, 'whiteboard minus light’ and 'whiteboard plus light’, shows Mads sketching a nice drawing on the whiteboard. The first movie is with no additional lighting, and the second has extra light added to the scene using an overhead projector. This will, like the former two movies, visualize the moving shadow issue. And as Mads draws on the whiteboard, new foreground material is added to the scene. Therefore we expect that parts of the drawing will show up as foreground.

C.2.3 Outdoor footage

Due to the fact that our camera needs a computer close by (or requires a very, very long cable) to store images, our possibilities of recording the outdoor movies were restricted to the scenery outside the software lab. The movies were recorded through an open window to reduce glare from the windows. Despite this restriction, the scenery outside the lab is not bad, since it contains a small road, a grass area and a small pathway partly hidden behind large trees.
C.3 Our results and comparison

The first outdoor movie, ‘road and grass walk’, shows Thomas walking on the road and grass, while throwing a small object in the air. The movie is 220 frames long, and contains no visible illumination changes. Likewise there are hardly any shadows showing. The grass is slowly moving in the wind, which demonstrates a mild version of the non-stationary background issue. The bottle is rather small, which is why we expect it to be difficult to detect. However, Thomas is expected to show up quite nicely as foreground.

The 'birds’ movie captures the same scenery as the previous movie, but we were lucky enough to capture two birds moving around. The illumination also change during the movie, as a cloud covers the sun towards the end of the movie. The birds are rather small in the images, and we expect that they will be difficult to detect as foreground objects. The issue of time of day is relevant in this scene, as the illumination changes slightly.

The movie ‘meeting’ shows Mohamad and Thomas walking on the road, greeting as they pass each other. Here the persons cast shadows, and some changes in the background occurs. The bush moves substantially due to wind during the second half of the movie. In the first half it was quite stationary however. Therefore, since we use the first 50 frames for learning the background, we expect that movements caused by wind will show up as foreground in the last part of the result movie.

The movie ‘grass walk’ shows Mads and Thomas walking in the grass in front of trees moving due to moderate wind. This is another example with a non-stationary background, and the illumination also change as the scene gets brighter towards the end. Here we expect the persons to be nicely categorized as foreground, even though there may be visible noise from the trees.

The movie 'path walk far’ pretty much captures the same scenery as the movie above. But in this movie Mads and Thomas are walking on the path, which is partly hidden by trees. We regard this movie as fairly difficult for pixel-based background subtraction, as the moving foreground objects are occluded by the moving trees. In the end, as the persons exits the occluded path, we should expect the algorithm to locate them nicely.

The movie 'path walk near’ shows Mohamad and Jakob walking on the path behind the trees. This time the camera is more zoomed in, so it should be easier to discover the foreground objects. However, we still expect that this is not an easy movie for our pixel-based background subtraction algorithm.

Once again using the same scene as with 'path walk far’, we have recorded the movie ‘wind and illumination change’, which only shows the trees moving moderately in the wind, while the illumination of the scene is changing. Here we expect nothing to become foreground, despite the moving branches and changes in illumination.

C.3 Our results and comparison

Here we will present the results from running our non parametric background subtraction (npbgs) implementation, previously mentioned in appendix D.1 on the movies described above. All the presented results have been made using default parameter settings to our program.
It is not easy to judge the performance of the algorithm by just inspecting the results. We could choose to judge the results based on what we expect from the algorithm, or what we believe humans could separate as background and foreground. However, we believe comparing our results with another algorithm is a more objective way to judge our results.

Early in the project, our supervisor provided us with source code developed by UMD. This pixel-based background subtraction implementation performs well, and is therefore good for comparison. Our findings and comparisons to the pixel-based background subtraction approach of UMD is presented below.

### C.3.1 Cars

We expected good results from the computer generated 'cars' movie, as this practically contains no moving background objects and no noise from the recordings. Our pixel-based background subtraction implementation detected the cars as foreground very nicely, and only missed a few holes in some of the car windows. We were also able to detect the actual cars without their shadows. This is a good thing due to the shadow issue. Being able to exclude the shadows from the foreground shows the advantage of using chromatic colors as described in section 3.2.1. We believe that the use of chromatic colors is also why the background subtraction was unaffected by the slight illumination changes towards the end of the movie.

