IN3405 - Bachelor of Science Project

Intelligent Multi-camera Video Surveillance

Final report

Authors:
Pieter Hameete - 4025539
Sebastiaan Leysen - 4001370
Tamis van der Laan - 4057694

Thesis Committee:
Prof. Drs. Dr. L.J.M. Rothkrantz
Ir. Iulia Lefter
Drs. P.R. van Nieuwenhuizen
Dr. Martha Larson

Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science

July, 2012
Summary

Video surveillance is found in different areas, such as public buildings, metro stations and military areas. These systems have become increasingly cost efficient, allowing for larger systems as well. Traditionally multiple security cameras are positioned throughout the area, and linked to monitor screens. These screens are monitored by security personnel. Unfortunately humans suffer from certain flaws when it comes to surveillance. For example, humans may lose focus when nothing happens for a long period of time.

Though computers may no be as good at computer vision and reasoning as human operators, they provide different advantages. For instance, computers are capable of working 24 hours a day, 7 hours a week. By using an automated intelligent multi-camera video surveillance system the security personnel could be supported in their work, allowing for less security flaws in monitoring the area.

For this Bachelor project the project group has developed such an automated intelligent multi-camera video surveillance system. In particular, the system was developed for monitoring the 'Netherlands Defence Academy' (NLDA) area of the 'Koninklijk Instituut voor de Marine' (KIM). In order to develop the system, a scientific approach was used in combination with an incremental, agile development technique called Scrum. Weekly meetings were held with the group, the supervisors and the domain experts.

The system consists of two applications: a Client written in C++ with OpenCV and a Server application written in Java. The Client application is attached to a security camera, and will then detect moving objects, classify them as either human or non-human, and determine their location relative to the camera. For each detected object it will then transmit this information to the Server application. This Server application gathers the information from the Clients and combines the information into actual objects in the monitored area. It then reasons about these objects by using their history of GPS locations to detect whether suspicious situations are occurring. Examples of such situations are when a person enters a restricted area, when a person suddenly starts running or when a person has been following another person or a period of time. In case of a suspicious situation the security personnel is alarmed, and relevant information is displayed to allow the personnel to take action if required.

Ensuring code quality, maintainability and extendability of the system was an important aspect of the project. Using a diverse set of tools we ensured that these properties of the system were maintained throughout the project. It is simple to change image processing modules in the Client, or add new reasoning rules to the Server. This resulted in a much appreciated 5 star rating from the Software Improvement Group.

Another important aspect of the system was testing. Using Google Test for C++ and JUnit for Java the parts of the code with simple in- and output behavior were automatically tested. The more complex code parts were tested manually using recorded video, or simulation files generated with a specially developed application.

A working beta implementation of the complete system has been delivered as a proof of concept and all initially set requirements were fulfilled.
Preface

This report is the final report which concludes the development of a Multi-camera Video Surveillance Support System for Military Areas. The project was executed for the IN3405 - Bachelor Project course at the Delft University of Technology in the Netherlands. The assignment was issued by the 'Koninklijk Instituut voor de Marine' which is located in Den Helder in the Netherlands.

The project began at April 23rd 2012 and finished at July 18th 2012. All research and development took place at the Delft University of Technology, department of Electrical Engineering, Mathematics and Computer Science. This report will provide details on the assignment, requirements, design decisions, implementation, testing, process and results of the project.

We would like to thank the following persons in particular:

- Leon Rothkrantz, for supervising the project and providing support and valuable guidance throughout the process.
- Iulia Lefter, for supervising the project and providing support and valuable guidance throughout the process.
- Peter van Nieuwenhuizen, for coordinating the Bachelor Project course.
- Martha Larson, for coordinating the Bachelor Project course and attending our final presentation.

Sebastiaan Leysen, Tamis van der Laan and Pieter Hameete

## Contents

1 Introduction 9  
1.1 Problem Description and Relevance: Video Surveillance 9  
1.2 Project Assignment 9  
1.3 Outline 10

2 Assignment and Requirements 13  
2.1 Assignment Summary 13  
2.2 Current and Proposed System 13  
2.3 Global Requirements 14  
2.4 Detailed Functional and Quality Requirements 14  
2.5 Graphical User Interface Mockup 16

3 Project Methodology 17  
3.1 Process Strategy 17  
3.2 Planning 19  
3.3 Tools 21

4 System Design and Implementation 25  
4.1 Global System Design 25  
4.2 Client Design and Implementation 25  
4.2.1 The Client Image Processing Pipeline 25  
4.2.2 Client Extendability and Maintainability 26  
4.2.3 Client Image Processing Modules 27  
4.2.4 Localization of Moving Objects 30  
4.3 Server Design and Implementation 32  
4.3.1 Message Handler 32  
4.3.2 Whiteboard Design and Implementation 34  
4.3.3 Reasoner Design and Implementation 36  
4.3.4 Utility Methods 41  
4.3.5 Graphical User Interface Design and Implementation 43

5 Testing 49  
5.1 Automated Testing 49  
5.1.1 Client Unit Testing 49  
5.1.2 Server Unit Testing 50  
5.2 Field Testing 51  
5.2.1 Field Testing the Client 51  
5.2.2 Testing the Server 52  
5.2.3 Testing the Client and Server 58  
5.3 Software Improvement Group Feedback 59
5.3.1 Intermediate Feedback .............................................. 59
5.3.2 Improvements .......................................................... 60
5.3.3 Final Feedback .......................................................... 60

6 Conclusion ................................................................. 61
A Appendix A - Project Assignment ..................................... 69
B Appendix B - Orientation Report ..................................... 73
C Appendix C - Literature Study Client Modules ..................... 81
D Appendix D - Localization Algorithm ............................... 89
E Appendix E - Literature Survey 1 : Background Subtraction Techniques 95
F Appendix F - Literature Survey 2 : Classification Techniques 107
G Appendix G - Software Improvement Group Feedback (Dutch) 119
1.1 Problem Description and Relevance: Video Surveillance

Video surveillance is a type of surveillance that is found in different kinds of areas, such as public buildings, metro stations and military areas. It has proven to be an effective tool to monitor large areas with limited resources. Because cameras and surveillance systems are continuously developed, these systems become increasingly cost efficient, allowing for larger systems as well.

Traditionally, multiple static security cameras are positioned throughout the area, and attached to monitor screens. These screens are monitored by security personnel in order to detect suspicious behavior. Depending on the seriousness of the situation, further action may be taken in order to resolve it. In Figure 1.1 a composition of images describing an area that is monitored by security cameras is displayed. In the top left a typical suspicious situation is shown: a person that is climbing a fence. In the center of the top of the image, an example of a pair of typical static security cameras is shown. In the top right and bottom half typical areas of interest are displayed. The first shows an entrance barrier, which is an area that requires specific attention. The second image shows an overview of areas of interest on a map, the red area represents an area with restricted access.

The main problem with the current system lies within the nature of the tasks of the security personnel. They must passively watch multiple monitor screens simultaneously. Humans easily get tired and lose concentration, especially in situations when no situations of notice occur for a long time. At regular times the guard will inspect the area in person, leaving the cameras unattended and being exposed to potential attackers. Furthermore, humans are not capable of noticing every small detail, let alone of focusing on multiple things at the same time.

A second problem is the fact that camera footage of some areas is being recorded without supervision of a human operator, simply because it would be too expensive to constantly monitor that area. The data is stored as evidence in case some event occurs. This prevents incidents from being detected in a timely manner, preventing swift responses to potentially dangerous situations. Moreover, it is time-consuming to find the correct video images, especially when the event occurred many hours before it is detected, and the system consists of many different cameras.

By using computers to support the security personnel in their task to watch the monitor screens, some of these problems may be prevented. Computers, unlike humans, are capable of working continuously, without losing focus. Moreover, computers are well scalable to allow for monitoring a large number of security cameras simultaneously. Additionally, computers can communicate constantly, allowing each computer to have a complete and up-to-date view of the monitored area.

1.2 Project Assignment

For this Bachelor of Science project a prototype application has been developed that is capable of monitoring the area of the Netherlands Defence Academy (NLDA) of the ’Koninklijk Instituut
voor de Marine (KIM). The team used a scientific approach combined with an agile development approach called Scrum. The goal of this project was to develop a system that is able to assist the security personnel in observing the NLDA area and alarming the control room in case of a suspicious situation. The personnel in the control room ultimately decides whether or not to send back-up to the location of the suspicious event. Examples of alarming situations include persons that suddenly start running, persons entering a forbidden area or security guards that are being followed.

In order to achieve the project’s goals, the prototype system utilizes the static security cameras located in the area. Each camera is linked to a client software application that detects moving objects from the camera images. This client computes different features, such as the locations of the detected objects and whether they are human or not. Because suspicious behavior may occur over multiple cameras, these extracted features are then transmitted to a central server. The central server application gathers, combines and maintains the information from the different security cameras and then reasons about the situation. Whenever a suspicious situation is detected, the server alarms the control room personnel by means of a graphical user interface.

1.3 Outline

In this report the assignment will be analysed to identify requirements in Chapter 2. Then, in Chapter 3 the strategy and methodology of the process will be explained by going into more detail on our scientific and agile approach, the project planning and tools used throughout the project.
Next, in Chapter 4 the implementation details and design decisions will be explained, first of the
global system, then converging to the more specific components. This is followed by Chapter 5 in
which our testing approach and the results of the different tests are discussed. Finally, the report is
concluded in Chapter 6 by evaluating the results and providing suggestions for future work. After
the conclusion a Bibliography and several Appendices can be found. These appendices contain
intermediate products and more detailed descriptions that are referred to in the text in order to
improve the readability of the main report.
2.1 Assignment Summary

The original assignment can be found in Appendix A. This Section provides a brief overview of the assignment. The ‘Koninklijk Instituut voor de Marne’ (KIM) has many static security cameras positioned in the ‘Netherlands Defence Academy’ (NLDA) area. These cameras are currently monitored in a central control room by security personnel. The most dangerous hours are the hours when almost nobody is present or walking around in the NLDA area. The main problems are caused by the fact that the camera images are being monitored by human operators:

- Humans easily get tired and lose concentration, especially when nothing happens for a long period of time.
- Security personnel must occasionally inspect the area in person, leaving the cameras unattended.
- Humans can not notice and combine every detail when monitoring a large number of cameras.
- Humans are too expensive to allow for 24/7 monitoring of the cameras.
- When an event is not directly detected it is time-consuming to find the correct video footage for the event in the stored data.

In order to reduce some of these issues, a computer system that aids the security personnel must be implemented. This computer system must be capable of tracking moving objects throughout the monitored area. It must then detect suspicious situations, such as when a person starts running or when a person climbs a security barrier. Additionally, it must be possible to define particular areas of interest, such as an area with restricted access. Whenever a suspicious event occurs, the security personnel must be alarmed to allow them to take further action.

2.2 Current and Proposed System

In Figure 2.1 a global overview is given for the current and proposed situation. The components outside of the yellow box describe the current system. There a multiple cameras that stream their live images to the control room, where the images are being stored and monitored by the security personnel.

In the proposed system each camera will be linked to a piece of client software. Corresponding to the original assignment in Appendix A this camera software will detect moving objects in the camera images and extract features such as the location and humanity of the object. Because suspicious behavior may occur over multiple cameras, this information will then be transmitted to a central server application that runs in the control room. This software will constantly reason about the information, such as object paths and speed, to determine whether a suspicious situation is occurring. When such a suspicious situation occurs the server software will display an alarm to the security personnel, such that the personnel can take action.
2.3 Global Requirements

To provide an overview of what the software must be capable of doing, first four global requirements for the system that must be fulfilled are provided. By first painting a broader picture of the goals of the project, the more specific requirements are put into context.

- The system must be capable of capturing data from multiple cameras, and subsequently detect, identify and locate moving objects.

- The received data must be processed into different objects which may then be tracked by the system. The path history of GPS coordinates needs to be used to extract features for further reasoning.

- Given a certain context or region of interest in combination with extracted features the system must determine whether behavior is considered suspicious. Suspicious behavior must result in an alert.

- The server in the control room should visualize the gathered data and the results of the reasoning process to the security personnel throughout a graphical user interface.

Most importantly it should be noted that the user will not provide input to the system. The information traversal therefore is one-way: from the system to the security personnel.

2.4 Detailed Functional and Quality Requirements

Table 2.1 shows the functional requirements that have been identified from the assignment for the proposed system, Table 2.2 enumerates the quality requirements for the system. For each requirement it is stated to which subsystem it relates, i.e. whether it is a requirement for the image processing client that is linked to the camera, 'C', or for the server which gathers the information and reasons about it, 'S'. Additionally each requirement is assigned a priority according to the
MoSCoW model. The 'M' indicates a must have requirement, 'S' represents the should have requirements, 'C' the could have requirements and 'W' marks the requirements that will not be fulfilled. Finally, an indication on how to verify that the requirement is fulfilled is provided for each requirement. A 'D' stands for Demonstration, meaning it can be shown that the requirement is fulfilled by running the application. A 'T' indicates that the requirement can be tested using a formal automated test. Requirements with an 'I' can be fulfilled through inspection of the source code.

Table 2.1: the functional requirements for the multi-camera security system

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Subsystem</th>
<th>Priority</th>
<th>Qualification</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>The system maintains a list of objects that are currently in the monitored area.</td>
<td>S</td>
<td>M</td>
<td>D, I</td>
</tr>
<tr>
<td>F2</td>
<td>The system maintains a history of locations for each object.</td>
<td>S</td>
<td>M</td>
<td>D, I</td>
</tr>
<tr>
<td>F3</td>
<td>The system tracks multiple objects simultaneously.</td>
<td>C, S</td>
<td>M</td>
<td>I</td>
</tr>
<tr>
<td>F4</td>
<td>The system distinguishes between humans and other objects.</td>
<td>C, S</td>
<td>M</td>
<td>D, I</td>
</tr>
<tr>
<td>F5</td>
<td>The system matches new locations to the corresponding object correctly.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F6</td>
<td>The system is capable of identifying an object that disappears from a camera as the same object when it is detected by a different camera.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F7</td>
<td>The system automatically removes information of objects that were not seen for a definable period of time.</td>
<td>S</td>
<td>M</td>
<td>T, I</td>
</tr>
<tr>
<td>F8</td>
<td>The system displays alerts in a graphical user interface when suspicious situations occur.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F9</td>
<td>It is possible to define areas of interest, such that when an object enters the area an alert is displayed.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F10</td>
<td>An alert is displayed when a person is running.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F11</td>
<td>An alert is displayed when a person is following another person.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F12</td>
<td>Areas of Interest are definable, firing an alert when an object is in the area and a specific rule applies.</td>
<td>S</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>F13</td>
<td>The system is able to calculate the mean and variance of the speed of an object over a certain time-frame.</td>
<td>S</td>
<td>M</td>
<td>T, I</td>
</tr>
<tr>
<td>F14</td>
<td>The system displays a map of the area on the GUI on which all objects, cameras and areas of interest are drawn on their correct GPS locations.</td>
<td>S</td>
<td>S</td>
<td>D</td>
</tr>
<tr>
<td>F15</td>
<td>A history of locations for each object is plotted on the map.</td>
<td>S</td>
<td>S</td>
<td>D</td>
</tr>
<tr>
<td>F16</td>
<td>The system prioritizes alarms at different priorities: low, normal, medium, high, highest. Alerts are colored in a color representing the priority of the alert.</td>
<td>S</td>
<td>C</td>
<td>I</td>
</tr>
</tbody>
</table>
The average speed of the objects currently tracked by the system is displayed over time in a plot in the GUI.

The system displays every object in its own unique color throughout the entire GUI.

The system detects a person that enters and later leaves a building as the same person.

The GUI could have a console to display informative messages on the state of the system.

The system uses other features than the GPS location and humanity of an object in the reasoning process.

The system is able to detect object gestures such as waving or falling to the floor.

The system is able to detect interaction between different objects.

<p>| Table 2.2: the quality requirements for the multi-camera security system |
|------------------------|------------------|------------------|</p>
<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Subsystem</th>
<th>Priority</th>
<th>Qualification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>The system is capable of updating at least 5 times per second</td>
<td>C, S</td>
<td>M</td>
<td>T, D.</td>
</tr>
<tr>
<td>Maintainability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>Different types of areas of interest are definable, each firing an alert in a specific case</td>
<td>S</td>
<td>S</td>
<td>D.</td>
</tr>
<tr>
<td>Q3</td>
<td>An additional application allows easy defining of areas of interest and testing scenarios</td>
<td>S</td>
<td>C</td>
<td>D.</td>
</tr>
<tr>
<td>Extendability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>New areas of interest are simple to implement and add to the system.</td>
<td>S</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>Q5</td>
<td>New reasoning rules are simple to implement and add to the system.</td>
<td>S</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>Communication between the client and reasoning programs is secure (SSL)</td>
<td>S</td>
<td>C</td>
<td>I, T.</td>
</tr>
<tr>
<td>Q7</td>
<td>When a client disconnects it is able to reconnect without errors.</td>
<td>S</td>
<td>S</td>
<td>D</td>
</tr>
<tr>
<td>Q9</td>
<td>The client localization algorithm provides accuracies within 2 meters.</td>
<td>C</td>
<td>S</td>
<td>D</td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>The Server application runs on multiple platforms.</td>
<td>S</td>
<td>S</td>
<td>I</td>
</tr>
</tbody>
</table>
2.5 Graphical User Interface Mockup

As follows from the requirements the user will not be allowed to give any input to the system for any of the must have or should have requirements. Instead there will be one way information traversal from the system to the user through the graphical user interface. Functional requirements F8, F14, F15, F17, F18 and F20 define the requirements on the GUI. An initial mockup according to these requirements is given in Figure 2.2.

Figure 2.2: an initial mockup of the graphical user interface

The GUI Component with label 1 satisfies requirements F14 and F15 by showing a (schematic) map of the area that is being monitored as well as the objects currently detected in the area and their path history. In the graphs with label 2 the speed of the objects could be plotted over time to meet requirement F17. The debug console allows for printing of informative messages as required by F20. Most importantly, the table with label 3 can be used to display information on each of the objects in the area, and potential alarms for these objects. Finally, it becomes visible from the mockup how requirement F17 can be met: by assigning each object its unique color throughout the GUI.

As Q3 was a ‘could have’ requirement, no mockup was made for this application. It has been implemented however. The actual graphical user interfaces for both the server application as this additional application can be found in Chapter 4.
In this chapter the methodology for the process is discussed, as well as the original and actual planning and the tools that were used throughout the process.

3.1 Process Strategy

The entire process from initial research up to developing the system took place at the Delft University of Technology rather than at an actual software development company. Working in a small team of three persons results in a different strategy compared to the strategies used by larger teams. For larger teams communication is difficult, where a small team working together in a single room allows for constant communication and discussion. Such situations are very suitable for agile development processes such as Scrum.

Scientific Incremental Development using Scrum

In the initial plan of action we made the choice for a full agile development approach by using Scrum. Scrum is an iterative approach, consisting of multiple incremental development cycles. At the beginning of each cycle a meeting is held in which is decided which features must be added during that cycle. At the end of the cycle the process is evaluated to improve the process quality and to evaluate the progress. We used the ScrumDo online tool for assistance in executing the Scrum process. This tool will be discussed later in this chapter. The Scrum cycles that were executed corresponded to the different phases of the project. These phases are defined by the different deliverables as stated on the IN3405 Blackboard course page and an overview of these phases is given in Table 3.1.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>Planning (P), Plan of Action (PoA)</td>
</tr>
<tr>
<td>Research and Prototyping</td>
<td>Client Prototype (CP)</td>
</tr>
<tr>
<td>Orientation Report</td>
<td>Orientation Report (OR)</td>
</tr>
<tr>
<td>Beta Implementation</td>
<td>Client and Server Beta (CSB), Code Evaluation 1 (SIG1)</td>
</tr>
<tr>
<td>Final Implementation</td>
<td>Final Application (FA), Code Evaluation 2 (SIG2)</td>
</tr>
<tr>
<td>Final Report</td>
<td>Final Report (FR)</td>
</tr>
<tr>
<td>Final Presentation</td>
<td>Final Presentation (FP)</td>
</tr>
</tbody>
</table>

The incremental nature of Scrum and the incremental cycles become obvious when looking at Figure 3.1. In practice Scrum was not practical for each cycle however. We found that Scrum works best during the actual software development phase, resulting in a Feature Driven development process. The most important features are added first, while more details are added after the important features have been implemented. For the research, orientation and report phases
Scrum was less useful and less used as it became difficult to identify and prioritize features.

**Scientific Approach**

The different aspects of a scientific approach are incorporated into the Scrum phases of the project. During the Planning and Research and Prototyping phases meetings were held with domain experts to investigate the domain. Requirements and a system specification were then made. Next, in the Research and Prototyping and Orientation Report phases, a literature survey was performed and different image processing techniques were prototyped. During the Beta Implementation phase the initial system design was made. This design was further refined while implementing the system using the agile Scrum approach. In the final stages of the Final Implementation phase, the system is thoroughly tested to confirm that the system functions correctly, and that the set requirements are met. Finally, during the Final Report phase, the work on the system is concluded and further work is suggested.

**Project Roles**

The Scrum approach requires different roles to be distributed, while within the project group a natural distribution of roles was made:

1. **Sebastiaan:** Scrum Master, Lead Designer, Secretary.
2. **Tamis:** Head Programmer, Prototyper.
3. **Pieter:** Product Manager, Lead Tester and Code Quality, Chairman.

The Product Manager and Scrum Master are roles specific to the Scrum process. The Product Manager determines which features will be implemented during a Scrum development cycle. The Scrum Master ensures that the entire team correctly follows the Scrum process. The Secretary made notes during the weekly group meetings, and the meetings with the supervisors, while the Chairman chaired the group meetings.

The other roles correspond to the implementation phases. The Lead Designer was responsible for the system design, ensuring that the design and its documentation using UML diagrams was
correct. The Head Programmer was responsible for the programming process and system architecture. Finally, the Lead Tester and Code Quality ensured that tests were correctly performed, and that the quality of the code in terms of aesthetics and structure, was maintained.

**Weekly Meetings**

As stated above, weekly meetings were held to support the Scrum process at the beginning and end of the week. At the beginning of the week the Product Manager determined which tasks were to be performed this week, in discussion with the group. At the end of the week the progress was discussed. If required tasks could be postponed to the next cycle. Additionally, the process was discussed to improve the quality of the process. Another important aspect of these meetings was that important design decisions and implementation problems were discussed with the group.

In addition to the weekly group meetings, weekly meetings with the supervisors were held. In the case of this project the supervisors were also the domain experts. During these meetings the project progress was discussed. The supervisors also provided the group with guidance and advice throughout the project. They ensured that the focus of the group was on the correct aspects of the system, and that the system was developed correctly as required by the assignment.

### 3.2 Planning

Using the Scrum agile development process important features are added initially. The system should be functional at the end of each development cycle, and the goal is to add as many features as possible in the available time. Besides the development of the actual applications there are other deliverables that must be finished within a timely manner, as they are requirements for the consequent phases of the project. To work as efficient as possible an initial planning was made in the first week. This planning can be found in Figure 3.2.

![Figure 3.2: the original planning as presented in the Plan of Action](image)

Although this planning was suitable for the first phases of the project, it quickly required some adjustments. Because of other responsibilities of the team members some holes appeared in the schedule that needed to be accounted for. This resulted in a break week in the middle of the project planning. Additionally the deadline for the project had to be extended slightly to account for the unavailability of the team members.

The most significant problem was the extension of the phase in which client image processing techniques were surveyed and the phase for writing the Orientation Report. This cost the team more time then initially reserved. During the first four weeks the team only had Skype meetings with the supervisors. Due to a misunderstanding the focus was too heavy on the image processing, rather than on the reasoning part of the system. This resulted in a severe loss of time and required swift reallocation of focus and available time. A new planning had to be made. This planning can be found in Figure 3.3 and is the planning that was actually followed. A textual description of this schedule that describes activities and deliverables during the phases from Table 3.1 is given thereafter.
### Planning: Week 1

- Make a planning for the 10 project weeks
- Analyse the project description
- Identify the requirements
- Determine tools to be used

**Deliverables:** Plan of Action, Planning

### Research and Prototyping: Week 2-4

- Search required image processing techniques for the client
- Record live video material
- Prototype image processing techniques on live video

**Deliverables:** Client Prototype

### Orientation Report: Week 2-4

- Write a literature survey on the client image processing techniques
- Analyse the application domain
- Make an initial global design for the system
- Determine tools to be used

**Deliverables:** Orientation Report

### Beta Implementation: Week 5-7

- Design the system structure with UML
- Identify and prioritize features that must be implemented
- Implement the designed system (incremental)
- Testing features after implementation

**Deliverables:** Beta application, Code evaluation 1, UML

### Final Implementation: Week 7-9

- Improve code based on SIG feedback
- Add extra features to the system
- Thorough test of complete system
- **Deliverables:** Final application, Code evaluation 2, UML

**Final Report: Week 8-11**
- Write the final report according to the requirements on Blackboard
- **Deliverables:** Concept Report(week 10), Final Report

**Final Presentation: Week 11-12**
- Make a final presentation
- Book a room for the final presentation
- **Deliverables:** Concept Presentation(Week 11), Final Presentation

### 3.3 Tools

Choosing the right tools is extremely important to guarantee a sound development process. Throughout the entire process the team made use of many different open source tools that significantly improved the development process in general, as well as the quality of the program that was being developed. In this section a short explanation is provided for each of the tools used.

**ScrumDo**

ScrumDo ([www.scrumdo.com](http://www.scrumdo.com)) is a free online Scrum process management tool. We used it to support us in managing the Scrum process and to improve planning during the cycles as well as the complete project. ScrumDo allows adding of tasks to the product backlog, which is basically a list of all tasks (or stories) that need to be completed. All team members may add stories to this backlog, but normally this is the task of the Product Manager. These tasks may be assigned a value that indicates the estimated time to complete this task.

At the beginning of each cycle, tasks from the backlog that must be completed during that cycle or sprint are chosen and added to the sprint backlog. These stories are then visualized on a Scrum Board, of which an example is shown in Figure 3.4. The stories may then be assigned to one or multiple team members to be performed, and separate tasks may be added to the story. Additionally the status of the story is maintained to provide an overview of the progress.

![Figure 3.4: the Scrum Board in action, with some assigned tasks and some in the cycle backlog](image-url)
Eclipse and SVN

The choice for a software IDE was simple. Eclipse requires no introduction and allows development in both C++ for the Client, as well as Java for the Server application. Additionally it allowed us to install different plugins that helped us to develop significantly better code than if we would use a more basic IDE. One of those plugins is the SVN plugin that allowed for code collaboration.

Maven and Jenkins

Maven ([http://maven.apache.org/](http://maven.apache.org/)) is a Project management and comprehension tool. It may be installed as a plugin in Eclipse and ensures that the Java application source code is well structured according to a defined standard. Using a Project Object Model (POM file) maven can manage a project’s dependencies, build, reporting and documentation. Additionally Maven allowed us to use continuous integration testing with Jenkins ([http://jenkins-ci.org/](http://jenkins-ci.org/)). Jenkins builds the application over night and sends a notification in the case of a defect. Additionally it generates reports using plugins that may be installed. We will now describe some of the plugins that we utilized throughout the project.

JUnit Testing

A very important plugin is JUnit. This plugin allows for automated unit testing of the application. In Chapter 5 we will go into more detail on the actual unit testing. Using JUnit in combination with Jenkins, reports were generated that display the amount of tests, and in particular the amount of tests that were unsuccessful. In Figure 3.5 a graph is shown that displays the number of (failed) unit tests against the build numbers.

![JUnit test result trend by build number](image)

Cobertura and Eclemma

To allow for good writing of automated unit tests it is important to assess the code coverage that is realized by those tests. Important classes require complete coverage with exception of simple methods to ensure the correctness of their functionalities. Using Cobertura for Maven and Eclemma for Eclipse the coverage achieved by the unit tests could be assessed. Using these plugins additional tests could be written in case the branch coverage was too low, and redundant tests could be removed. An example Cobertura coverage report is shown in Figure 3.6. We will go into more detail on coverage and testing in Chapter 5.
Checkstyle

Code aesthetics may not matter much for the functionality of the system, but make a great difference when it comes to maintainability of the code. It ensures that code is well readable, consistent and easier to understand. We took this very seriously by making use of the Checkstyle plugin. This plugin may be set up to notify the programmer of mistakes in the programming style. We set the plugin up with accepted default values, such as a maximum line width of 100 characters, and maximum method length of 30 lines. Eclipse provided real-time feedback on the code using Checkstyle and Jenkins generated an overnight report on the amount and type of checkstyle errors, as well as their respective line numbers. Figure 3.7 shows the evolution of the number of checkstyle errors throughout the project development, by build number. It clearly shows that special attention to checkstyle was shown when a code quality evaluation deadline was near. The final version of the application is very clean according to the checkstyle plugin.

Google C++ Testing Framework (gtest)

The Google testing framework provides a handy platform for unit testing the Client C++ code. It requires its own library to be compiled with the code and its own main function. The platform provides an easy way of adding assertions to the unit tests such as ASSERT_EQ and ASSERT_TRUE. It shows the results of the unit tests after running as either failed or passed and shows which specific assertions failed or passed. The results from the Google Testing framework will be elaborated on in Chapter 5.

Google CodePro Analytix

Google provides a Code Analysis Eclipse plugin for Java applications called CodePro (developers.google.com/java-dev-tools/codepro/doc/). This tool allows computation of certain code
metrics such as cyclomatic complexity and the average number of method parameters. Additionally it can find dead code that is not being used and analyze the dependencies between packages and classes. This tool was particularly useful to ensure the quality of the code.
4.1 Global System Design

The system as a whole is developed to filter the information captured by the cameras and to compress it into an understandable graphical representation for the security monitor personal. Additionally, it automates several processes, such as detecting suspicious behavior. This suspicious behavior is not restricted to a single camera. Suspicious behavior may span the view of multiple cameras. This was accounted for by spreading the workload over two types of applications: the client and the server. A very global overview of the system design has already been provided in a previous chapter, in Figure 2.1.

The client runs on a dedicated computer that is directly connected to a security camera. This client application filters out important information such as moving objects and their location relative to the camera. It also classifies the moving objects. In the current prototype distinction is only made between humans and non-humans. Of course these object types could easily be extended in the near future. During the initialization phase between a client and the server the client passes all its essential camera parameters of its camera to the server. These parameters include among others the exact GPS location and the field of view angles. After this handshake the client starts to process and transmits incoming camera frames to the server.

The server reasons and graphically displays the information in a clear, understandable format. In order to do this it must first process and combine the incoming information from the cameras into actual objects that move around the area. For each of these objects a history of GPS locations is stored. The server then starts reasoning about the current situation, given this history of locations, to determine whether or not a suspicious situation is occurring. A flowchart that displays the information flow in the system, starting from a camera frame all the way to an actual alert, is shown in Figure 4.1.

Multiple security cameras may be linked to multiple security monitors. However, only one server is running to receive the information extracted from all the clients. The prototyped system does not channel all the raw camera image data to the server application, it is assumed that there is a dedicated line for this already in place. A schematic overview was given in Figure 2.1.

4.2 Client Design and Implementation

The client is responsible for extracting the information from a real-time input stream of images, required by the server to make reasonable predictions about moving objects and behavior.

4.2.1 The Client Image Processing Pipeline

In order to fulfill its part in the complete system, a series of image processing steps are required. These processes are listed below, and can be found in the client part of the system data flow diagram found in Figure 4.1 as well.
Figure 4.1: a global flowchart of the system. The flowchart describes the data flow of a captured camera frame to firing an alarm

1. Capture a frame from the input stream.
2. Identify pixels belonging to moving objects and shadows in the frame, and return the binary mask containing this information.
3. Identify the different moving objects from the frame using binary mask and label them.
4. For each identified object calculate its location.
5. Classify each identified object as either human or non-human.
6. Transmit extracted information on the moving objects to the server.
7. Repeat steps 1 to 6 as long as there are more available frames (i.e. when the camera is running).

Due to the linear nature of the feature extraction process, the client design was modeled in terms of a finite state machine where each class embodies a part of the process. The top part of Figure 4.1 resembles this state machine.

4.2.2 Client Extendability and Maintainability

An important aspect of the client is the extendability and maintainability of the software, internally or by third parties. The team took this software engineering aspect very serious, and used the abstract class paradigm. The class instances representing each separate process in the image processing pipeline were of the abstract type, but the actual implementations of the class instances
extended these abstract classes. In this way the client behaves like a plug and play model, where one can easily replace functionality by creating a new class that extends the abstract base class, without having to rewrite other parts of the code that use these classes.

The following abstract classes were created based on each step in the finite state machine:

1. **ForegroundDetector**: This abstract class enforces the use of the method `segment` which is responsible for determining which pixels belong to foreground and which do not. Optionally, it could remove shadows as well. The actual implementation does this by modeling the background as a Mixture of Gaussians [32].

2. **ObjectDetector**: This abstract class enforces the use of the method `detect` which is responsible for identifying and labeling the objects in a given binary frame that separates the foreground from the background.

3. **Classifier**: This abstract class enforces the use of the method `classify` which is responsible for classifying an object detected in a frame. This method was purposely kept very generic. In the future a more complex classifier may be used that differentiates between more than just humans, such as cars and cyclists.

4. **TriangulationLocator**: This abstract class enforces the use of the method `locate` which is responsible for extracting the location of a given object based on the cameras intrinsic properties, such as viewing angles and height.

Figure 4.2 depicts the abstract classes and other classes in a UML class diagram. Figure 4.3 shows a sequence diagram that displays inter-class communication in the client pipeline in more detail. Apart from the abstract classes there are three classes which are not abstract. The first is **Camera** which embodies the intrinsic properties of the physical camera. These properties are loaded from a configuration file when the application is started. There was no need to make this class abstract because it is very unlikely that the types of parameters that are stored for a camera will change.

The second non-abstract class is **MotionBox** which summarizes the information that has been extracted from a moving object. This class is not abstract because it is unlikely this class will change, though it may be extended with more extracted information.

The final non-abstract class is the **NetworkManager**, which is responsible for sending a Frame of MotionBoxes to the server. This is done using a basic Socketed connection. The server port and ip are again set in a config file. Because the Server and Client must agree on a communication protocol this class should not be changed. Any changes made to this class require adaption of the Server application as well.

### 4.2.3 Client Image Processing Modules

The client makes uses of the open source C++ computer vision library OpenCV 2.4 ([opencv.willowgarage.com/wiki/](http://opencv.willowgarage.com/wiki/)) for all basic computer vision operations. Currently the Client application makes use of different scientific image processing techniques. In this section we will briefly discuss the used techniques. A more extensive survey of these techniques can be found in Appendix C.

For the separation from foreground to background the client makes use of a background subtraction technique referred to as 'Mixture of Gaussians (MoG)', which is developed by Stauffer and Grimson [32]. This technique uses three dimensional Gaussians per pixel to model the image background. The three dimensions correspond with the red, green and blue (RGB) color values of each pixel.
Figure 4.2: the Class diagram of the final implementation of the Client application

Figure 4.3: a sequence diagram that describes the client image processing pipeline corresponding to the initial design

The main advantage of this background model is its capability to handle multi-modal backgrounds, such as waving trees and objects moving in and out the background scene. Whenever
there is enough statistical evidence for a pixel to belong to the background it is classified as background. The client makes use of a slightly improved version of the original MoG algorithm, which was developed by Zivkovic and van der Heijden [40]. A literature survey of this MoG technique and other background subtraction techniques for outdoor security cameras can be found in Appendix E. The input variables for the MoG algorithm were set according to what worked best in a series of experimental setups intended to simulate all possible situations, but restricted to our available time.

In the resulting MoG foreground ground mask detected objects are identified and labeled using the OpenCV CvBlobs library. This library contains functions for finding the different connected components in a binary mask and for thresholding these connected components. A minimum size of moving objects can be defined in order to suppress noise. The importance of choosing a correct size can be clarified by looking at Figure 4.4. It is clear that filtering based on a correct size results in only the three moving objects being identified and labeled.

![Figure 4.4: a binary image resulting from the Mixture of Gaussians background subtraction](image)

For human classification a Support Vector Machine (SVM) was used in conjunction with an algorithm called Histogram Oriented Gradients (HOG). This algorithm computes a feature vector of the given input image, which is then used by the SVM for classification. The Support Vector machine was trained on the MIT pedestrian dataset.