Compared to the UMD-BGS implementation, we believe our approach does a better job. This is because the foreground objects (the cars) contained more holes, and some of the shadows were detected as foreground objects. To show this we have provided some screenshots.

Figure C.1: Screenshots from the 'cars' movie. At the top are 8 images from the original movie. In the middle are the result from our background subtraction algorithm. In the bottom are the results from UMD-BGS.

### C.3.2 Sudden light

Once the overhead projector is turned on, we see a large area being detected as foreground like we expected. As soon as the overhead projector is turned off, the foreground object disappears. One unexpected thing happens with this movie, as the dark computer monitors shows up as
constant foreground objects. During our other tests, we have discovered that our algorithm seems to have general problems with dark areas. A description and a possible explanation of why we have this problem is explained in section 3.2.4.1.

The UMD-BGS implementation seems to have the same problem as us, when it comes to large foreground objects being detected as soon as the overhead projector is turned on. However, this foreground object is removed again before the overhead projector is turned off. We guess that this is due to an update of the background model while the light is on. Though, we do not see the same sudden foreground object appear when the light is finally turned off. This one could have expected in case the background model was really changed.

### C.3.3 Camouflage

As expected the coffee cup fails to be detected fully as foreground in the movie, once it is placed in front of the coffee pot of almost the same colors. Only a small area in the bottom left area of the cup, is detected along with a small shadow projected on the table.

The UMD-BGS implementation has the same problem, although more of the cup is detected in the beginning. This only lasts for a short while, after which only the handle is detected, and later the cup completely disappears.

We expect the camouflage issue to be rather hard to overcome. Even using more advanced model-based background subtraction approaches, we believe some scenarios are simply impossible to background subtract probably.

### C.3.4 Camera bump

Here we expected foreground detections once the camera was bumped into. This was also the case, however our movie contained a large area with the same white background color. Therefore, the false detection is only seen in parts of the scene, where objects of other colors are visible. Had the scene contained more objects, and thereby been more colorful, we would expect the camera bump to generate more severe false detection.

The UMD-BGS algorithm reveals the same problem, but here the falsely detected foreground objects gets smaller during the end of the movie. We expect that this is because of the faster update of the background model.

### C.3.5 Slowly moving object

When the mouse enters the scene it is detected, but soon a hole appears which occupy approximately half of the object. Thereafter it disappears completely as foreground, until the very end where the movement gets faster and the object is detected again.
With the UMD-BGS implementation the same problem reveals itself, just with larger holes in the foreground object. Despite the larger holes, and the very sparse detection, the algorithm does a better job than ours. In situations where ours totally fail at locating the object, it is still able to draw half of an edge of the mouse. To show this we have provided some screenshots in figure [C.2]

![Figure C.2: Screenshots from the 'slowly moving object' movie. At the top are five images from the original movie. In the middle are the result from our background subtraction algorithm. In the bottom are the results from UMD-BGS.](image)

**C.3.6 Find bottle minus light**

The person in the movie is detected quite nicely, but there are many false foreground detections which appear as noise in the result. One of the constant false detection areas is one of the computer monitors, which we expect to show up due to our problem with dark areas as described in section [3.2.4.1](#). The sign on the door and pin-board are also contributing with false detections as well as the shadow on the door as the persons enters the room.

The UMD-BGS implementation does not have nearly as many false detections, but the shadow on the door gives the same problem in this algorithm. The detection of the person has occasional holes, as opposed to ours which detects the person as a completely closed object. Therefore a clear winner can not be found with this movie. Our pixel-based background subtraction has a nice full object of the detected person, but it also has a lot of unwanted detections.

**C.3.7 Find bottle plus light**

Again the person is detected nicely, and the false foreground detections that was a problem with the 'find bottle minus light' movie is reduced. However, the wall shadow made by the strong extra light, added by the overhead projector, also shows up as foreground.
The UMD-BGS implementation has a bit less noise than us, but the problem of also detecting the shadow is the same. There are still some holes in the person occasionally, but to a lesser degree than with the movie without extra light.