The HOG algorithm computes the oriented gradient of the input image and overlays the image with a grid of cells. Then, for each cell a histogram is created with bins based on the orientation, where each bin is weighted by the magnitude of the gradient. Next, overlapping blocks are created consisting of adjacent cells and the cells in each block are contrast normalized to get rid of change in light intensity over the image. The histograms are then put into a single vector, which is the feature vector used for classification. For a more comprehensive explanation of the histogram oriented gradient algorithm we refer to the original paper by Dalal and Triggs [9]. In Appendix F a literature survey of multiple classification techniques can be found. The parameters of the feature extraction algorithm HOG were set according to the original paper [9] as they have done research

---

1The MIT pedestrian dataset: http://cbcl.mit.edu/software-datasets/PedestrianData.html
as to what works best in general.

### 4.2.4 Localization of Moving Objects

A crucial part of the Client pipeline is the problem of determining the 3D world coordinates of a point in a given camera image (i.e. object localization). In order to be able to mathematically solve this problem it is assumed that the camera is fixed in space at a particular height and has a fixed field of view. Furthermore the assumption is made that the camera is vertically directed under an angle that lets it fully face the ground, so no horizon is visible in the image. In the context of camera surveillance systems these are all defendable and plausible assumptions that do not pose great limitations to the employability of the system. In this section a location algorithm is derived from a Linear Algebra point of view. An alternative and more illustrated approach uses the Cross Theorem and can be found in Appendix D.

The algorithm locates the objects in space relative to the camera based on the following information: the height of the camera to the ground ($h$), the yaw (vertical camera angle) and pitch (horizontal camera angle) of the camera ($\alpha, \beta$), the fields of view ($f_x, f_y$) and the center image pixel of the underside of the detected object bounding box, which is assumed to touch the ground ($p_x, p_y$). See Figure D.1 to get a visual interpretation of these values.

![Figure 4.5: camera view and required parameters](image)

We first construct the vector $v$ which is initialized as a unit vector pointing along the $z$-axis and then rotated using the pitch and role of the camera such that the vector points through the center of the image plane. The vector is then rotated to the left upper corner of the image plane by rotating a half times the horizontal field of view and half times the vertical field of view.

Now the vector is ready to be rotated into the direction of the pixel in the corresponding image plane. This is done by dividing the horizontal field of view with the width and the vertical field of view with the height of the image. By multiplying the pixel coordinates with these values and rotating we obtain a vector that points into the direction of the pixel in the image plane. The complete
The formula is as follows:

\[ \mathbf{v} = \mathbf{R}_x \left( \alpha + \frac{f_y}{2} \cdot \frac{p_y}{\text{imageHeight}} \right) \mathbf{R}_y \left( \beta + \frac{f_x}{2} \cdot \frac{p_x}{\text{imageWidth}} \right) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \]

where the rotation matrices \( \mathbf{R}_x(\theta) \) and \( \mathbf{R}_y(\theta) \) are defined as:

\[
\mathbf{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}
\]

\[
\mathbf{R}_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}
\]

We can now use the constructed vector \( \mathbf{v} \) to calculate a new two dimensional vector \( \mathbf{d} \) that points into the direction of the pixel, neglecting the pixel’s \( y \) component. We first calculate the angle between the vector pointing down, \((0,0,-1)\), and the vector \( \mathbf{v} \), using the inverse cosinus of the dot product between the two vectors. This angle is then used to calculate the angle \( \phi \) between the ground and the pixel, using the fact that the sum of the angles of a triangle equals 180 degrees. This yields the following formula:

\[ \phi = 180 - (90 + \cos^{-1}(-y_v)) \]

Figure 4.6 illustrates the calculation of \( \phi \).

\[
\phi = 180 - (90 + \cos^{-1}(-y_v))
\]

\[ l = \frac{h \cdot \cos^{-1}(-y_v)}{\sin(\phi)} \]

where \( y_v \) denotes the \( y \) component of vector \( \mathbf{v} \). We can now use this distance to compute the two dimensional vector \( \mathbf{d} \) using the equation below:

\[ \mathbf{d} = \begin{pmatrix} x_v \\ y_v \end{pmatrix} \cdot l \]
4.3 Server Design and Implementation

The server design was challenging as its processes are mostly non-linear, compared to the Client which processes new camera frames linearly throughout the image processing pipeline. The server is responsible for storing, updating, displaying and reasoning about data. This has to happen dynamically and by asynchronous processes. The design decisions that were made to improve code quality and extendability are discussed separately for each component. The essential server components are listed below:

1. **Message Handler**: This component runs in a separate thread and creates connection handler objects that handle the connections from client to the server. Each newly created connection handler runs on its own dedicated thread to assure that the other components will not suffer from computational congestion.

2. **Whiteboard**: This is the central component of the Server application. It is responsible for storing, maintaining and distributing data and embodies all the methods to do so. Information on the cameras, moving objects in the area, areas of interest and alerts are stored on the whiteboard.

3. **Reasoner**: This component reasons about the objects and areas stored on the Whiteboard and adds generates to the whiteboard whenever it derives that a suspicious situation is occurring.

4. **GUI**: This is a collection of components responsible for graphically displaying the data stored on the Whiteboard in a clear format. The design follows a clear separation between the view and control, according to the Model-View-Control (MVC) design pattern. A subcomponent called Map allows for plotting all the latest object locations and location histories on a draggable and zoomable image of the area. The GUI is also responsible for displaying the speed of the objects over time and the alerts that have been sent by the reasoner. Lastly, the GUI has a Console component that displays informative messages on the system status.

5. **Util**: This component consists of static utility methods that are used throughout the entire application, such as a method used for translating locations in a received camera frames to actual GPS locations.

The interaction between the different components is displayed in Figure 4.7. Figure 4.8 shows the highest level class diagram. Later in this section each of the packages in Figure 4.8 will be explained in more detail together with more specific diagrams.

The **Main** class is responsible for initializing the application. Its only function is to create a new Server object and calling the **start** method of this object. The **Server** class can be considered a collection class combining all the large Server components. It has four main attribute classes: **MainWindow**, **ConnectionManager**, **Reasoner** and **Whiteboard**. In the following subsections each of these components are described in more detail, together with their corresponding class diagrams. Figure 4.9 gives a sequence diagram illustrating the general functioning of the Server. The Server is started using the **start** method and will then start running through an infinite loop (i.e. **while**(true){⋯}). In this loop it calls in succession the methods check, reason and repaint of the classes ConnectionManager, Reasoner and MainWindow respectively.

4.3.1 Message Handler

The Message Handler consists of two separate classes called **ConnectionHandler** and **ConnectionManager**. The ConnectionHandler, as its name implies, handles incoming connections of Clients who wish to transmit their data to the Server. The ConnectionManager is in charge
Figure 4.7: A component diagram that describes the initial design of the server as well as interactions between components.

Figure 4.8: The highest level class diagram of the Server showing the main relationships between the server components.

of dispatching ConnectionHandlers as new Clients try to connect such that all connections are managed simultaneously. Each ConnectionHandler is dispatched on its own thread such that parallel processing is possible as to assure maximum efficiency and usage of resources. This in turn ensures a higher processing capacity allowing for a larger system to be run and processed.
Figure 4.9: sequence diagram illustrating the main functioning of the Server

One important part of the message handling system is the protocol that is used between client and server. Its definition is illustrated in Figure 4.10.

Figure 4.10: graphical description of the Server side of the Client-Server communication protocol

4.3.2 Whiteboard Design and Implementation

Figure 4.11 displays the UML Class diagram of the Whiteboard component package. The Whiteboard Class itself is the central component of the Server application. All information is either retrieved from or written to the Whiteboard.

Data storage

The implementation of the Whiteboard is simple. It is mainly a container of **Objects of Interest**, **Cameras**, **Areas of Interest** and **Alarms**. The latter two are Reasoner component objects (see
Figure 4.11: a Class diagram displaying the Whiteboard component package classes

Figure 4.12). An Alarm is basically a message, with a priority level linked to a color. These objects are stored in basic datastructures, such as ArrayLists, and may be retrieved using the usual getters and setters. The only exception is Objects of Interest. These objects are stored in a HashMap to allow for O(1) time complexity retrieval by their label. In addition to the basic getters and setters, a method exists that allows retrieval of a filtered list of Objects of Interest. Any filter may be applied. The default filter returns only those Objects of Interest that exist on the Whiteboard for longer than two seconds and have an average speed of less than 30 km per hour.

Corresponding Client Objects

The Camera, Frame and MotionBox classes are not abstract as they directly correspond to the Camera, Frame and MotionBox classes in the Client application. For the same reason they are not fit for replacement, but could be extended when the Client gets extended, with extra object features for instance. Like on the Client side the Camera class is responsible for storing the intrinsic properties of a camera that belongs to a client application that is connected to the system. One addition is its functionality to store a list of Frames that have not yet been processed by the reasoner. The Frame is a packet that is received from a Client. It has a mark for the date of transmission as well as a camera id of the camera that transmitted the Frame. Additionally it contains a list of MotionBoxes. The MotionBox class corresponds directly with the MotionBox class on the Client side. It is encapsulated in the Frame that transmitted it to the Server. Furthermore it contains the relative (x, y) location to the camera and a humanity score determined by the classifier for a single moving object.
Object of Interest - the main data class

The **ObjectOfInterest** class is the main data representation class of the Server application. An ObjectOfInterest corresponds to a single entity in the monitored area and allows reasoning about this entity. Currently a list of GPS Locations of fixed length is maintained for each object, as well as their cumulative humanity factor. Additionally its methods provide the basic reasoning tools. For instance, retrieving the last \( n \) milliseconds of GPS locations. Importantly, the ObjectOfInterest class is an abstract class, which forces extension. This improves the extendability of the program, as multiply types of Objects of Interest may be identified. Of course this would first require a more complex classifier for the Client application. Currently, the system distinguishes solely between **Humans** and **Unknown Objects**. These different types of Objects carry no meaning. Humans may be of different types, such as Guards or Pedestrians and can be performing different behavior such as waving or laying down.

4.3.3 Reasoner Design and Implementation

The **Reasoner** package puts the actual intelligence into the System. In Figure 4.12 an UML Class diagram providing an overview of the Reasoner component package is displayed. The Reasoner class itself combines all the separate reasoning components, and ensures that they are executed in the correct order during the reasoning cycle. First, new camera Frames are processed by the **Object Identifier**. Then irrelevant objects are cleaned. Next the **Areas of Interest** are evaluated and finally the reasoning rules are applied. The Reasoner also allows for communication of the reasoning components with the Whiteboard and its stored objects. Moreover, it allows for addition of new rules. As becomes obvious from Figure 4.12 extra attention was paid to provide means for extendability. Now, the separate reasoning components will be discussed in more detail.

Object Identifier

The responsibility of the **Object Identifier** is to match the MotionBoxes in the incoming frames from the Client applications to actual Objects of Interest that are followed by the system. It is extremely important that this happens correctly. A systematic figure that explains the process of the Object Identifier can be found in Figure 4.13.

The Object Identifier loops through the Camera objects on the Whiteboard to poll for any unprocessed Frames (step 2). The MotionBoxes contained by the Frames are then split up and each considered separately (step 3). Every MotionBox contains a distinct moving object that was spotted by the camera.

Based on the corresponding GPS Locations of the objects captured in the MotionBoxes, the Object Identifier attempts to match the MotionBox to an Object of Interest (step 4). It does so by evaluating **LocMatchers** on the GpsLocation of the MotionBox and an Object of Interest in the system.

As shown in Figure 4.14 the LocMatchers implement an interface, **ILocMatchers**, to allow for extendability by adding new LocMatchers to the system. Currently the Object Identifier checks two matchers. The first matcher is basic: the MotionBox location is within two meters of an actual object and was spotted at most two seconds after last seeing the object by the same Camera. The second matcher allows for detection of objects that move from one camera to another. When it picks up a MotionBox it compares it to objects that disappeared from another Camera. Based on the last known GpsLocations it estimates a direction and speed, by using methods from our Util package, to then evaluate if there is statistical evidence to believe that it is an already known object that moved here over the time it was out of view of the cameras.

When a match occurs, the MotionBox is added to the corresponding Object of Interest as the most recent GpsLocation. Furthermore, the time the object was last seen and the humanity ratio of
the object are updated. If no match occurs a new UnknownObject is created and the GpsLocation of the MotionBox becomes the first known location. Additionally, the Object Identifier converts UnknownObjects into Human Objects when the humanity ratio passes a defined threshold.

**Reasoning Rules**

In order to place alarms onto the Whiteboard (one of the main goals of the system) the Reasoner must be capable of evaluating whether a suspicious situation occurs. In Figure 4.15 an overview of the Rules package is given. The reasoner is in fact an Expert System that uses human defined rules to reason on the situation.

Two types of rules are distinguished. Normal reasoning rules which apply solely to an ObjectOfInterest, and comparative rules which allow comparing an ObjectOfInterest to a list of all other ObjectsOfInterest currently on the Whiteboard. Additionally a 'cooldown' may be set, indicating how often to display an Alarm. Both types of rules are implemented using an interface. This allows for very easy extension of the system with additional rules by implementing such an interface and adding the rule to the Reasoner. The type of rule determines the amount of data from the Whiteboard that is allowed to be accessed. All rules make use of the available history of GPS locations that is available for the objects. This history of GPS locations is also displayed on the map in the GUI. In Figure 4.16 multiple objects and their paths are displayed in a part of the Server GUI. This GUI will be discussed in more detail later in this chapter.

Two reasoning rules, one of each type, have been implemented as a proof of concept. The **Is-Running** rule applies only to Humans. It will place an Alarm on the Whiteboard when the average
speed of the Human increases to above a certain threshold. This average speed is determined using the Util package method for determining the mean and variance of the speed of the object.
over a certain timeframe.

The **IsFollowing** rule is a little more complicated. It evaluates the average distance between all pairs of objects over a certain timeframe. It also computes the mean direction of the object, to evaluate whether it is walking in the direction of the other object. In order to be considered following the other object, two criteria must be met. First, the average distance to the potentially followed object remains within predefined boundaries (relative to the following distance). Second, the mean direction of the object must be directed towards the potentially followed object. This may differ by at most a predefined number of standard deviations in the angle of the direction. In Figure 4.16 a situation is displayed where four objects are moving through the area. Based on the location history of the objects the system derived that object 1 is following object 0.

A final rule that should not be removed, but still allows for extension, is the **IsUnnecessary** rule. The rule itself is in face necessary, as it is required for cleaning up irrelevant objects of the Whiteboard. Without this rule the Whiteboard would be congested with objects that are not of interest, or no longer relevant to the system. Currently this rule removes those objects that have not been seen for a timeframe that is larger than the total timeframe they have been seen, plus two seconds to account for delay.

**Areas of Interest**

A second reasoning component besides rules are the **Areas of Interest**. An Area of Interest is basically a predefined polygon of (x,y) GPS locations on the area that is being monitored. By using a winding number algorithm from the Computer Graphics field it may be checked if an ObjectOfInterest is currently within the defined polygon of the area.

Again when designing the Areas of Interest we focused on extendability. It is easy to define new areas due to of the structure of the Area package, as is displayed by Figure 4.17. A basic abstract Area of Interest class allows defining a polygon and setting a corresponding alarm message and priority. Extensions of this class may then implement their own detect method, similar to implementing the Rule interface for the general reasoning Rules. An example implementation is given in Listing 4.1.

As a proof of concept we have added three areas, to show how simple it is to define new areas. The **SpeedLimit** checks for objects within the area with an average speed (computed using the Util package) over a definable timeframe that exceeds the preset speed limit. The **RestrictedArea** and **Water** areas both fire an alarm whenever an object enters the Area of Interest. They only differ in the message and alarm priority caused by the Area.

Despite the fact that the Areas package resides within the Reasoner package, the actual Areas
Figure 4.16: a screen capture of the system showing a situation where object 1 is following object 0.

Areas of Interest implement the Serializable Java interface, to allow writing of the object as well as any possible extension to a file. We chose to add the Areas to the Whiteboard, due to dependency issues. To prevent the GUI depending on the Reasoner for drawing the Areas, they were added to the Whiteboard to which it already had access.
Areas

+ detect(w : Whiteboard) : boolean
+ inside(point : Point2D.Double) : boolean
+ updateAlarm(o : ObjectOfInterest, offSet : double) : void

- alarm : boolean
- priority : Priority
- defaultColor : Color
- alarm : Alarm
- message : String

Areas

+ detect(w : Whiteboard) : boolean

- speedLimit : double
- detectedSpeed : double

Listing 4.1: A typical area of interest detect method implementation

```java
public Alarm detect(ObjectOfInterest o) {
    GpsLocation lastLoc = o.getLocations().getLast();
    if (inside(lastLoc.getLoc())) {
        detectedspeed = SpeedMeter.getAvgSpeed(o, 5000)[0];
        if (detectedspeed > speedlimit) {
            setAlarm(true);
            return new Alarm(o.getLabel(), lastLoc.getSourceCamId(),
                              getMessage(), getPriority());
        }
    }
    setAlarm(false);
    return null;
}
```

4.3.4 Utility Methods

The Util package is not very interesting in terms of system design, but it does fulfill an important role within the Server application. Its class diagram is shown in Figure 4.18.

Like the Util package in Java, the Util package for the Server application provides several useful static methods that may be used throughout the entire application. An example of such a method is given in Listing 4.2. This particular fragment allows for computing the distance in meters between two GPS Locations. It uses the haversine formula to determine the great-circle distance between two points [36]. This formula is defined as follows:

\[
a = \sin^2 \left( \frac{\Delta \text{lat}}{2} \right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2 \left( \frac{\Delta \text{long}}{2} \right)
\]

\[
c = 2 \cdot \text{atan}2 \left( \sqrt{a}, \sqrt{1-a} \right)
\]

\[
d = R \cdot c
\]

where R denotes the earth's radius.
Some util methods handle the IO of the application, such as writing and reading Areas of Interest to a config file. Other util methods play an important role in the reasoning process. Examples of such methods are the methods that estimate mean and variance of the speed and direction of an ObjectOfInterest over a timeframe of a number of milliseconds. The example for calculating speed can be found in Listing 4.3. It uses other util methods for determining the speed between all pairs of consecutive locations and then calculates the mean and variance using well known statistical methods.

Listing 4.3: the \texttt{avgSpeed} util method

```java
class Util {
    public static double distance(Point2D.Double p1, Point2D.Double p2) {
        double dLng = Math.toRadians(p2.x - p1.x);
        double dLat = Math.toRadians(p2.y - p1.y);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
                   + Math.cos(Math.toRadians(p1.y)) * Math.cos(Math.toRadians(p2.y))
                   * Math.sin(dLng / 2) * Math.sin(dLng / 2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        return EARTH_RADIUS * c;
    }
}
```
while (!locations.isEmpty()) {
    GpsLocation dst = locations.pop();
    double speed = calcSpeed(src, dst);
    sum += speed;
    sumsq += speed * speed;
    src = dst;
}

// calc mean
result[0] = sum / (double)(amount - 1);

// calc variance
result[1] = (sumsq - sum * result[0]) / ((double)(amount - 2));
return result;

4.3.5 Graphical User Interface Design and Implementation

Before explaining the design and functionality of the Graphical User Interface (GUI) in detail, we will first briefly discuss the final design of the GUI. Despite the fact that the system, as required, does not allow for much direct user interaction, we found it important that the GUI is easy understandable for the security personnel. In order to achieve a final result for the GUI design a User Centered Design approach was used. After meeting with the supervisors to create specifications for the GUI an initial mockup (as shown in Chapter 2) was shown to the supervisors. Based on their feedback and requirements during the weekly meetings this design was further refined and shaped into the final design.

Graphical User Interface Explanation

The final GUI design is shown in Figure 4.19. The GUI consists of four main components. The first component is the Map. In the Map Objects of Interest that are currently in the area are plotted in their unique colors. In Figure 4.19 two objects are currently visible: Unknown objects 1 and 2, drawn in blue and pink respectively. The current location of each of the objects is marked with a red dot and the gps location history is shown as a tail in the object’s color. Besides the two objects two Areas of Interest are also visible in Figure 4.19: a speed limit area and a water area. Each type of area has its own predefined color when not firing an alarm. When firing an alarm it assumes the color corresponding to its alarm priority.

A second component is the Console. This console displays informative messages on the status of the Server application and the connected Client applications. In Figure 4.19 a simulation is run so the Console indicates that two simulation files, called ‘Test Camera 1’ and ‘Test Camera 2’, have been added. Additionally it mentions that the map configuration and definitions of Areas of Interest were successfully read from their configuration files.

A third component is the Speed History graph. This graph displays the average speed over time for all the Objects of Interest that are currently in the monitored area. The line for an object is drawn in its corresponding color, such that the colors for the Map and Speed History correctly correspond.

A final and crucial component is the Alarm Table. This table displays all fired alarms. For each fired alarm it shows its priority and time, the object that triggered it, the type of alarm and the camera that last detected the object that caused the alarm. The row containing the alarm information is assigned a color corresponding to the priority of the alarm.
The Main Window

The GUI package UML class diagram is shown in Figure 4.20. The main class in the GUI package is the **MainWindow**. The MainWindow contains all the individual GUI components, corresponding with the original GUI mockup that was shown in Chapter 2. The MapPanel, PlotPanel, InfoTablePanel and Console are each added to the MainWindow.

One responsibility of the MainWindow is to handle the layout. This is done using a Java GridBagLayout, which divides the window into a grid in which each cell contains a component. It may also be specified how extra size is distributed when resizing. This ensures that the GUI adapts easily to different resolutions and resizing, compared to a hardcoded GUI.

A second responsibility is ensuring that each of the components in the MainWindow gets updated with the correct Whiteboard information. The MainWindow handles the communication of the GUI package classes with the Whiteboard. By calling the `update` and `repaint` methods all individual components are updated and repainted accordingly.

Color Dispenser

The **Color Dispenser** was introduced to ensure ObjectsOfInterest have a unique color attached to their Label. This color may be used throughout the GUI. The Color Dispenser starts with a pool of available Colors. When a GUI component attempts to retrieve the color of an Object its label is used to look up the corresponding Color in a HashMap. When no Color is assigned a color is retrieved from the Color pool and attached to the Object label in the HashMap. Otherwise the Color is retrieved and returned. Whenever an Object is no longer on the Whiteboard its Colors will be freed and put back into the Color pool. The MainWindow creates the ColorDispenser and ensures that all GUI components make use of this same ColorDispenser.
Figure 4.20: this image shows and labels each of the components in the GUI design in the actual implementation of the system

Map Panel

The **MapPanel** is the most advanced GUI component. It is responsible for displaying a map of the area with the ability to move it around and zoom in and out. Because different areas should be monitored, the map information is loaded from a configuration file by using IO Util functions. Additionally, the MapPanel is responsible for displaying the current locations of the Objects of Interest in the area, their corresponding location histories and the Areas of Interest. The result of the MapPanel can be seen in the top left corner of Figure 4.22.

The MapPanel makes use of several key components from the Map package, which is a sub-package of the GUI component package. Figure 4.21 shows a class diagram of the Map package. Its components will now be described.

The first component is the **CustomMap** class. This class stores information on the map of the area that is being monitored by the system. For instance, the GPS location of the center of the map as well as the width and height are stored. It also stores the zoom levels of the map and a list of markers that must be painted.

A second and more complex component is the **MapPainter**, which ensures that the image of the Map is displayed in the right proportions, and allows for zooming and dragging of the map. It does this based on the information in the CustomMap component. It also provides methods for translating GPS Locations to locations on screen. The MapPainter makes use of the **Drawing** class. This class is responsible for storing all the specific drawing information in order for the map image to be drawn correctly in the Map panel.
The third component is the **WhiteboardPainter**. This class is responsible for drawing all the Whiteboard concepts to the MapPanel (i.e., Areas, Objects of Interest, etc.). When the MapPanel is forced to update by the MainWindow, it is provided access to the Whiteboard information. This Whiteboard information is then transferred to the WhiteboardPainter. It draws all Areas of Interest by drawing a polygon using the GPS locations that define the area of interest. It then fills the polygons using the desired color specified in the Area of Interest. A different color may be specified in case an alarm is fired by the area. Labels 4, 5, and 6 in Figure 4.22 display Areas of Interest on the MapPanel.

For each ObjectOfInterest, the most recent location is painted as a larger red marker. Next, the history of GPS Locations is painted in the Color corresponding to the object label, which is retrieved from the Color Dispenser. This is done by adding a marker for each of the locations in the history, and then linking them with a line using the Java Graphics `drawLine` method.

Finally, the Cameras that are connected to the Server application are drawn on the Map. The origin of a Camera is depicted by label 2 in Figure 4.22. The black dotted lines with label 3 indicate the viewing angle of the camera to give the security personnel a clear image of the camera’s viewing frame.

The final component is the **MapController**, which extends the MouseInputAdapter class and implements the ActionListener interface. Whenever the map is double-clicked, it will update the CustomMap zoom level and instruct the MapPainter to update the image. Alternatively, the zoom buttons labeled with 1 in Figure 4.22 may be used for this functionality. Similarly, when the mouse is dragged, an offset is added to the map center in the CustomMap, after which the MapPainter is forced to repaint the map.
Plot Panel

Though the PlotPanel was intended for plotting general properties of the ObjectsOfInterest over time, it was eventually solely used for plotting the objects’s average speed over time. When the PlotPanel is updated it is provided an up-to-date list of ObjectsOfInterest that are currently in the area.

For each object a new average speed is calculated and linked to the time of measuring. Then these values are added to the custom SpeedHistory data structure for each object. This SpeedHistory contains a history of average speed and time of measurement tuples. Additionally it contains the label of the corresponding object and ensures that a definable maximum history length is maintained. For all objects that are no longer on the Whiteboard the history is removed.

After the SpeedHistories for all objects are updated, they are plotted by using the JMathPlot library (code.google.com/p/jmathplot/). The result of the plot is displayed in the bottom right color of 4.19. The speed history of each object is painted in the corresponding color, by using the Color Dispenser.

Alarm Table Panel

This GUI component is a JPanel that contains a JTable. This table is adapted in order for alarms to be added to the table. The table shows the priority of the alarm, the date when the alarm occurred, the ObjectOfInterest involved in the alarm, the alarm message and the Camera that spotted the situation. By implementing a custom Java TableCellRenderer a row containing an alarm is painted in the color that corresponds to the priority of the Alarm. In Figure 4.19 a medium priority alarm is displayed in the table (under the Map overview). Whenever the InfoTablePanel is updated by the MainWindow it is provided a collection of new alarms to add.
The Console is a very simple GUI component. It is a JPanel containing a JTextArea. Through a static method messages may be added to the JTextArea throughout the entire Server application. After adding a new message the Console will repaint the JTextArea. In Figure 4.19 the Console can be seen at the top right of the window, with several messages printed to it.
This Chapter discusses the different type of testing methodologies and tools that were used for testing both the Client and Server. The tests are globally divided into two categories: **automated tests** and **field tests**. By field testing we refer to manually testing the software. This testing consists of testing individual methods, testing multiple methods and classes, and high level testing the system as a whole. We will be using the following testing constructs to test our system to a level of proper adequacy:

- **Unit testing**: Testing individual methods for correctness over their input space. This is our primary form of testing and assures a reasonable form of system correctness from classes that have simple in- and output behavior.

- **Scenario based testing**: Testing the system as a whole based on common or uncommon scenarios. In our case we will be testing scenarios using the Client and Server together based on pre-recorded camera footage and artificially created data that mimics input data from the Client to the Server.

- **System integration testing**: Testing the software with other software components (e.g., third party software). In our case we will be testing the interaction between the Client and the Server.

- **Software Performance Testing**: We test the limits of our software in terms of how many active functional cameras the server supports without displaying forms of lag.

- **Usability Testing**: Testing the interface to see if it is easy to use and to confirm that the displayed information is clear.

Finally we will discuss the code analysis result. This embodies information about the code itself related to the complexity and its relation to extendability and maintainability. For example the inter dependability between classes in the code will be analyzed, which we of course liked to keep to a minimum.

### 5.1 Automated Testing

In this Section we will discuss the automated testing we have used throughout the project for both the Client and Server applications.

#### 5.1.1 Client Unit Testing

The Client proved to be the least prone to testing, as its main functionality rests heavily on image processing. Thoroughly testing such a system requires a large array of test data in the form of images. This test data is large in size, as it requires many frames, and is complex to generate. The only practical way of attaining test data would be to shoot test movies in the field which is both challenging, time consuming and impractical to even impossible in some cases. For this reason
we rely heavily on the use of scenario based testing. However, some unit test cases were written for those Client methods that do not require image based input.

The Unit tests were executed using Googles Test suite for C++ (code.google.com/p/googletest/). The MotionBox class was tested by creating unit tests for each set method, asserting that the get method retrieved the same value as which was set. The Camera class was tested in the same way using artificial settings files generated manually. Lastly, the TriangulationLocator class was tested. This class consists out of more then just get and set methods. It actually implements an algorithm to locate an object based on its bounding box relative to the camera. The following test cases were considered:

- the object is *inside* the camera frame
- the object is *on the border of* the camera frame
- the object is *in the corner of* the camera frame.
- The object is *not in* the camera frame.
- different camera field of views.

### 5.1.2 Server Unit Testing

Although the Server is more suitable for automated testing, not every package in the Server application is well fit for Unit testing. Unit tests depend on simple input/output behavior. The parameters must be simple in order to allow for testing. The reasoner package suffers from problems similar to the client. The input for the methods consists of multiple objects, with large histories of GPS Locations. Test cases for these methods are extremely time consuming to develop.

Unit testing for the Java Server application was performed using JUnit. JUnit is the standard framework for Unit testing in Java. An example of a JUnit test case is displayed in Listing 5.1. Especially packages consisting of data classes and exact methods, such as the Whiteboard and Util packages, were fit for unit testing. The MapPanel class from the GUI package was unit tested as well. Where the other GUI components do not provide means for interaction with the security personnel, the MapPanel allows for dragging and zooming in and out. These actions were automated using Unit tests to see if no errors were thrown.

```java
@Test
public void testAddToManyLocations() {
    int length = 1;
    Human h2 = new Human(length);
    h2.addGpsLocation(g2);
    h2.addGpsLocation(g1);
    assertThat(h2.getLocations().size(), equalTo(1));
    assertThat(h2.getLocations(), hasItem(g1));
}
```

When developing Unit tests for the Server application we used the Eclemma Eclipse plugin for assessing the coverage that was achieved through automated testing. Though the goal is to achieve maximum coverage, there must be a balance between effort and achieved coverage, as is also stated on the JUnit official webpage: "While it is usually better to test more, there is a definite curve of diminishing returns on test effort versus code coverage".

In Figure 5.1 the Eclemma coverage report of the Server application Unit testing suite is displayed. As becomes clear the coverage is over 80% and thus reaches the desired 100%. A portion of the
lines of code within the Whiteboard classes consists of get and set methods and relatively large
hashCode methods. Testing of such methods is generally considered irrelevant. All the important
functional methods within the classes are fully covered however.

5.2 Field Testing

Field testing is done by performing manual tests or assessing the outcome of a test manually. When appropriate we will refer to the requirements from Chapter 2 so it can be confirmed if these requirements are met by using the outcome of the test.

5.2.1 Field Testing the Client

We will first describe the manual field tests that were performed for the Client application.

Testing Localization Accuracy

The localization algorithm accuracy is not only dependent on the accuracy of the camera specific properties, it depends also on other camera aspects. There are many aspects that can create an error in the localization such as warping of the image due to lens inconsistencies. The sum of these errors (together with the measurement errors) have been measured to get an estimate of the total average error in a camera. Table 5.1 shows the result of this empirical research. A frame from the video used to test the localization accuracy can be found in Figure 5.2.

Based on this empirical research, taking practical issues in mind, we conclude that the maximal deviation of the Euclidean distance tolerated from the real position is never more than half a meter, so that quality requirement Q9 is met.
Table 5.1: the results of an empirical research into the accuracy of the localization. The left column represents the location measured and the right column represents the value returned by the algorithm.

<table>
<thead>
<tr>
<th>Empirical value</th>
<th>Theoretical outcome</th>
<th>Difference of euclidean distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,11)</td>
<td>(2.104,11.016)</td>
<td>0.105224</td>
</tr>
<tr>
<td>(0,11)</td>
<td>(-0.138,11.112)</td>
<td>0.17773</td>
</tr>
<tr>
<td>(-2,11)</td>
<td>(-2.180,10.814)</td>
<td>0.258836</td>
</tr>
<tr>
<td>(2,13.25)</td>
<td>(1.851,13.233)</td>
<td>0.149967</td>
</tr>
<tr>
<td>(0,13.25)</td>
<td>(-0.180,13.307)</td>
<td>0.188809</td>
</tr>
<tr>
<td>(2,14.25)</td>
<td>(1.702,14.418)</td>
<td>0.342094</td>
</tr>
<tr>
<td>(0,14.25)</td>
<td>(-0.265,14.640)</td>
<td>0.471514</td>
</tr>
<tr>
<td>(-2,14.25)</td>
<td>(-2.316,14.207)</td>
<td>0.318912</td>
</tr>
</tbody>
</table>

Testing the Background Subtraction

Background subtraction may be tested by comparing the results of the background subtraction to an image in which the ground truth foreground is selected. This is a time consuming process however. Instead we chose to manually assess the quality of the background subtraction, by simultaneously displaying the source image and the background subtraction result as displayed in Figure 5.3.

By manually assessing the quality of the result we could fine-tune the parameters until the achieved result was clean enough for consequent image processing processes.

5.2.2 Testing the Server

All parts of the Server that were not suitable for automated testing, were tested manually. In this section we describe the manual tests that were performed in order to ensure proper functioning of the Server application.
Scenario Based Testing and Debug Server application

As discussed in the Unit Testing Section the Reasoner is not very suitable for Unit Testing because it requires large and complex input data. In order to allow for testing of the Reasoner rules and areas we have developed an extension to the Server application, called the **Debug Server**. The Debug Server extends the GUI MapPanel component by adding a set of extra user interaction functionalities: drawing of areas, addition of test cameras and drawing of object paths to be linked to a selected test camera. Currently the Debug Server is an unofficial extension of the application and used for testing purposes only.

In Figure 5.4 the Debug Server is displayed in the process of defining an area. The area can be defined by choosing a type of area (see Figure 5.5) and then clicking on the map to define a polygon. When the area is complete it can be saved by clicking the 'Save' button. Areas may also be selected and removed by clicking on the 'Delete' button which becomes enabled when an area is selected. When the Debug Server is shut down the areas are written to a file using the IO Util package. This file is loaded whenever the DebugServer or Server are launched, ensuring that the defined areas are instantly used by the actual Server application.

Besides defining Areas of Interest the Debug Server allows for an even more important debugging tool. It is capable of generating simulation files. First one or more cameras have to be positioned (drawn) on the map. This can be done by clicking on the 'New Camera' button. A dialog box will pop up asking for the desired horizontal field of view of the new camera (see Figure 5.6). Next
the desired update speed, in milliseconds, of the camera has to be specified (see Figure 5.7). Once these values have been entered the user needs to specify the location of the camera by clicking somewhere on the map. A red point will be drawn on the map, indicating the location of the camera, together with a black line between this red point and the current mouse location, indicating the viewing direction of the camera (see Figure 5.8). When the user clicks again the current specified field of view is saved and the camera is drawn on the map (see Figure ??)

Once the cameras are added a camera may be selected, by clicking on it. To add an object path to the camera the 'New Path' button has to be clicked. This will show a dialog box asking for the path settings (see Figure 5.11). Paths can then be drawn by clicking on the map as shown in
Figure 5.10. Between each two clicks the specified number of interpolation points is interpolated between the two points, with some Gaussian noise added to it in order for the points to not lie on a straight line. In this manner multiple cameras and multiple object paths may be drawn on the map. When closing the Debug Server these Cameras and object paths are translated into simulation files. Each simulation file represents a camera, consisting of MotionBoxes containing the object paths, and the intervals at which the Motion Boxes should be processed.

The Debug Server may be started with the simulation files as program parameters. The simulation files can then be played back and processed as if the information is received from actual cameras as displayed in Figure 5.12.
In order to test the functioning of our different Areas Of Interest and reasoning rules we have developed many different scenario’s using the Debug Server. These scenarios consisted of object
tracks that should lead to alarms being fired. By playing back the scenario it can be confirmed if the Server application actually fires an alarm correctly. All currently implemented areas and rules have been scenario tested, and have been confirmed to work as intended. Many different requirements from Chapter 2 that can be proved using a demonstration, such as requirements F8, F9, F10 and F11, have been tested using Scenario Based testing. For requirement F11, a simulation file was created consisting of four objects. One object was walking around, following a random path with curves. Another object followed a path that should be considered as following the first object. Two other objects were added that cross the paths of the first two objects to ensure that no false alerts are fired when more objects come into play. Many more simulations similar to this ‘following’ simulation were created using the Debug Server. The Debug Server allows for more scenarios to be very easily generated.