For both the ‘find bottle’ movies, we expected that the area occupied by the bottle would show up as foreground once it had been removed by Mohamad. This did indeed happen with our pixel-based background subtraction, and it remains as foreground until the end of the movies. The UMD-BGS algorithm on the other hand, only has the removed bottle showing for a short while.

### C.3.8 Whiteboard minus light

The person is detected nicely, and the shadow is only vaguely detected sometimes. The problem of false detection is heavily reduced in this movie, as opposed to the ‘find bottle’ movies. This is probably due to the fact, that there are much less objects in this scene potentially causing variations or reflections from the room.

The UMD-BGS implementation is also able to spot the person rather well, only it has many more holes in the foreground object compared to us.

### C.3.9 Whiteboard plus light

Here the shadow is more intense due to the bright light added artificially to the scene. This produces a shadow which is too dark for removal by our pixel-based background subtraction approach. Both ‘whiteboard’ movies have a problem with a dark area in the lower right corner. A problem mentioned earlier, belonging only to our pixel-based background subtraction approach. The UMD-BGS implementation still has rather large holes in the foreground object of the person being detected. Both ours and this implementation fail to detect any of the drawings made on the whiteboard, which we expected to leave a visible trace. It seems as the drawing is too vague to be detected at all.

### C.3.10 Road and grass walk

The person walking is clearly detected, although the foreground shape appears a bit rough around the edges. The bottle being thrown into the air while walking is only detected the first time, after which it is eaten up by our filter. This does not come as a surprise, and was what we expected. Adjusting the filter size does not seem to remedy this problem, since viewing the unfiltered results from our algorithm, show that the noise lumps are as big as the bottle. So by reducing the filter size to capture the bottle, we would also introduce random false detections.

The UMD-BGS algorithm, on the other hand, is able to spot the bottle being thrown into the air quite nicely through the entire movie. It also has a less rough edge around the person. To show this we have provided some screenshots.
Figure C.3: Screenshots from the ‘road and grass walk’ movie. At the top are five images from the original movie. In the middle are the result from our pixel-based background subtraction algorithm. In the bottom are the results from UMD-BGS.

C.3.11 Birds

As we expected the birds are too small, and therefore our filter eats them up in the final result. However, to our surprise and enjoyment, adjusting our filter settings makes it possible to detect the birds quite nicely.

The UMD-BGS implementation also spots the birds very nicely by default, or at least the bird in the front of the scene. The second one shows up partially during the movie.

C.3.12 Meeting

Again we witness our problem with dark areas, as the shadow from the bush shows up as foreground. The persons are detected nicely as we would expect, but unfortunately so are their shadows. The movements of the bushes towards the end, which we expected to show up as foreground, did indeed show up.

The UMD-BGS implementation gets pretty much the same result as us, although without the dark areas where the shadow is. Also there is less foreground detection by the moving bush.

C.3.13 Grass walk

The persons are detected rather nicely despite the moderately moving trees. However, at the end when the persons have exited the scene, a heavy wind moves the trees in the right hand side of the scene. This movement results in a foreground detection of a large parts of the branches.
C.3 Our results and comparison

The UMD-BGS approach is also able to detect the persons nicely, but some small noise spots appear in the middle of the scene during most of the movie. In the end the large false foreground detection is much smaller than ours.

Path walk far: In this movie we experience periods with false foreground detection due to moving trees. There is no foreground detection of the people walking on the path behind the trees, but once they clear the trees they show up nicely as expected.

Once again the UMD-BGS implementation has random noise from the moving trees, but still not as heavily as us. Although not much, it detects the persons a bit before we do on the path behind the trees.

C.3.14 Path walk near

In this movie we have constant false foreground detection in the right side, and also a little in the middle of the scene. This happens even though the affected areas does not seem to be moving that much in the movie. We expected that this might had to do with the frames used for training the background model, as the first 50 frames shows moderate movement in this area. Increasing the number of train frames, and experimenting with the update frequency, did however not result in any mentionable improvements. As the areas that show up as false foreground are the darkest in the scene, we now believe this to be yet another victim of our problem with dark areas.

The detection of the persons only happen when they are located in the left half of the scene. We expect that we could have detected them earlier towards the middle of the scene as the path is quite visible there. Unfortunately this area is mostly covered with false foreground detection, which makes it impossible to see if we actually detected the people there. The times where the persons do show up as foreground, the detected objects are a bit fuzzy in the edges.