GUI: User Acceptance Testing

Due to the large automatization factor of our system it does not allow for much user input. Most of the present interaction is directed from the System to the user, rather than input from the user to the System. As mentioned before the GUI was designed by following a user centered design. During the weekly meetings feedback was provided in order to improve to GUI to its current state. To ensure that the final GUI design meets the requirements a user acceptance test was performed for the GUI. Test subjects that have no affiliation with Computer Science were asked to watch the Server GUI and asked what exactly it is that they saw on the screen. In this way we could determine if the GUI is clear and if the information is interpreted as intended. The results from these tests for the individual components are listed below:

- **Map Panel:** The map GUI component was generally easily interpreted and understood, probably because it works on the same principles as Google maps. The map itself, as well as moving around by dragging and zooming in and out using the buttons were considered self-explanatory. The locations of the Cameras, indicated by dotted lines displaying its field of view, were easily identified by all users as well. The fact that the mac address of the camera was used as a unique identifier, and that it was consequently drawn on the map beneath the position of the camera, confused some of the testers. Each user deduced that the red dots corresponded the current positions of objects in the area and that the colored tails represented the paths of these objects. The different colors assigned to different objects were considered to be of much help when tracking an object manually. Finally, the Areas Of Interest were easily identified thanks to the use of distinct colors and a label that marks the type of area. The fact that an area lights up in a color corresponding to the priority whenever it fired an alarm was considered to be useful.

- **Alerts Table:** The alert list was said to be clear in terms of readability and interpretability. The different colors for the different priorities of the alarms were said to be a clear indicator of the different priorities. A suggestion was the fact that the introduction of a new alert does not trigger a sound effect. Without a sound effect the user is obliged to look at the screen continuously. When the user looks away from the screen and a new alert is added it might be difficult to recognize the fact a new alert has been added in a large list of alarms. Another suggestion was to introduce some graphical mechanism that indicates that an alarm was recently added, by for instance coloring it differently or making it flash.

- **Speed History Graph:** Not all subjects found the Speed History graph to be clear at first notice. After reading the values on the axis of the graph all testers identified the meaning of the graph however. In order to improve this a text label was added to the Speed History Graph panel indicating the meaning of the graph. Moreover, not all test subjects understood the significance of the Speed History Graph regarding the functionality of the system. They
did not see why this information would help them make better judgement of a suspicious situation.

- **Console Panel**: The need for a console was not apparent for the test subjects at first. After a short explanation they better understood the significance, as it let them know about system updates such as new cameras that connected and critical system failures that may need to be addressed.

**Performance Testing**

In order to test the performance of the system we created a basic performance testing setup. Because it was impossible to perform this test at an actual scale by using a large number of cameras, we made use of simulation files generated by the Debug Server. These simulation files each represented a camera transmitting the location of a single object at a rate of two frames per second. This object is constantly zig-zagging in and out of a restricted area in order to simulate unrealistic heavy reasoning.

Next, we added multiple of these simulated cameras to the Server application. Then, we calculated the amount of cycles that were performed while running the simulations, and divided this by the total time span of the simulations. We then calculated the average processing speed, measured in processing cycles per second, while the simulations were played. By incrementing the number of cameras that are simulated we increased the load on the Server application. This test was executed on a 3.1 GHz Intel Core i5-2400 quad-core CPU.

<table>
<thead>
<tr>
<th>Number of cameras/objects</th>
<th>Low reasoning load</th>
<th>Heavy reasoning load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000+</td>
<td>1000+</td>
</tr>
<tr>
<td>5</td>
<td>1000+</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>1000+</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>763</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>567</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>332</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.2 shows the results of the performance test. Under low computational load the Server application scaled extremely well maintaining reasoning cycle speeds of over 300 cycles per second for 30 cameras and objects simultaneously. However, when a large amount of reasoning computations was performed on each of the objects (object in an area and following rule was enabled), the reasoning cycle speed drops to 3 per second. The Server application managed to process information of at most 20 cameras simultaneously under bad conditions, while maintaining an average processing speed of 6 cycles per second. This ensures that quality requirement Q1 is met for monitoring security areas with up to 20 cameras when a single PC is used for the Server application.

**5.2.3 Testing the Client and Server**

Creating test footage is a challenging task, because it requires accurate measurements of the position of the camera and a suitable environment. It is also very time consuming to shoot test footage. We have therefore decided to create one test case which includes as many aspects as possible.
Scenario Based Testing

We tested a couple of different scenarios (comprised in one test footage) listed below:

- Noise introduced by wind. The tree in the footage has moving leaves which generates constant periodic motion. This motion should be filtered out over time by the Mixture of Gaussians model.

- Noise generated by camera movement. The camera moves slightly generating large blocks of noise in the footage. The system should recover from this as fast as possible.

- Classification was tested by introducing humans in the frame and cyclists. The system should not label the cyclists as humans.

- The footage includes a human moving over predefined locations who's real world location relative to the camera are known. In this way it could be checked if the localization algorithm works correctly.

- The footage includes a human walking after which it runs which should trigger a human is running alert.

Each scenario described above passed the test. However, the system does have minor difficulties differentiating humans from cyclist. In practice this could lead to moderate amounts of false running alarms from clients observing areas containing a lot of cyclists.

System Integration Testing

Using the same footage still, we have tested the client server integration. We found that the client connects to the server correctly, and that with a good connection no frames were lost. With a lagging connection however frames did get lost, because the connection could not always be established. This was to be expected, however. One aspect that we wanted to test, but have not been able to due to a lack of resources, is the capacity of clients able to connect to the server and the capacity of the network to handle such large amounts of data.

5.3 Software Improvement Group Feedback

At the moment of writing this report we have received feedback for both the intermediate and final evaluation. The feedback can be found in Appendix G, and is written in Dutch. In this Section we will summarize the feedback, and state our improvements to process the feedback.

5.3.1 Intermediate Feedback

According to the Software Improvement group our intermediate product scored 5 out of 5 stars on the maintainability model. This indicates a very good maintainability of the code. The two aspects that got a lower score were:

- **Unit Interfacing**: The number of parameters in methods must be low. A high number of parameters results from a lack of abstraction. The Server side of the system scored lower in the MotionBox, Camera and Frame objects. These objects are frequently passed to methods together. The relationship between these objects should be made clearer.

- **Unit Size**: Some methods consist of too many lines of code. Smaller methods are easier to understand and make the system more maintainable. Separate pieces of functionality can be identified in longer methods, such as the Client Camera constructor, and split up into separate methods to reduce Unit size.
5.3.2 Improvements

Based on the feedback we received we put through several improvements. First of all we started using the Google CodePro Analytix Eclipse plugin. This plugin has already been previously described in Chapter 2. It allowed us to compute certain code metrics ourselves, which the Software Improvement Group uses for evaluation.

Examples of code metrics we evaluated are Unit Size, Unit Interfacing, Unit Dependencies and Cyclomatic Complexity. Figure 5.13 shows a Dependency analysis by CodePro. It shows that the dependencies within the project are well within boundaries. It also shows how the Whiteboard is the central component, and that the util package is commonly used throughout the entire application.

![Figure 5.13: a CodePro dependency analysis](image)

In order to improve the code according to the Software Improvement Group feedback we have split up all methods that were longer than 20 lines. These methods were identified using the Code-Pro and Checkstyle plugins. The final program has only a few methods with over 20 lines of code and contains no methods with more than 30 lines of code.

Additionally, we reworked many constructors and dependencies between classes. In the particular case of the MotionBox, Frame and Camera classes we clarified the relationships between the classes. From the code it now becomes obvious that the Camera contains Frames, and that Frames contain MotionBoxes. Each of the Objects has a reference back to its container, to prevent the objects from having to be passed to methods together. We also ensured that methods required only those objects that are absolutely nescessary. Previously the entire Whiteboard object was passed as a parameters. This caused many objects to be heavily dependant of the Whiteboard, but this issue is solved now as shown in Figure 5.13.

5.3.3 Final Feedback

The final feedback stated that we managed to maintain our 5 out of 5 star rating for the maintainability of the code. Although the system size increased by 59% the maintainability of the code only decreased by a very small amount. On the aspect of Unit Interfacing and Unit Size the code has approved according to the feedback. Unfortunately, the system scored slightly lower on Module Coupling. This is the part of the code that has relatively many calls. This was unavoidable however, due to our Whiteboard design pattern.
Conclusion

For this Bachelor Project an Intelligent Multi-camera Video Surveillance system has been developed, with a particular focus on the Netherlands Defence Academy area of the Koninklijk Instituut voor de Marine. This system is a proof of concept, capable of gathering information from multiple surveillance cameras, to then reason about the situation. When suspicious behavior is detected, an alarm is displayed for the security personnel.

The system was developed by following a scientific and agile approach using Scrum, dividing the process in several phases. First, meetings were held with the domain experts and a specification and requirements were made based on the assignment. Next, a literature survey was performed to determine what techniques to use during the project, and an initial design of the system was made. Then, the system was implemented incrementally using Scrum. Finally, the system was tested to ensure correct functionality and to ensure the set requirements were met. Suggestions for future work are provided later in this chapter.

This report described the project assignment and the identified requirements. Then, the scientific and agile approaches were discussed, and the tools used were introduced. Next, the design and implementation of the global system, consisting of a Client and Server application were explained and illustrated. During the development process particular attention was paid to ensure that the Client components are maintainable and that the Server is maintainable and extendable with additional reasoning modules. This showed from the fact that we achieved a maximum score on maintainability from the Software Improvement Group. Several reasoning components were implemented as a proof of concept and additional reasoning rules can be easily added to the system by implementing the corresponding interfaces. Finally, different tests were performed and the testing results were discussed. Using acceptance testing it was assured that the important ‘must have’ and ‘should have’ requirements have been met.

The original global goals of the project were stated in Chapter 2. Each of these goals have been met. For each of these goals we will now discuss how they have been achieved during the project.

- **The system must be capable of capturing data from multiple cameras, and then detect, identify and locate moving objects.**
  For this sole purpose a separate Client application, written in C++ with the OpenCV library, has been developed. This application runs through an image processing pipeline. A camera is directly attached to the hardware that is running this application. Next, the moving objects are detected in real-time using a Mixture of Gaussians algorithm. By applying a Histogram of Oriented Gradients classifier, the moving objects are identified as either human or non-human. Finally, by using a custom geometrical localization algorithm, the locations of the moving objects are determined. To allow the System to capture data from multiple cameras simultaneously a Client/Server model is introduced. When a Client application is booted up it identifies itself with the server to get a unique label. It is then allowed to transmit the computed features of the detected moving objects to a central Server application in real-time.
The received data must be processed into different objects which may then be tracked by the system. The path history of GPS coordinates is then used to extract features for further reasoning.

The Server application cycles through the received information from each of its connected Client applications. For each of the detected objects it determines an accurate estimate of the GPS location. It then tries to match this detected object to an actual object of interest within the system, based on its GPS location. For each of the objects of interest it applies location matcher rules, to see if the new GPS location is a match to the existing object. Such a rule could for instance predict the next location of an object based on its average direction and speed. When it is statistically plausible that the new location is the new location of this object it is considered a match. When no match is found a new object of interest is created. Such matcher rules can easily be added to the Server by implementing an interface. Moreover, the Server ensures that objects that are considered irrelevant are removed from the Server application. A current rule for this discards objects that have not been seen for a time that is 1.5 times longer than the time it has been seen in total.

Given a certain context or region of interest in combination with extracted features the system must determine whether behavior is considered suspicious. Suspicious behavior must result in an alert.

This goal has currently been achieved by implementing the system as a rule based Expert System. The Server application allows for definition of Areas of Interest. Such areas may be defined on the map of the area which is being monitored, and whenever an object is within the area a certain rule is applied. This rule may result in an Alarm being fired. An example of such a rule is the rule for a SpeedLimit: whenever an object is within this area and its average speed over a certain timeframe exceeds the speed limit, it fires an alarm. New types of areas may easily be added to the system by extending the abstract AreaofInterest class. Besides Areas of Interest the System allows for defining Rules and Comparative Rules. Such rules evaluate a single object, or compare each pair of objects respectively. The basic rule reasons about a single Object of Interest and its extracted features, an example of such a rule determines when an object starts running to then fire an alert. A comparative rule compares the computed features of two Objects of Interest. For example, this could be used to determine if one object is following another, based on its history of GPS locations. Like the object matchers, Rules and Comparative Rules may be easily added to the Server application by implementing an interface.

The server in the control room should visualize the gathered data and results of the reasoning process to the security personnel through a graphical user interface.

Through a user centered design process a GUI has been developed that will visualize the gathered data and results of the reasoning process (alarms) to the security personnel. An important component of this GUI is the Map Panel, which shows the up-to-date locations of all objects in the area on a map. Each unique object is marked using a unique color throughout the GUI. The Map Panel allows zooming and dragging to survey the complete area. Besides the MapPanel the GUI contains a Console in which informative messages on the status of the connected Clients, and the Server system itself are displayed. Additionally, the GUI contains a plot in which the average speed of each of the objects is displayed over time. Again each object is labeled using its unique color. Finally, the GUI contains a table. This table is the crucial component of the GUI, as it displays the alarms that have been fired. Each alarm is colored according to its priority, and contains information on the time, type of event and source object of a suspicious event. Moreover, it displays the camera that last detected the suspicious event, such that the corresponding video may easily be retrieved.
Recommendations for Future Work

Although all the ‘must have’ and ‘should have’ requirements that have been set at the beginning of the project have been met, there are still features that could be implemented to further improve the System. In this section we will discuss ideas for such features and improvements that could be carried out in future work.

- Due to use of Scrum the features were implemented in an incremental way, ordered by their priorities. Before implementing any reasoning modules, the rest of the Server components, such as the Whiteboard, had to be in place. Additionally a Debug Server application had to be developed to allow for testing of these reasoning modules. Because the reasoning component is well extendable with new reasoning modules, the system could be made more intelligent by adding more rules and areas to the Expert System.

- More complex classifiers for the Client application would allow for more and more complex reasoning modules to be added. When the Client is capable of detecting actions such as waving or lying down the Server has more features to reason about, allowing it to detect more distinct suspicious situations. Additionally a classifier could distinguish different types or persons, such as guards and pedestrians. Again this would allow for more complex and specific reasoning rules to be added to the Server. This would for instance allow for areas to distinguish between different types of persons. Secured areas could allow guards to patroll the area, while a pedestrian entering the area would fire an alarm.

- Currently, some parameters for the Mixture of Gaussians (MoG) background subtraction are selected manually. This technique is more robust when these parameters are optimized automatically. In order to optimize this, a sequence of images with a ground truth foreground is required however. The background subtraction may then be applied for a range of parameters, and the combination that results in the smallest overall difference with the ground truth can be considered the best set of parameters.

- A useful improvement through future work would be improving the Debug Server application. This currently serves as an unofficial extension of the Server, and is used mainly for testing purposes. Its tools that allow simple drawing of Areas of Interest could be useful functionality in the main application too. It could even feature an intuitive, graphical method to define new reasoner rules.

- As shown by the performance test, the Server application currently scales up to 20 cameras under heavy reasoning load. This is insufficient when monitoring extremely large areas. To solve this the Server application should be extended to function in a distributed setting, separating the monitored area into sub areas. Each distributed application then reasons for its own sub area, but relies on constant communication with the other distributed applications. This could be implemented using an Agent framework such as JADE or JASON.

- One of the original problems is that the security camera monitors are unattended when the security personnel is patrolling the area in person. To allow the security personnel to still be up to date on the complete information on the monitored area, an mobile application could be developed. The security personnel could then be provided an overview of the area, and alerts whenever something suspicious occurs. The personnel may then determine based on this information whether the situation is worth checking out in person.

- A suggestion that resulted from the user acceptance test was that alarms could be accompanied by an alarm sound. This would ensure that the security personnel would not need to constantly monitor the Server GUI as well, but only check occasionally or when an alarm notice is played.
Currently the camera images are not incorporated into the Server application. It would be a very useful improvement to allow the security personnel to request from a camera to start streaming its live video whenever a suspicious situation occurs. The security personnel could then instantly observe the situation to provide a better judgement before determining whether to call for backup. It would be even better to allow the Server application to request not only the live footage, but any timeframe from the history of stored video, so that past events may also be replayed easily without having to search for the correct video footage manually.

Because the system is currently modeled as an Expert System it suffers from some of the disadvantages of Expert Systems. The system can not adapt without changing the rules. A situation is either suspicious, or not. There is no grey area in between. Moreover, the rules in the Expert System could contain errors that result in incorrect decisions. This situation can be improved by replacing the Expert System by a form of probabilistic reasoning, such as a Bayesian Network or a Fuzzy Expert System.

Our final recommendation involves automated testing of the code. Currently not all classes in the Client and Server applications allow for Unit testing. The classes in the Reasoner package are currently tested by using manually generated simulation files, and manually assessing the outcome of these simulations. Playing back these simulations currently happens in real-time. By improving the application to allow for fast playback of these simulations, these scenario based tests could be automated. This could be done by automatically evaluating whether an alarm was fired at the expected movement in the simulation.

A second automated testing suggestion regards the GUI. Though the MapPanel is tested automatically as it allows for user interaction, the rest of the GUI is not yet automatically tested. Though testing a GUI is more difficult than testing classes with simple in- and output behavior, these classes could still be tested.


[34] L. Van der Maaten, E. Postma, and J. van den Herik. Dimensionality reduction: A comparative

and approximation methods for the real time object detection. In Asian Journal of Computer
Science and Information Technology, volume 1, 2011.

[36] C. Veness. Calculate distance, bearing and more between latitude/longitude points. Re-
trieved July 10, 2012, movable-type.co.uk/scripts/latlong.html.

-- 1518, dec. 2003.


Computational Intelligence and Software Engineering, 2009 CiSE International Conference
on, pages 1--4, 2009.

[40] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for
Appendix A - Project Assignment

The assignment is stated on the next two pages. Please note that this is the original project assignment. The actual performed assignment has changed from this original assignment. The description given in the final report is the description that matches the actual assignment.
BSc project: Protection of the Safety Guard using surveillance cameras

Location: Netherlands Defence Academy at Den Helder

Period: 10 weeks in the 4th quarter (May-July 2012)

Supervisor: Prof. drs. dr. L.J.M. Rothkrantz (TUD, NLDA), ir I. Lefter

Problem: The area of the NLDA at Den Helder is an area surrounded by a fence and water and supervised by a camera system. The main goal of the security system is to protect the area against intruders and to detect unwanted behavior of persons in the area. One of the priorities is to protect NLDA-employees (i.e. safety guard)

At this moment the recordings of the cameras are supervised by human operators. There is a need for automatic supervision. At regular time the safety guard inspects the area. The camera system can also be used to protect the safety guard during his inspection. In case the guard will be attacked by intruders or in the case he is attacked by a suddenly illness, this should be noticed by the surveillance system. In all those case the guard shows typical behavior which should be detected, classified and interpreted by the pattern recognition software implemented in the software of the surveillance system.

The proposed Automatic Surveillance System should be able to detect the following events:
- safety guard walks in a strange way, not along the common straight line along the street (wounded, illness) or takes an unusual path (disorientation)
- safety guard is running, possible chased
- safety guard falls down or seeks protection
- safety guards waves his hands (call for help)
- unusual interaction visitors-safety-guards, fight
- safety guard is tracked by unknown persons(s).

Model: The most dangerous hours are the silent hours when almost nobody is present or walking around at the NLDA area. In those cases it is possible to use to use “tracking-software”. The tracked path has to be analyzed. Some features as location, speed and curvature can be extracted or computed. Based on those features it is possible to take some conclusions. In case of interaction, or no movement at all, the amount of
movement can be computed from one frame to the other, and again some conclusions can be taken. Because there are more camera’s available, with no or partial overlapping vision fields it is important that camera’s can communicate with other. Every camera can be modeled as an agent, which is a computing device with the following characteristics, autonomous, able to perceive the environment, able to extract features from the context and able to reason about the observed features and able to take some actions. Software for multiple agents system (MAS) is available for free (JADE). The system can be modeled as a centralized or decentralized system or as a hybrid system. In principle videorecordings from the surveillance system can be analyzed. But with respect to the limitation in space and time, it is expected that as a first approach the proposed system will run in a simulated environments as a proof of concept.

Methodology:
- Literature survey surveillance systems, recognition of non verbal behavior, gestures, tracking systems, path analysis
- Video-recordings of behavior of safety guards using actors, annotation of data
- Data analysis
- Write reports with problem definition, planning proposed methodology, progress report, final report.

Deliverables:
- Reports with a description of the design, implementation and test results
- Model, architecture, design
- Prototype of a system (running on a PC), which is able to analyze video recordings and detecting unusual behavior of safety guards as specified.

Conditions: It is possible to do the project (part time) on the NLDA location as “stagiair”.

February, 2012, L.J.M. Rothkrantz
Appendix B - Orientation Report

Introduction

For this Bachelor of Science project a software product is being developed that can monitor the pedestrians in the Netherlands Defence Academy (NLDA) area of the ‘Koninklijk Instituut voor de Marine’ (KIM). The goal of the project is to develop a system capable of assisting and automatically alarming the personnel in the control room when required. This may for example be the case when the security guard is walking in a strange way, which might indicate that he is drunk or wounded, or when a person is waving his hands to call for help.

This report documents the findings of the orientation phase of the project. The scope and domain of the application are defined in section 2 followed by a discussion of the global system design and its major components in Section 3. The document is concluded with a fourth section describing the tools and languages that will be used during the project.

Application, Scope and Domain

In this section the domain of the application will be described. Next the scope of the system is explained in order to provide an overview of what is to be expected of the system.

The area of the NLDA(Netherlands Defense Academy) at Den Helder is a military area surrounded by a fence and water. Currently the site is supervised by a camera system and monitored by security personnel in a control room. The main problem with the current system lies in the nature of the tasks of the personnel. They have to passively watch multiple monitors simultaneously. Humans easily get tired and lose concentration, especially when no events of interest occur for long periods of time. Furthermore humans do not notice every small detail or let alone focus on multiple situations simultaneously. A second problem is that in some areas the camera footage is recorded without supervision of any personnel for financial reasons. The video footage is only stored in case something would happen. This prevents incidents from being detected and situations being controlled in a timely manner.

Maarten Somhorst [31] made a first attempt at developing a surveillance system model designed to detect suspicious behavior in a non-public area. This system consisted of different components but was not fully functional and only vehicles were considered. The system described in this report will partly build on the ideas described by Maarten Somhorst in an attempt to further refine the concept.

In Figure B.1 the current situation as well as the desired expansion of the security monitoring system is displayed. There are $n$ cameras which stream their captured video images in real-time to the monitors in the control room so the personnel can overview the terrain and look for any suspicious events. This part of the infrastructure already exists and is also currently deployed. The goal of the project is to develop the content inside the orange ‘Our software product’ box,
an intelligent reasoning system. The system that we will develop will provide assistance to these human operators to counter the problems with human operators described previously.

The system will display an overview of the area that is being monitored. The first priority is to display the locations and paths of all objects that are in view of the cameras. Next basic parameters can be derived from this data such as the speed and curvature of these objects. These parameters are also be presented to the human operators in the overview. Additionally areas of interest may be provided to describe extra domain specific information. This allows the system to derive when unauthorized persons enter a forbidden area. A next step can be to model the common pedestrian walking routes for the area. Whenever a person follows an uncommon path the system can track this persons moves extra closely in attempt to detect undesired activities. A final step may be to identify the postures of humans in the camera images such as lying down, sitting, running and interaction with other humans such as fighting or a discussion. This information may prove to be valuable extra information for the system to reason about situations.

Global System Design

Overview

This Section describes the global system design and its main components. The scope of the system was discussed in the previous section. As can be derived from Figure B.1 the software product will consist of three major components: the client, the server and the graphical user interface (GUI) in the control room.

Figure B.2 illustrates the main data flow within the system from capturing a camera frame to visualizing the scene and alarming the personnel in the control room. Each process component in this flowchart has one out of three priority levels: low, medium or high. Low priority components will initially not be focussed on. They will only be implemented when extra time is available at the end of the project. For the medium priority components only the basic functionality will be implemented. The high priority components will have the main focus during design and implementation.
In the next three subsections the essence of the client, server and graphical user interface will separately be described, in correspondence with the flowchart of Figure B.2.

The server will combine all the gathered data and try to build a bigger picture of the overall situation in order to determine if anything suspicious is happening. It will use the idea of a whiteboard architectural pattern to model a global knowledge base containing all the objects data received from the clients, which will have to of course continuously be updated. A context reasoner component in the server will be responsible for deducing any alarming events from the data on the whiteboard.

A separate GUI in the control room will visualize the locations of all the cameras and detected persons on one big map, together with some extracted object parameters like speed, curvature, etc. Furthermore the GUI will be responsible for displaying a message in case of an alerting event.
**Client**

The idea is to associate every camera with a corresponding client component. This component will be responsible for processing the sequence of images supplied by the camera, and for detecting and locating the patrolling guard and other humans in the scene. Once this object information is extracted the clients will send it to a central server component. The client component will thus focus on the image processing part of the system. It will however not be like any commonly seen ordinary object oriented program. It will for example have no direct interaction with a user, but rather will serve as a purely functional processing unit: for each frame it will run through a cycle of consecutive processes. The client can therefore be considered as a **processing pipeline** (see top part of Figure B.2)

Each of the processes in this pipeline can be considered as a separate module, representing one subfunctionality in the pipeline process such as foreground detection and object classification. Some modules perform advanced computer vision operations and many different algorithms have already been developed by others that could help with implementing these modules. The client should be designed such that each of the components may be replaced by a new implementation to improve maintainability. In Appendix C a literature survey on some possible techniques is presented for every module. In the appendix we also discuss the suitability of the different techniques for this project and we give an argumentation of which technique we think will work best.

**Server**

This Section described the main components of the server software which will contain the reasoning part of the system and provide the situational overview to the human operators in the control room. In the second half of Figure B.2 the components of the server are displayed as part of the complete system. In Figure B.3 the components and their relationships are displayed separately.

---

Figure B.3: server software components and relationships overview
The **Client message handler** will be responsible for handling the connections with all the camera clients. When a new camera wants to connect to the server, an initialization procedure of this client message handler will make sure the new connection is made properly and the camera’s static parameters (like height, viewing angle and the two field of views) are also saved. The clients will continuously send new image information to the server, and so the main job of the client message handler will be to regulate this stream of incoming information, extract the object information (i.e. location and type) from the received packages and update the whiteboard component by supplying it this new data.

The **Whiteboard** will be a central part of the server, as can be seen by looking at Figure B.3. It will be based on the ‘Blackboard architectural pattern’ [14]. It will act as a global shared knowledge base containing all the object information received from the clients. Its role in the system will be highly passive in nature. It won’t take much initiative itself, but rather will listen to the messages and queries received from other components, and try to adequately respond to them. These queries include update (1) messages received from the client message handler with new data to be added to the whiteboard, (2) queries received from the context reasoner to read data of the whiteboard, and (3) update messages received from the context reasoner to change or delete data from the whiteboard. There is one part in which the whiteboard will play a bit more of an active role, namely providing object information to the view component in order for latest information to be visualized on the monitor in the control room.

The **Context Reasoner** will be the brain of the server. Its main function will be providing high-level reasoning over the data on the whiteboard. It will analyze the movements of the objects by calculating among others the speed and curvature, using a series of input locations. If an object is a patrolling guard then the deviation from the standard route will also be considered as a parameter. Furthermore the context reasoner will have access to so-called region of interest (ROI). These are areas on the terrain to which a particular function/meaning can be associated (e.g. a road section, a building,...). This ROI’s will enable the context reasoner to determine whether or not objects are moving where they belong. Because the context reasoner has the best view on the overall situation it will also be given the task of deleting irrelevant information from the whiteboard.

The **View and Controller** components together compose the user interface and are described in the following Section.

### Graphical User Interface

In Figure B.4 an initial mock-up of the Server Graphical User Interface that will provide an overview of the security situation of the NLDA area is shown.

The first GUI component is the Map. This map is a schematic 2D layout of the area that is monitored. On this map the current location and path of all objects within camera view are displayed. Each object will have its own unique color that will serve as its identifier within the overview. The current location of the object is indicated by a circle and the history of its locations or path is shown as a line. The different cameras that are situated in the area are displayed on the map as well.

The second GUI component contains graphs that visualizes over time the basic object features that were derived from the objects location. These features include among others speed, curvature and deviation of the standard path.

The third GUI component is a table which lists all detected objects that are currently in the area. In addition to the information displayed in the graphs, the table provides a complete overview of all features extracted on the different objects. In the table each object will have an indication of
its suspiciousness showed as a color from green to red for respectively low to high suspiciousness.

The fourth component is an optional debug console. This can be used to report technical issues such as cameras that have lost connection with the reasoning server.

When the system derives from the situation that an alarming event has occurred, the system will fire an alarm by showing a pop-up window and a sound effect to alert the human operators in the control room.

Project Tools and Software

In this Section the programming languages, libraries and tools used during the project are described and briefly explained. Table B.1 gives a list of references to the websites of all the libraries and tools mentioned in this Chapter.

Programming Languages and Libraries

Due to the extensive need for image processing on the camera side software we have chosen to use the C++ programming language [33] in conjunction with the OpenCV library. C++ provides fast operational speed which is important because the captured camera images need be processed at runtime. Importantly, C++ allows for a reasonably high level programming environment.

The field of computer vision is very broad and many algorithms have already been developed to solve the common problems. The OpenCV library for C++ offers an extensive array of basic functionalities, consisting of over 2500 different optimized algorithms. Furthermore these functionalities have been tested and are reasonably well documented. The version that will be used during this project is version 2.4.0, which is the latest stable version.

The cameras act as clients which send their extracted object information to a central server. This server provides a high level of reasoning over the data and tries to find the bigger picture of the
current situation. The image data the server receives from all its clients has been stripped to its essentials, so there is no great data overhead on the server side. Furthermore the server will not perform any computationally intensive calculations. Therefore we have chosen to use the Java programming language [7] to program the server. Java offers an easy and clear syntax and enforces proper programming as opposed to the older C++ language. Java also allows for simpler testing and easier GUI design and implementation. A final advantage is that Java runs on every platform, allowing the server software to run on any machine.

Table B.1: references to the websites of the mentioned libraries and tools

<table>
<thead>
<tr>
<th>Libraries and tools references</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Development Tools (CDT)</td>
<td>eclipse.org/ctd</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>maven.apache.org/plugins/</td>
</tr>
<tr>
<td></td>
<td>maven-checkstyle-plugin</td>
</tr>
<tr>
<td>Cloudbees</td>
<td>cloudbees.com</td>
</tr>
<tr>
<td>Cobertura</td>
<td>cobertura.sourceforge.net</td>
</tr>
<tr>
<td>Doxygen</td>
<td>home.gna.org/eclox</td>
</tr>
<tr>
<td>Eclipse IDE</td>
<td>eclipse.org</td>
</tr>
<tr>
<td>Google Test framework</td>
<td>code.google.com/p/googletest</td>
</tr>
<tr>
<td>Jenkins</td>
<td>jenkins-ci.org</td>
</tr>
<tr>
<td>JUnit</td>
<td>junit.org</td>
</tr>
<tr>
<td>Maven</td>
<td>maven.apache.org</td>
</tr>
<tr>
<td>OpenCV</td>
<td>opencv.willowgarage.com/wiki</td>
</tr>
<tr>
<td>Scrum</td>
<td>scrumdo.com</td>
</tr>
<tr>
<td>Visual Studio</td>
<td>microsoft.com/visualstudio</td>
</tr>
</tbody>
</table>

**Development Tools**

A downside of C++ is that the language is less straightforward when it comes to project management as opposed to newer higher level programming languages. The need for an IDE for project management is obvious. Although Visual Studio allows for simplified C++ development in Windows, the software will be designed under Linux. The C Development Tools (CDT) plugin for the Eclipse IDE provides tools for developing C++ applications on any platform using Eclipse, which naturally supports Java development as well. Eclipse makes development easier through project and package management, debugging tools and style checking.

Within Eclipse multiple different plugins allow better development of the system. For this project the Doxygen plugin that comes with the CDT plugin allows for Javadoc-like documentation of the C++ code, and automated generation of API's.

Although Computer Vision software is often more suitable for manual testing than automated testing some classes and methods will still require automated testing. For the C++ part the Google Test framework allows unit testing. The Java project will be built as a Maven project. Maven ensures proper standardized structure of the program to enforce maintainability. For testing the Java based server software the JUnit testing suite will be used in combination with the Cobertura plugin to measure test coverage, and the Checkstyle plugin to ensure the written code meets programming style standards. Finally the Java code will be documented using Javadoc. All these aspects will be integrated with Maven so that proper distributions of the program and documentation can easily be generated. Finally we will use the online Jenkins service provided by Cloudbees.
Jenkins provides continuous integration testing of the server software. Additionally it will provide information on the test coverage and amount of checkstyle errors generated in the different builds.

**Process Tools**

The development process must also be monitored. As described in the Plan of Action we will make use of the agile programming method Scrum. To supervise this Scrum process the free online Scrum tool is used.
Appendix C - Literature Study Client Modules

For every image processing subproblem described in the pipeline process of the client software, this appendix discusses the algorithms that were surveyed during the literature phase of the project. It was originally an appendix of the Orientation Report to keep it readable.

Foreground Detection Module

Background Subtraction techniques or Foreground Detection techniques are techniques that attempt to classify pixels as either part of the static background of the picture or as part of a moving/foreground object. A background subtraction technique requires a typical RGB camera frame as input. The pixels belonging to the foreground objects are stored in a binary image or mask which is returned as result. This process is an important initial step in the pipeline process of the camera client. Detection of foreground objects is crucial as it provides a basis for the consecutive computer vision operations, such as localization and classification. Inaccurate or incomplete segmentation of foreground objects can lead to large errors in the consequent processes.

The security cameras at the military domain are positioned outdoors. This introduces some important background segmentation problems. Firstly, the brightness of the image is subject to change due to the weather conditions. Secondly, background objects such as trees might not be completely static due to the effects of wind. Thirdly, objects that are moved in and out of the background scene, such as a car being parked, must be incorporated into the background image when the object remains static long enough. Finally, sunlight causes foreground objects to cast shadows onto the image. These shadows should preferably not be included in the foreground images. Each of these problems must be taken into account when selecting a suitable background subtraction technique.

Static Background Subtraction

We have been prototyping different background subtraction techniques. Initially, we tried the most basic static background subtraction technique where a single background frame is maintained which is subtracted from the incoming new frame. Subsequently, we thresholded the resulting image at a predefined value, the sensitivity to change, to find the foreground. This technique however proved to be too sensitive to illumination changes and moving background objects such as trees.

Weighted Moving Variance Background Subtraction

The next prototype was the weighted moving variance approach. This technique estimates a background model by computing the mean and variance using the last \( n \) frames, with a weight attached to each frame and the sum of the weights summing up to one. The new incoming frame is then compared with this model and a pixel is considered foreground when it differs more than 2.5 standard deviations from its mean in either of the RGB color spaces. This technique adapted the
background model too fast, causing objects that stay static for a short period to be incorporated into the background. This made it unsuitable for this system.

**Chromaticity and Gradient Based Background Subtraction**

The third background subtraction technique we evaluated was the chromaticity and gradient background subtraction, which was implemented by combining the approaches of McKenna, Jabri, Wechsler and Rosenfield [24] as well as the approach of Yuan and Yang [39]. This technique converts the RGB images to YUV space which consists of two chromaticity components and a brightness component. The chromaticity components are relatively insensitive to changes in illumination. For each new frame the models for the chromaticity and brightness values are recursively updated using the following equations

$$\mu_{t+1} = \alpha \mu_t + (1 - \alpha)z_{t+1}$$

$$\sigma^2_{t+1} = \alpha (\sigma^2_t + (\mu_{t+1} - \mu_t)^2) + (1 - \alpha)(z_{t+1} - \mu_{t+1})^2$$

where $\alpha$ is the adaption rate that may be chosen freely. The difference between the chromaticity and brightness values of the new frame and the mean of the model are again computed and tresholded for a value of 2.5 standard deviations to find foreground pixels. When the chromaticity does not differ much, but the brightness does, the pixel is considered a shadow pixel. Next, the edges in the brightness component are detected and the largest component is filled. The resulting mask is then combined with the mask found using tresholding to find the final foreground mask.

This technique proved to be not very effective in detecting multiple persons, or persons that stand further away, mainly due to the gradient component of the technique.