The UMD-BGS implementation obviously does not have the same problem with dark areas, and therefore the false detection is not that severe. Only small random spots show up from the moving trees. The detected foreground of the persons walking the path has more holes in them than our results, but overall it does perform better with this movie.

C.3.15 Wind and illumination change

Here we hoped not to see any foreground detection at all, but we do occasionally get foreground detections in this movie. This is especially true during the heavy wind blows. We did expect the amount of false detection to be less than what the result reveals however. The illumination change however does not affect the results, which is a good thing.

Again the approach made by UMD-BGS performs better, as it only shows small, almost none, random foreground detection spots.
This concludes the presentation of our test, in which several recorded movies have been used to compare our pixel-based background subtraction implementation with the UMD-BGS approach.
Appendix D

Technical implementation description of pixel-based background subtraction

In this appendix we will describe the technical aspects of the pixel-based background subtraction implementations, and explain how to find the executables on the CDROMs attached to the thesis.

During our project we have experimented with several different algorithms. These have both been implemented in Matlab and C++. Matlab provided ease of implementation, whereas C++ provided execution efficiency. In this section we will describe where to find source code, executables, and documentation for the source code. Furthermore, the usage of the executables will also be described.

D.1 Executables and prerequisites

The executables can be found on CDROM one in the directory `cplusplusBGS/executables`. They are all, to a certain extent, self describing. In the next section further instructions on how to use these programs will be given. Below is a table with the various programs that can be found on CDROM one and a short description of them:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npbgs</td>
<td>Non-parametric background subtraction, based on Ahmed ElGammal’s algorithm</td>
</tr>
<tr>
<td>ShowV4L</td>
<td>Show input from a Video4Linux webcam</td>
</tr>
<tr>
<td>recordMovie</td>
<td>Records a movie from a Video4Linux webcam or from an IEEE1394 camera</td>
</tr>
<tr>
<td>basicBGS</td>
<td>Basic background subtraction using single Gaussian (mean of color channels).</td>
</tr>
</tbody>
</table>

Table D.1: BGS programs.

It should be possible to execute these files on an Intel 386 compatible computer running Linux. However one will need to have certain libraries installed:
- Simple DirectMedia Layer (SDL)[18]
- Blitz++[1]
- Libraw1394[11]
- Libdc1394[10]

We have not tested every flavor of Linux, though it runs on RedHat 8.0, SUSE 8.1 and Slackware 9.0 distributions.

A IEEE1394 or Video4Linux supported camera is also needed, if a live demo is to be run.

Some Matlab based software is also located on CDROM one in the directory ’matlab/sh’ and descendants, though we will not describe that in detail.

D.1.1 Compilation of C++ programs

It may be an advantage to compile the programs from scratch, as it will give the compiler an opportunity to optimize for the reader’s specific system. However, not all systems will be recognized by the makefile (see CDROM one directory ’cplusplusBGS/Makefile’).

Make sure you have installed the right software, see above. Copy the directory ’cplusplusBGS’ from CDROM one onto the hard disk. Let the current directory be in ’<path to cplusplusBGS>/cplusplusBGS’ and type ’make’. The programs should now compile.

D.2 Usage of background subtraction programs

The usage of the executable C++ programs should all be fairly self describing. Notice however, that the programs cannot be used to adjust the settings of a IEEE1394 based camera. Therefore we recommend the use of the utility program Coriander [4] in combination with our programs. For ease of use, we will give a few examples of using the programs.

D.2.1 npbgs

`executables/npbgs -camera movies/renderedRoadScene/test -number-of-train-frames 20`

Will create background subtraction on the movie sequence stored in ’movies/renderedRoadScene’. The images begin with the name ’test’. Twenty training frames will be used.

`executables/npbgs -camera ieee320x240 -threshold 0.08`

Will perform live background subtraction on frames captured from a Sony digital IEEE1394 camera. The threshold will be set to 0.08.
D.2.2 ShowV4L

executables/ShowV4L
Will show a movie captured form a Video4Linux webcam. The device '/dev/video0' will be used.

executables/ShowV4L movies/renderedRoadScene/test
Will show the movie stored in directory 'movies/renderedRoadScene', where the files has the base name 'test'.