**Mixture of Gaussians**

The final technique we considered uses a mixture of Gaussians and is described by Stauffer and Grimson [32]. The technique allows a pixel to be part of multiple background surfaces, under different lighting conditions.

Each pixel from the incoming new frame is matched with its existing $K$ Gaussian distributions. Using the Mahalanobis distance all distributions that are less than 3 away will be matched to the pixel. The distribution with the highest weight is then selected. When no match is found the distribution with the lowest weight is replaced with a new distribution taking the pixels current value as a mean and an initially high variance. This allows new objects or new lighting conditions to be incorporated into the background model. After evaluating the matches the distributions are updated as follows:

$$w_{k,t} = (1 - \alpha)w_{k,t-1} + \alpha(M_{k,t})$$

Here $\alpha$ is the adaption rate that we also saw in the chromaticity/gradient based approach and $M_{k,t}$ is 1 for the closest Gaussian distribution, and 0 for the other Gaussian distributions. For every unmatched distributions the parameters $\mu$ and $\sigma$ remain unchanged. For distributions that matched the pixel these values are updated using the following equations:

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t$$

$$\sigma_t^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)(X_t - \mu_t)^T$$

with $\rho = \alpha N(X_t|\mu_k, \sigma_k)$. To determine the combination of distributions that model the background
the Gaussians are ordered by their weight/variance ratio. This is a value that increases when the
distribution often matches the pixel value, and when the variance decreases. Both indicate that
the distribution models the background. Next the first $B$ distributions are selected to model the
background model:

$$B = \arg \min_b \left( \sum_{i=1}^{K} w_i > \tau \right)$$  \hspace{1cm} (C.6)

The fraction of the entire image that should be accounted for by the background models in $B$ is
$\tau$. This value may be chosen freely. A small value for $\tau$ causes the background to be modeled
by only a small amount of distributions, resulting in computational advantages when matching a
pixel against its distributions. Larger values for $\tau$ allows a larger set of distributions, which is com-
putationally more intensive, but can result in more than one color being included in the background
model. This makes the background model more robust in the case of background motion, such
as a tree in the wind.

The described process above allows detection of background pixels as well as updating the back-
ground model. When a pixel is matched with a distribution in $B$ it is considered as belonging to the
background. The method itself proved to be effective at adapting to anything but quick changes
in weather conditions. It succesfully adapts to new objects entering or objects leaving the scene,
as well as changes in brightness and day/night cycles [32]. The improved version from Zivkovic
and van der Heijden [40] automatically selects the optimal number of distributions $K$ per pixel and
provides some performance optimalizations. We have chosen this technique as it performed best
under the conditions of this project.

**Image Enhancing Module**

The image resulting from foreground detection is not always perfect. The picture may contain
holes or an object may be separated in two parts. Additionally the camera generates noise that
is sometimes detected as part of the foreground. To improve the quality of the foreground mask
several techniques can be used to enhance the image.

OpenCV offers some basic operations to do this. By using the erosion operation the noise pixels
that are detected as foreground are removed from the mask. This may also remove some of the
shadow pixels that got incorporated into the foreground. When the dilation operation follows the
erosion this allows the original larger objects to be restored to their initial size, and objects that got
split up to grow back together. Finally the remaining objects may be filled to close any holes that
resulted from the background subtraction.

**Object Labeling and Filtering**

For detecting different objects in the foreground image and labeling these objects some basic
image processing operations are required. By using a connected component finding algorithm
on the foreground mask and labeling the different connected components this can be done. The
OpenCV CVBlob library [2] provides the functionality that is required to do this and some extra
functionality. It allows filtering the found objects based on a minimum size, as well as tracking
a labeled object over multiple consecutive frames. The bounding boxes of the different labeled
objects can then be aquired from the list of detected objects for further processing such as clas-
sification.
Classification Techniques

Once the foreground regions of the moving objects are detected and transformed into binary connected regions (blobs), the next step is to classify these blobs as either human or non-human, and if human to also determine the pose that is being adopted (standing, sitting or lying). This Section describes a few classification techniques, that were surveyed during the literature study, that can be used to achieve this goal. Note that the list of proposed techniques is far from exhaustive. Due to the page limitation only a global description of the techniques could be given. So with simplicity and clarity in mind, all smaller details have been omitted.

Every technique assumes a cropped binary image of the object as its input and returns an identifier indicating the computed type of the object (i.e. human-standing, human-sitting, human-lying or non-human), see Figure F.2 for an example.

The techniques can broadly be divided in two categories: feature-based and pixel-based classification. Feature-based techniques first calculate one or more feature descriptors out of the given input image and subsequently give these as input to a specific classifier which returns the type of the object. Pixel-based techniques don’t calculate any features in advance but rather base their classification directly on the pixels of the input image.

This section discusses two feature-based classification techniques: a silhouette-based and a shape-based neural network classifier. Additionally a pixel-based neural network will also shortly be discussed.

Silhouette-based

The idea of this technique is to classify the binary blob image by looking at its contour and comparing this to the contours of previously collected images of which the human poses are already known. This technique is described in more detail in [11] and is based on a so called silhouette (or contour) descriptor. This descriptor characterizes the distance of the border points, of the blob in the binary image, to its centroid. A simple distance measure is used to compare the descriptor to a pre-established dataset of human contour descriptors in order to determine the type of the given object.

First the centroid $C$ of the binary input image $f$ is calculated. Next, the contour of the object region is extracted from the binary image using some simple algorithm, like setting all pixels whose 4-connected neighbors are all 1 to 0. An example is giving in Figure F.3.
Now a so called distance signal $D = \{d_1, ..., d_n\}$ can be generated containing for each boundary point $p_i$ the Euclidean distance $d_i$ between this point and the centroid $C$ (see Figure C.3). In order to be able to compare signals belonging to different sized object images, the length of each signal is reduced to some predefined length, by over- or undersampling.

Next the acquired signal is normalized, resulting in the final signal $\hat{D}_N$. This signal is the descriptor that will be used to compare the object with the predefined database. The comparison makes use of a pairwise distance formula to measure the similarity between two descriptors.

Suppose $P$ is a new object and let $Q$ denote an object from the database. To determine the type of object $P$, its descriptor can now be compared to all object descriptors in the database. The type label of object $Q$ is assigned to object $P$ iff their is no other object in the database whose descriptor distance to $P$ is smaller than that to $Q$, and the descriptor distance is below some predefined threshold $T$.

**Shape-based Neural Network**

This technique first calculates a list of shape parameters which are then supplied as input to a pre-trained neural network which has an output neuron for every object type that can be distinguished. Table F.1 gives a list of shape parameters that can be distinguished, together with their formula or description.

<table>
<thead>
<tr>
<th>Shape parameters</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect ratio</td>
<td>$\frac{\text{width}}{\text{height}}$</td>
</tr>
<tr>
<td>Formfactor</td>
<td>$\frac{4\pi \times \text{area}}{\text{perimeter}^2}$</td>
</tr>
<tr>
<td>Roundness</td>
<td>$\frac{4 \times \text{area}}{\pi \times \text{maxdiameter}^2}$</td>
</tr>
<tr>
<td>Orientation</td>
<td>the &quot;angle between the $x$-axis and the major axis of the eclipse that has the same second-order moments as the region&quot; [3]</td>
</tr>
<tr>
<td>Relative areas</td>
<td>for every image quadrant the proportion of pixels equalling 1 against the total area of the binary image is calculated, giving</td>
</tr>
</tbody>
</table>

Figure C.2: Boundary extraction  
Figure C.3: centroid contour distance
object type, so the expected values of the neurons in the output layers are known (e.g. input parameters corresponding to a "human-standing" image should result in an output of (1,0,0,0) in Figure F.5). If the output-values don’t match with the expected output, then the weights of the network will need to be adjusted to correct for the error that was made. A possible technique that can be used is the supervised back-propagation algorithm.

One of the advantages of a neural network is that in general the time required to train a neural network is much less than the time needed to set up an rule-based (i.e. expert) system to solve the same problem. The main drawback however is the black-box characteristic of the network. It’s very hard to tell how good the network will perform and to be sure the network is trained with a comprehensive and fully representative dataset.

![Global overview of the shape-based neural network](image)

**Figure C.4: Global overview of the shape-based neural network**

**Pixel-based Neural Network**

The technique is very similar to the previous shape-based neural network. The only difference lies in the input layer of the network. The network doesn’t require any shape-parameters as its input. Instead every input neuron corresponds with a particular pixel value of the binary input image.

The advantage of this technique over the previous one is that more information of the image can be captured: where the previous network only had the ability to reason about a fixed number of image properties, this network takes into account every pixel of the input image. This makes the technique more flexible. There are, however, two drawbacks concerning this technique. First of all, due to the enormous number of input variables the network will need much more training data compared to the previous network in order to be robust and have a good performance. Secondly, the network has a fixed number of input values meaning that every input image will first have to be rescaled to an image of some predefined width and height.

**Histogram Oriented Gradient Features for Classification**

Histogram oriented gradients[9] are features that can be used for classification. Often a support vector machine is used for the classification part. The algorithm works by computing the gradient of a window in the image and subdividing this into cells. For each cell gradient direction histograms are computed and combined to become the descriptor of that cell. Because lighting conditions
can cause large differences in contrast, changes in cell descriptors can occur. For this reason, cells are contrast-normalized by measuring intensity of an larger area of the image (including the cell itself). After classification often a support vector machine is trained using pictures of the object to be classified (positive examples) together with pictures of the scene without the object in view (negative examples). Because the size of the feature vector produced by the histogram oriented gradient algorithm is constant and based on the size of the original input window, a sliding window and multi-dimensional scaling is used to detect the object in question within a given frame.

In the end we settled for the histogram orient feature extraction technique in combination with a support vector machine. The reason for choosing the combination of the histogram oriented gradients feature extraction in combination with a support vector machine stems from the results published in the original HOG paper[9], that show that the combination produces very few false positives and robustly classifies large well known testing sets.
Appendix D - Localization Algorithm

This Section focusses on the problem of determining the 3D world coordinates of a point in a given camera image (i.e. object localization). In order to be able to mathematically solve this problem it is assumed that the camera is fixed in space at a particular height and has a fixed field of view. Furthermore the assumption is made that the camera is vertically directed under an angle that lets it fully face the ground, so no horizon is visible in the image. In the context of camera surveillance systems these are all defendable and plausible assumptions that do not pose great limitations to the employability of the system.

More formally the following input values are required: height of the camera to the ground ($h$, in meters), vertical camera angle ($\alpha$, in degrees), the camera’s vertical field of view (FOVx, in degrees), the camera’s horizontal field of view (FOVx, in degrees). See Figure D.1 to get a visual interpretation of these values.

![Figure D.1: camera view and required parameters](image)

Cross Ratio Theorem

Before proceeding with the discussion a little trip to the Cross Ratio Theorem is required. The Cross Ratio [21] is a useful geometry concept, and is defined as follows:

**Definition 1** Let $A,B,C$ and $D$ be four points lying on a line (i.e. collinear points). The Cross Ratio
The cross ratio $\mathcal{R}$ of the points $A, B$ and $C, D$ is then defined as follows:

$$\mathcal{R}(A, B; C, D) = \frac{|AC|}{|AD|} \cdot \frac{|BC|}{|BD|}$$

(D.1)

(Where $|XY|$ indicates the euclidean distance between $X$ and $Y$)

**Definition 2** Let a line $l_1$ be given containing four points $A, B, C$ and $D$. Furthermore, let a point $O$ and a line $l_2$ be given. Define the points $A', B', C'$ and $D'$ as follows. $A'$ is the intersection of $l_2$ and the line going through $A$ and $O$. $B'$ is the intersection of $l_2$ and the line going through $B$ and $O$. $C'$ and $D'$ are defined similarly (see Figure D.2 for an illustration). The Cross Ratio Theorem now states that

$$\mathcal{R}(A, B; C, D) = \mathcal{R}(A', B'; C', D')$$

(D.2)

(i.e. the Cross Ratio is **projectively invariant**).

---

**Determining the z-coordinate**

Figure D.3 is an abstraction of Figure D.1 and will serve as a guide for the mathematical derivation of the solution to the localization problem. Every point $X$ on the ground has a corresponding point in the camera image, indicated by $X_i$. The point $E_i$ is the center of the camera image: the optical axis of the camera goes through $E_i$ and intersects the ground plane in $E$. The image point whose ground coordinates are to be determined is $P_i$ (i.e. the point $P$ needs to be determined).

To calculate the z-coordinate of the point $P$ we focus on the triangle $TOB$, i.e. the zy-plane (see Figure D.4). The projection of $P$ along the x-axis on the zy-plane is the point $M$. The goal is now to determine $y = |OM|$. We distinguish two cases: (1) $P_i$ lies in the upper half of the image, and (2) $P_i$ lies in the lower half of the image.

1. **$P_i$ in upper half of image**

   We first calculate the Cross Ratio’s $\mathcal{R}(H_i,E_i;M,B)$ and $\mathcal{R}(H_i,E_i;M_i,B_i)$. 

---

90 Delft University of Technology, Computer Science, IN3405 - Bachelor of Science Project
Figure D.3: field of view and camera image

Figure D.4: the zy-plane
\[ \mathcal{R}(H, E; M, B) = \frac{|HM|}{|HB|} \cdot \frac{|EB|}{|EM|} \]

\[ = \frac{|HE| + |EM|}{|HB|} \cdot \frac{|EB|}{|EM|} \]

\[ = \left( \frac{|HE|}{|EM|} + 1 \right) \cdot \frac{|EB|}{|HB|} \quad (D.3) \]

\[ \mathcal{R}(H_i, E_i; M_i, B_i) = \frac{|H_i M_i|}{|H_i B_i|} \cdot \frac{|E_i B_i|}{|E_i M_i|} \equiv C_1 \quad (D.4) \]

Using the Cross Ratio Theorem, formulas (D.3) and (D.4) may be equated, and the following derivation can be made:

\[ \left( \frac{|HE|}{|EM|} + 1 \right) \cdot \frac{|EB|}{|HB|} = C_1 \]

\[ \Leftrightarrow \quad \frac{|HE|}{|EM|} = \frac{C_1 \cdot |HB|}{|EB|} - 1 \]

\[ \Leftrightarrow \quad \frac{|EM|}{|EB|} = \frac{C_1 \cdot |HE|}{|HB|} - 1 \quad (D.5) \]

The value for \( y \) now becomes:

\[ y = |OM| = |OE| + |EM| = |OE| + \frac{|HE|}{C_1 \cdot |HB|} \quad (D.6) \]

(2) \( P_i \) in lower half of image

(M now lies on the left side of \( E \) in Figure D.4)

Again we first calculate the Cross Ratio’s \( \mathcal{R}(H, M; E, B) \) and \( \mathcal{R}(H_i, M_i; E_i, B_i) \), giving:

\[ \mathcal{R}(H, M; E, B) = \ldots = \left( 1 + \frac{|EB|}{|ME|} \right) \cdot \frac{|HE|}{|HB|} \quad (D.7) \]

\[ \mathcal{R}(H_i, M_i; E_i, B_i) = \frac{|H_i E_i|}{|H_i B_i|} \cdot \frac{|M_i B_i|}{|M_i E_i|} \equiv C_2 \quad (D.8) \]

Equating (D.7) and (D.8), using the Cross Ratio Theorem, and performing a similar derivation as previously gives:

\[ |ME| = \frac{|EB|}{C_2 \cdot |HB|} \cdot \frac{1}{|HE|} \quad (D.9) \]

The value for \( y \) now becomes:

\[ y = |OM| = |OE| - |ME| = |OE| - \frac{|EB|}{C_2 \cdot |HB|} \cdot \frac{1}{|HE|} \quad (D.10) \]
The input values for equations (D.6) and (D.10) can easily be calculated using basic trigonometric functions:

- \(|OE| = h \times \tan(\alpha)\)
- \(|OH| = h \times \tan(\alpha - \theta)\)
- \(|OB| = h \times \tan(\alpha + \theta)\)
- \(|HE| = |OE| - |OH|\)
- \(|HB| = |OB| - |OH|\)
- \(|EB| = |OB| - |OE|\)

The values for \(C_1\) and \(C_2\) can be determined by calculating the proper pixel differences in the given image (see equation (D.4) and (D.8) respectively).

**Determining the x-coordinate** To calculate the x-coordinate of the point P we focus on the triangle TRL (see Figure D.5). The goal is to determine \(|MP|\).

![Figure D.5: triangle TRL](image-url)
We first calculate the Cross Ratio’s $\mathcal{R}(R,M;P,L)$ and $\mathcal{R}(R_i,M_i;P_i,L_i)$. 

\[
\mathcal{R}(R,M;P,L) \overset{(D.1)}{=} \frac{|RP|}{|RL|} \cdot \frac{|ML|}{|MP|} \\
(D.11)
\]

\[
= \frac{|RM| + |MP|}{|RL|} \cdot \frac{|ML|}{|MP|} \\
= \left( \frac{|RM|}{|MP|} + 1 \right) \cdot \frac{|ML|}{|RL|} \\
= \left( \frac{|RM|}{|MP|} + 1 \right) \cdot \frac{|ML|}{2|ML|} \\
= \frac{1}{2} \left( \frac{|RM|}{|MP|} + 1 \right) \\
\]

Equating (D.11) and (D.12), using (D.2), and performing a similar derivation as previously gives:

\[
x = |MP| = \frac{|RM|}{2 \cdot C_3 - 1} \overset{(D.13)}{=}
\]

The input value $|RM|$ for equation (D.13) can be calculated as follows:

$|RM| = \tan(\gamma) \times |TM| = \tan(\gamma) \times \sqrt{y^2 + h^2}$

The value for $C_3$ can be determined with the aid of formula (D.12).

Remark: formula (D.13) can be used to calculate $x$ regardless of whether $P$ lies on the left or right side of $M$ in Figure D.5. If $P$ lies on the right side of $M$ then $|M_iP_i|$ in (D.12) will be negative but the other values won't change sign (compared to the situation where $P$ lies on the left side of $M$), so $x$ will have the same value but with a negative sign.
Appendix E - Literature Survey 1: Background Subtraction Techniques

An important aspect of computer vision is the capability of computer systems to detect foreground objects. Background subtraction techniques detect foreground objects by comparing every new frame to a representation of the image background. In this article, the Mixture of Gaussians, Chromaticity and Gradient Based Model and Eigenbackgrounds techniques are introduced. These techniques are compared to evaluate their effectiveness at detecting foreground objects in an outdoor street view camera image in real-time. The Chromaticity and Gradient Based Model is concluded to be least applicable as it is least capable of adapting to highly dynamic backgrounds. The Eigenbackgrounds technique provides good results and is easy to set up, while the Mixture of Gaussians technique yields best results at the expense of additional setup time.