D.2.3 recordMovie

executables/recordMovie ieee320x240 50
Will show 50 frames from the IEEE320x240 camera, though it will not store them.

executables/recordMovie ieee320x240 50 /bob/test
Will show 50 frames from the IEEE320x240 camera and store them in '/bob' with base name 'test'.

D.2.4 basicBGS

executables/basicBGS –help
Will print all available usage information

executables/basicBGS -m color -d 2 -f 0.05
Will perform a basic color based background subtraction, by combining the R, G and B color channels (calculating their mean). A pixel must be more than 2 standard deviations away from mean to become foreground. Exponential forgetting is used with a forgetting rate of 0.05.

D.3 Source code and documentation

The source code and documentation for the C++ part can be found on CDROM one in the directory 'cplusplusBGS' and its descendents. The directory has the structure shown in table D.2.
### Table D.2: BGS source code and implementation.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>automaticDocumentation</td>
<td>Contains auto-generated documentation.</td>
</tr>
<tr>
<td>bgs</td>
<td>Contains some general background subtraction code.</td>
</tr>
<tr>
<td>basicBGS</td>
<td>Contains code for basic background subtraction algorithms.</td>
</tr>
<tr>
<td>npbgs</td>
<td>Contains the non-parametric background subtraction algorithm.</td>
</tr>
<tr>
<td>devices</td>
<td>Contains code for IEEE1394 and Video4Linux camera. It also contains a pseudo camera which reads from the disc. Furthermore it contains the source for the executables 'ShowV4L' and 'recordMovie'.</td>
</tr>
<tr>
<td>executables</td>
<td>Contains the executables.</td>
</tr>
<tr>
<td>filter</td>
<td>Contains filter functions.</td>
</tr>
<tr>
<td>movies</td>
<td>Contains test movies.</td>
</tr>
<tr>
<td>util</td>
<td>Contains small utility functions.</td>
</tr>
</tbody>
</table>
Appendix E

Technical implementation description of model learning software

In this appendix we will describe the technical aspects of the model learning implementation, and explain how to use the software found on CDROM one attached to this thesis.

E.1 Prerequisites

We have not tested every flavor of Linux, though it runs on RedHat 8.0, SUSE 8.1 and Slackware 9.0 distributions.

We have used Matlab version 6.1 with the Matlab graphics package installed. It may work with other versions of Matlab.

E.1.1 Compilation of C++ Matlab function

A single Matlab function has been implemented in C++ for efficiency. This function can be compiled by copying CDROM one directory ’matlab/sh/objectDetection’ to the hard disk. Then change directory to ’matlab/sh/objectDetection/exemplarExtraction’ and type ’make clean’ and ’make’. This function is already compiled on CDROM one, but recompiling it may be necessary.

E.2 Introduction to usage of the Matlab function

We will not describe every Matlab function implemented. In stead we will here make a short walk through where we construct a pedestrian model from a background subtracted movie. The easiest way to get such a movie is to use one of those which is on CDROM one, see directory ’movieBackgroundSubtracted/bgs050104’.
Below the walk is shown. Notice that Matlab functions are shown as italic.

- Copy the 'matlab' directory onto your local hard disk.
- Change directory to ‘<local hard drive>/matlab/sh’.
- Start Matlab.
- Execute `addpaths` function in Matlab.
- Execute `[ObjectTypes, Rejected] = exemplifyMovie('BaseName', 1000, 1==1, 0, 80);`, where BaseName is the path to the movie + filename except 0001.BMP. For example '<CDROM one>/movieBackgroundSubtracted/bgs050104/050104_take2_'. Remember to enclose BaseName in single quotes.
- Answer the asked questions.
- Execute `Exemplars = normalizeExemplar(ObjectTypes(2).exemplars, 80);`.
- Create ‘model’ directory in ‘<local hard drive>/matlab/sh’ directory.
- Execute `[Hierarchy, Order] = makeAndSaveModel('model', Exemplars, 3, 4);` in matlab.

A model should now be found on the hard drive and in variables in the Matlab programming environment.

E.3 Source code structure and documentation

In this section we will describe the general structure of our Matlab software. In table E.1 we list the packages we have implemented (directories). These packages can be found on CDROM one in directory ‘matlab/sh/objectDetection’.

The easiest way to add all these directories to the Matlab search path, is to execute the function 'addpaths' located on CDROM one in the directory ‘matlab/sh’.

The build-in Matlab 'help' function can be used to list the contents of the different packages and show the documentation for the individual functions.

E.4 Technical description of interface to the recognition part

All data to transfer from the model learning part is stored in the same directory. The files in this directory are described below.
E.4 Technical description of interface to the recognition part

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
<td>Clustering algorithms and helper functions.</td>
</tr>
<tr>
<td>Distances</td>
<td>Distance measure algorithms and helper functions.</td>
</tr>
<tr>
<td>Exemplar extraction</td>
<td>Functions to extract exemplars from a background subtracted movie.</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Hierarchy construction algorithms and helper functions.</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>Miscellaneous helper functions, including functions to save the model to the hard drive.</td>
</tr>
<tr>
<td>Moments</td>
<td>Functions to compute various moments</td>
</tr>
</tbody>
</table>

Table E.1: Matlab packages.

E.4.1 Order.dat

For each level in the hierarchy, the chance of one cluster (node in hierarchy) following another cluster is calculated. Furthermore for each node, the number of data used to calculate these chances are written. The result of these calculations are written to ‘order.dat’. There are N-1 sections in this file (N=levels in hierarchy). The first level is not considered as it only contains one cluster. For each section the following is written (in the shown order):

- Level number: int
- Matrix size: int
- Matrix: array of float of size $<\text{Number of cluster at level}> \times <\text{Number of cluster at level + 1}>$

Level 2 is written first, then level 3, then level 4, ...

If it is impossible to calculate these value for a clusters, then -1 is written to all elements of the cluster. This can be impossible to calculate if no members of the cluster has members which follow, that is all members of the cluster is the last in a series.

If the matrix is index by (row, column) then the row will decide which node one is presently at and column X will be chance of the next exemplar being of cluster X.

The last column is the number of exemplars used to calculate the chances for a cluster (row in matrix).
E.4.2  hierarchy.dat

This file contains the hierarchy. For each node the following is written (in the shown order):
'B' or 'L' if it is a leaf or branch node.
For branches the following is then written:

- Semi-centroid exemplar number: int
- Number of children: int

For leaves the following is written:

- Semi-centroid exemplar number: int
- Number of exemplars in this leaf cluster: int
- Exemplar number: array of int of size <Number of exemplars in this leaf cluster>

The hierarchy are written in pre-order.

E.4.3  Silhouette****.BMP

A BMP image (a silhouette) representing each exemplar is written with the following name:
Silhouette<Exemplar number>.BMP
The exemplar number will use four chars no matter what.

E.4.4  Outline****.BMP

A BMP image (an outline) representing each exemplar is written with the following name:
Outline<Exemplar number>.BMP
The exemplar number will use four chars no matter what.

E.4.5  MeanSilhouette****.BMP

The mean silhouette of each node in the hierarchy. They are ordered by traversing the hierarchy in preorder.
E.4.6 maxDistances.dat

For each node in hierarchy the following is written:

- Max Hausdorff (contour) distance from center exemplar: float
- Max moment based (silhouette) distance from center exemplar: float

Max distance from center exemplar is the distance between the exemplars farthest away from
the center and the center.

The nodes are written in preorder. First all the Hausdorff distances are written, then all the
moment based distances.
Appendix F

Why Chamfer distance is not a metric

We will in the following show that the chamfer and the symmetric chamfer distance do not fulfills the triangle inequality, see section 4.4.2.4 for description of Chamfer distance and see section 4.4.2.2 for description of triangular inequality.

In order to show that it does not fulfill the triangular inequality, we just need to show one case where this is the case. Consider the three images in figure F.1.

Image A consist of a lot of pixels in the top and a single pixel at the bottom. Image B is like image A except it has been mirrored around the middle. Image C has a single pixel at the bottom located at the same spot as Image As bottom pixel.

The distance between image A and B must be small as the majority of pixels at A (those in the top) are located close to image Bs single pixel at the top. Likewise the distance between image B and C is small as the majority of pixels at image B are located close to Cs single pixel. However, the distance between image A and C is large as the majority of pixels at image A are a long way from Cs single pixel. Therefore the triangular inequality is not fulfilled for the directed (non-symmetric) chamfer distance.

If we look at the distance from image C to image B, from image B to image A, and from image C to image A they are also small. Therefore the symmetric Chamfer (maximum of directed Chamfer distance from each direction) distance must be small from image A to image B and
from image B to C. And it must be large from Image A to C. Therefore the triangular inequality is also not fulfilled for the symmetric chamfer distance.
Appendix G

Test movies

All our movies, used in connection with pixel-based background subtraction, can be found in the directory "bgs-test-movies/avis-DivX" located in the root directory of CDROM one. The movies are stored as AVI files.

We have included the following movies and the result of their background subtraction:

- Birds
- Camouflage
- Cars
- Find bottle minus light
- Find bottle plus light
- Grass walk
- Path walk far
- Road and grass walk
- Road and grass walk
- Sudden light
- Wind and illumination change
Index

active contours, 47, 49
alternative K-means, 64
application of background subtraction, 2
architecture of system, see system parts
articulated moving parts, 45
average exemplar, 73

basic probabilistic model, 21
camouflage, 16
centroid, 56
definition, 69
Chamfer distance, 75
chromatic colors, 28
clustering, 56
alternative K-means, 64
agglomerative, 57
divisive, 57
K-means, 63
non-hierarchical, 57, 63
to contour, 52
active, see active contours
definition
exemplar, 52
noise, 32
template, 51
distance
Chamfer distance, 75
Hausdorff distance, 74
moment based distance, 75
distance measures, 67
dynamic expanding filter, 35, 39, 54
efficient towards space, 45
exemplar, 47, 51
definition, 52

extraction, 53
exponential forgetting, 22
extracting exemplars, 53
fast pixel estimation, 30
filtering, 31
dynamic expanding, 35, 39, 54
mean, 33, 38
median, 33, 39, 54
n-connected, 54
test, 37
fuzzy exemplar, 73, 75
distance
Chamfer distance, 75
moment based distance, 75
hardware, 8
Hausdorff distance, 74

hierarchy
traversing, see traversing
implementation, 41, 83, 115, 119
incremental update, 29
K-means, 63
levels, 17
frame, 17
pixel, 17
region, 17
light switch, 16
mean exemplar, 73
mean filter, 33, 38
median filter, 33, 39, 54
metric, 72
mixture model, 23
model-based background subtraction, 3
models
basic probabilistic, 21
mixture, 23
neighborhood, 22
non-parametric, 24 42
moment based distance, 75
moving camera, 16
moving shadows, 16
n-connected filter, 54
neighborhood model, 22
noise
definition, 32
non-binary exemplar, see fuzzy exemplar
non-parametric model, 24 42
non-stationary background, 16 22
normal distribution, 18
multivariate density, 20
univariate density, 18
normalization, 68
in-planar rotations, 68
mirror, 54
position, 70
size, 70
object model
representation, 45
types, 46
2D, explicit, 46
2D, general properties, 46
3D, object shape, 47
orientation, 45
in-planar rotation, 45
view-dependent shape variations, 45
outline, see contour
position, 45
problem definition, 5
asked questions, 7
problems, 16 26
camouflage, 16
light switch, 16
moving camera, 16
moving shadows, 16
non-stationary background, 16 22
slow-moving objects, 16
stationary objects, 16
time of day, 16
related literature, 26
representation
object model, see object model
Requirements for object model, 45
robust towards noise, 45
robust towards partial occlusion, 45
scale, 45
selective update, 30
semi-centroid, 64
silhouette, 52
example, 52
similarity matrix, 56
slow-moving objects, 16
stationary objects, 16
statistical distance, see moment based distance
support efficient recognition, 45
system architecture, see system parts
system parts, 3–5
initial segmentation, 3
model learning, 3
recognition, 3
techniques, 26
chromatic colors, 28
fast pixel estimation, 30
incremental update, 29
selective update, 30
template, 47 51
definition, 51
test
clustering, 66
distance measures, 77
pixel-based background subtraction, 43
104
time of day, 16
traversing, 60
UMD-BGS, 43 108