Introduction

The detection of foreground objects using a single static camera image is a fundamental problem in the area of computer vision [4]. An established group of techniques for foreground object detection are background subtraction techniques. Background subtraction is a group of techniques that detect moving objects by comparing a camera frame to the background. This background information may be a static picture or a statistic model of the background [28]. Detecting these foreground objects has a wide range of applications, such as human detection [19], human tracking, traffic monitoring, human-computer interaction, and automated video surveillance [5].

Outdoor street-view cameras are commonly used for traffic monitoring and automated video surveillance. The positioning of these cameras imposes some additional challenges to background subtraction techniques compared to indoor cameras that will be discussed in more detail later. The outdoor environment can not be completely controlled as is the case for indoor cameras. Additionally the cameras are placed at heights of around 5 meters, and at a distance of over 10 meters from the objects of interest.

Different surveys of commonly used background subtraction techniques have previously been performed by Piccardi [28], Kuralkar & Gaikwad [18] and by Benezeth, Jodoin, Emile, Laurent & Rosenberger [4]. They assess the qualities and weaknesses of multiple techniques, and conclude that each different technique is applicable in a different situation. In their articles these different situations are not discussed in more detail however. Solely an overview of the qualities of each technique is given. In particular the specific case of outdoor street-view cameras is not discussed in these articles.

This article will focus on evaluating the aptness of three different background subtraction techniques at detecting foreground objects from a street-view camera in real-time. The first technique is the Mixture of Gaussians technique. The popularity of this technique is confirmed by the fact that it is integrated into the OpenCV library, which is the main open source library that is available for computer vision applications. The technique was originally developed to handle outdoor environments by being able to adapt to lighting conditions and dynamic background objects [32].
The second technique uses a statistical model based on chromaticity and gradient information. Though this technique was not specifically designed for outdoor applications, it does provide means for shadow detection [15]. Additionally it was applied successfully in the process of tracking groups of humans in an outdoor scene [24].

The final technique is Eigenbackgrounds. This technique was designed specifically to handle dynamic lighting conditions. It naturally applies to the specific case of outdoor street-view cameras as it detects the smallest moving objects [28, 26]. Each of the three techniques were previously shown to yield promising results at detecting foreground objects and are promising for the specific case of this article.

The concept of background subtraction and the problems introduced by the properties of the street-view camera will be discussed in section E. Next, the three different techniques will be introduced in Section E. The techniques are then evaluated in Section E based on three different criteria: is the technique simple to set up regarding the parameters that must be set? Is the technique robust considering the problems introduced in section E? Is the technique computationally efficient enough to run in real-time? Finally, in Section E conclusions will be drawn on which techniques and what properties make techniques suitable for real-time foreground object detection using a street-view camera.

The Background Subtraction Process

The goal of background subtraction techniques is to detect all the objects that make up the foreground of an image. It could be naively described as comparing an image to the static background of the camera’s scene, and taking the absolute difference between each pixel and its corresponding pixel in the static background [28]. This difference is then thresholded at a value $\theta$, the sensitivity to change, to return a binary mask image containing the foreground objects. This basic operation is described by equation E.1. The result of a successful background subtraction is shown in Figure ??.

$$\text{Foreground} = |\text{Frame} - \text{Background}| > \theta \quad (E.1)$$

The initial problem lies within obtaining the static background of the camera’s scene and selecting the value for $\theta$. The most simple approach for obtaining the foreground is by assuming that the foreground object consists of colors different from the static background scene. The camera is assumed to be static and noise-free with constant lighting conditions [4]. In this case the static background scene could simply be an image of the background taken at an earlier moment when no foreground objects were in the image.

Unfortunately problems occur in real-life situations, especially in the case of outdoor street-view cameras. Generally cameras suffer from noise which causes a pixel to rarely have the exact same value in each consecutive frame. The illumination of the image can change rapidly due to weather conditions and slowly due to the day-night cycle. Objects may cast shadows, which should not be part of the foreground objects.

Additionally, the background may not be completely static: trees may move due to the influence of wind and objects such as cars may enter or leave the background scene. This property is also called multi-modal backgrounds. Objects that only remain static for a short predetermined period should not be incorporated into the background however.

Another important difficulty is that in the case of a street-view camera the objects of interest may not be the largest foreground objects in the image. Each of these problems will cause erroneous outcomes for the simple background subtraction approach described at the beginning.
Background Subtraction Methods

In this Section three different commonly used background subtraction techniques will be introduced: The first technique models the background as a mixture of Gaussians. The second technique on the other hand uses a statistical approach based on chromaticity and gradient information for each pixel. The final discussed technique, Eigenbackgrounds, uses no statistical models. It rather performs a background subtraction by using Principal Components Analysis [27].

Each of these techniques belong to the group of more advanced background subtraction techniques. Simple techniques such as running average or median background subtraction perform well when the background situation is completely static and the camera is noise-free. However, for more advanced situations with highly dynamic backgrounds these techniques do not manage to provide good results [28, 4].

Mixture of Gaussians

As part of the OpenCV open source computer vision library the Mixture of Gaussians(MoG) is a commonly used background subtraction technique. This approach was first introduced by Stauffer & Grimson in 2000 [32] and uses a mixture of Gaussian distributions to model the background scene for each pixel. This introduction will closely follow their explanation of the technique.

The MoG technique considers a pixel to be foreground until one of its Gaussian distributions consistently includes it with a high probability. Unlike techniques that model the background scene using a single Gaussian, the MoG technique has a memory that can contain multiple background scenes. When a background object is temporarily replaced it will be maintained in the background model, allowing a faster recovery when the replacing object is removed. This allows high adaptivity for each of the problems imposed by outdoor cameras: illumination changes, waving trees and objects moving in and out of the background scene.

Every pixel in the image has $K$ Gaussian distributions describing the background model, where $K$ may be chosen freely. Higher values for $K$ go at the expense of performance however. Every $i^{th}$, $1 \leq i \leq K$ distribution has a weight $w_i$, with $\sum_{i=1}^{K} w_i = 1$. Finally, $\Sigma_i$ is the covariance matrix.
of the $i^{th}$ Gaussian and $\mathcal{N}$ is the Gaussian probability density function that describes each Gaussian distribution and is shown in equation E.2. Figure E.2 displays three Gaussian distributions that model the background of a pixel in RGB space.

$$\mathcal{N}(X, \mu, \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} \exp\left(-\frac{(X - \mu)^2}{2\Sigma}\right)$$ (E.2)

The RGB values of every new pixel $X$ are checked against its $K$ distributions. When the Mahalanobis distance to a distribution is smaller than 3 [40] the pixel is matched to this distribution. If no match exists the Gaussian distribution with the smallest weight is replaced by a distribution with the pixels RGB values as initial means, and an initially large variance. The weights of the distribution are then adjusted:

$$w_i = (1 - \alpha)w_i + \alpha M_i$$ (E.3)

Where $M_i = 1$ if the $i^{th}$ distribution was matched to the pixel, and $M_i = 0$ otherwise and $\alpha$ is the rate of adaptivity which may again be chosen freely. The means and variances for unmatched distributions remain unchanged, while the mean and variance of a matched distribution is updated using the following equations:

$$\mu = (1 - \alpha)N(X, \mu, \sigma)\mu + \alpha N(X, \mu, \sigma)X$$ (E.4)  
$$\sigma^2 = (1 - \alpha)N(X, \mu, \sigma)\sigma^2 + \alpha N(X, \mu, \sigma)(X - \mu)^2$$ (E.5)

Finally, to determine if a pixel is considered foreground the Gaussian distributions are first ordered by $w/\sigma$. Distributions that are often matched to the pixel have a high weight, meaning there is much reason to believe it is part of the background. The $\sigma$ of distributions modeling a very static background surface is small resulting in a high value for $w/\sigma$ for likely background distributions. Next, the first $B$ distributions that make up the background model are chosen from the pool of $K$ distributions as follows:

$$B = \text{argmin}_b \left( \sum_{i=1}^{b} w_i > (1 - \tau) \right)$$ (E.6)
Whenever a pixel is matched to a Gaussian distribution that is part of $B$ it is considered a background pixel. Otherwise it is considered as a foreground pixel and added to the binary mask containing the foreground objects. In this equation $\tau$ is chosen freely between 0 and 1 and determines the maximum fraction of the data that may be accounted for by foreground pixels. Together with $\alpha$ this parameter determines the amount of frames $n_f$ that an object should remain static in order to become part of the background model, according to equation E.7 [40].

$$n_f = \log(1 - \tau)/\log(1 - \alpha)$$

(E.7)

When the value for $\tau$ approaches 1 this will result in a background model consisting of only one Gaussian distributions. A value for $\tau$ that approaches 0 allows for multiple distributions in the background model, such that dynamic multi-modal backgrounds can be modelled by these distributions. This results in a more robust technique regarding dynamic backgrounds at the expense of performance.

**Statistical Model based on Chromaticity and Gradient information**

Many background subtraction techniques base their background information on the regular RGB information of the image [28]. Initial approaches were described by Horprasert, Harwood & Davis [15] as a separate technique and by Mckenna, Jabri, Duric, Wechsler & Rosenfeld [24] as part of their process of tracking groups of people. There are also more recent successful applications by for instance Yuan and Yang [39] who used the chromaticity and gradient based model technique as part of their robust system for detecting human actions by using a single camera. The description given in this article is a mixture of descriptions given in there articles.

Techniques that use the RGB information in combination with single distribution per pixel yield good results when illumination changes slowly. However when illumination changes very rapidly due to shadows these techniques struggle at adapting. When the pixels are converted from RGB to YUV color space this problem can be circumvented [39]. The YUV space consists of two chromaticity components and a brightness component. When an area is cast in shadow this results in a large change for the brightness component, while the chromaticity components are insensitive to this change [24].

Given that the means and variances of the two chromaticity components $r$ and $g$ and brightness component $b$ are known for the pixel at time $t$ they can be calculated for the newest frame at time $t + 1$ using the following equations [20] and the corresponding new value $x_{t+1}$:

$$\mu_{t+1} = (1 - \alpha)\mu_t + \alpha x_{t+1}$$

(E.8)

$$\sigma^2_{t+1} = (1 - \alpha)(\sigma^2_t + (\mu_{t+1} - \mu_t)^2) + (1 - \alpha)(x_{t+1} - \mu_{t+1})^2$$

(E.9)

In this equation the value for $\alpha$ is the rate of adaption similar to the MoG technique described earlier. This value may be chosen freely and determines how fast the background model adapts to changes in the image. Based on the chromaticity and brightness models and current chromaticity and brightness values $r, g, b$ a pixel from the new image may now be classified as either background, shadow or foreground using the following rules [39]:
The values $T_{b1}$ and $T_{b2}$ are thresholds for the absolute brightness difference. Typical values are multiples of $\sigma^2_b$ and $T_{b2} > T_{b1}$. The variables $T_r$ and $T_g$ are threshold values for the two chromaticity values $r$ and $g$, again typical values are multiples of $\sigma^2_g$ and $\sigma^2_r$.

This foreground segmentation based on chromaticity and gradient information alone is not sufficient however. In cases when there is no change in chromaticity between foreground and background, for example when a person wearing a green jacket moves in front of a grass patch, this segmentation alone will not detect the person as foreground. This can be solved by using a Canny edge detector \[39, 6\] on the brightness component of the YUV image. All connected components that are larger than a set threshold can then be found in the resulting image. This component is filled and the result is concatenated with the binary mask found after the segmentation result from equation E.10 to find the final binary image containing the foreground objects.

### Eigenbackgrounds

Unlike the other methods the Eigenbackgrounds technique uses a different approach than modeling each pixel using Gaussian distributions. It instead performs a Principal Component Analysis \[27, 16\] on the complete image. This technique was first introduced by Oliver, Rosario & Pentland \[26\] as part of their Bayesian system for modeling human interactions and this description will use their description as a guideline.

For the Eigenbackground method a sample of $n$ frames is taken. The $n$ frames are then ordered as columns of a matrix $A$. Next the mean vector $\mu$ and covariance matrix $\Sigma = AA^T$ are calculated from this matrix $A$. The covariance matrix is then diagonalized by using an eigenvalue decomposition:

$$L = \Phi \Sigma \Phi^T$$  \hspace{1cm} (E.11)

Here $L$ is the diagonal matrix containing the eigenvalues of the covariance matrix $\Sigma$. $\Phi$ is the eigenmatrix of the covariance matrix $\Sigma$. To reduce the dimensionality of the image to a lower dimensionality only the first $m$ eigenvalues and corresponding eigenvectors are kept. This value for $m$ may be chosen freely between 1 and the total size of the image. Because the eigenvalues are ordered by the amount of variance in the original picture that they describe these $m$ eigenvectors describe as much of the variance in the original image as possible. This step results in the matrix $\Phi_m$ consisting of these first $m$ eigenvectors.

When a new frame $X$ is then captured it is first mean-normalized by subtracting the mean vector $\mu$ and then projected onto the eigenvectors contained by the matrix $\Phi_m$ to get the image $X'$ which models the background. The eigenvectors model the background, because moving objects are not in the same location during the sequence of $n$ frames and therefore make only a small contribution to the background model. This image $X'$ is then reconstructed and the absolute difference with the original image $X$ is computed. By using a basic threshold such as equation E.1 the foreground can then be found.
Discussion of the Three Criteria

In the previous Sections the basic principle of background subtraction and its problems were introduced. The workings of the three different techniques that are discussed in this paper were explained as well. The different qualities and flaws of each of the techniques can now be discussed, as well as their aptness of detecting foreground objects in street-view camera images in real-time.

This will be assessed based on the three criteria that were previously mentioned in the introduction: Is the technique simple to set up regarding the parameters that must be set? Is the technique robust regarding the problems introduced by the outdoor street-view property of the cameras that were described in section E? Is the technique computationally efficient enough to run in real-time? For each criterion the strengths and weakness of all three techniques will be discussed and compared.

Ease of Parameter Selection

The amount of parameters that must be set provides a measure for the ease of implementation. To fine-tune the parameters the quality for the foreground detection must be evaluated for a range of different combinations of parameters. When performing background subtraction it is difficult to evaluate the quality of the results [18]. One option is to select a ground truth foreground by hand, and computing the amount of different pixels between the result and the ground truth [15].

For each additional parameter the number of possible combinations increases quadratically. This makes selection of the correct parameters a time consuming process. More parameters therefore make setting up the background subtraction technique more difficult and can only be justified when the additional flexibility results in more accurate foreground detection.

The Mixture of Gaussians (MoG) method is a very flexible background subtraction technique as it provides many parameters that can be set freely. The most important parameters are the adaptivity rate $\alpha$, the maximum portion of the image accounted for by foreground $\tau$ and the amount of Gaussian distributions per pixel $K$. Each of these parameters provides significant advantages in adapting the MoG technique for street-view real-time foreground detection.

As was also discussed in Section E the parameters $\tau$ and $\alpha$ together make up the number of frames $n_f$ that an object must remain static in order to be incorporated into the background model according to equation E.7. However, $\alpha$ influences the rate of adaptivity for the Gaussian distributions regarding the changes in illumination caused by weather or day-night cycles as well. The value for $\tau$ determines the maximum portion of the image that is allowed to correspond to foreground objects.

Selecting a value for $n_f$ depends on the situation as well as the requirements of consequent processes and is therefore a subjective choice. More importantly the values $\tau$ and $\alpha$ must be fine-tuned in order to achieve the required value for $n_f$. Finally, there is the parameters $K$ that has a similar effect to decreasing $\tau$. An increase of $K$ allows MoG to adapt to multi-modal backgrounds at the expense of performance. More Gaussian distributions must be searched when matching a pixels RGB values to its $K$ distributions. Zivkovic and van der Heijden [40] presented an improved MoG implementation that automatically selects the optimal value for $K$ such that this parameter needs not be fine-tuned.

The chromaticity and gradient based background model approach allows for selection of parameters as well. Similar to the MoG technique this approach has a rate of adaptivity $\alpha$ for its background model. Additionally the thresholding step displayed in equation E.10 allows for selection of the threshold values $T_{b_1}$, $T_{b_2}$, $T_r$ and $T_g$. 

Appendix E. Literature Survey 1 : Background Subtraction Techniques
Intelligent Multi-camera
Unlike the MoG technique there exist default values for these parameters that generally yield good results. These default were discussed in Section E and used by McKenna et al. [24] for successfully tracking groups of people. Horprasert et al. [15] present techniques for automatically selecting these threshold values as well. Fine-tuning of the parameters may still be performed to achieve better results, but it is not necessary as the default values were shown to yield usable results as well.

Finally, the Eigenbackgrounds method provides only one parameter that can be set. The target number of eigenvectors \( m \). Though fine-tuning is a time consuming process the fine-tuning of a single parameter requires much less effort than the multiple parameters that are required for the MoG technique or the chromaticity and gradient based background model. However as stated by Piccardi [28] a suitable training set must also be selected for the eigenmatrix.

### Robustness to Outdoor Street-view Camera Properties

Though determining the correct values for the parameters is a time consuming process it does provide a high degree of flexibility to customize the technique. It can be possible to adapt more general background subtraction techniques to the specific case of outdoor street-view cameras by adapting the parameters to yield good results. Other techniques may apply naturally to the case of outdoor street-view cameras.

In the case of MoG the technique was not specifically designed for outdoor street-view camera images. Stauffer and Grimson [32] have tested their technique on such images however and achieved good results. The main cause for this is the flexibility provided by the parameters that can be set. As mentioned in Section E objects that remain static for only a short period should not be incorporated into the background.

For street-view cameras a typical value for \( n_f \) is over 600 frames, which is 30 seconds at 20 frames per second. When a desired value for \( n_f \) is selected this value must be realized by tuning the parameters \( \alpha \) and \( \tau \) such that they result in this value \( n_f \) according to equation E.7. The objects of interest are typically over 10 meters away from the camera, so that they usually occupy less than 25% of the image. A value for \( \tau \) smaller then 0.25 may then be chosen. The MoG technique then allows for multi-modal backgrounds to be incorporated into the background model. Once \( \tau \) is selected the value for \( \alpha \) is fixed, because a desired value for \( n_f \) is chosen. It is shown that the parameters of the MoG technique allow for adaption to the case of outdoor street-view cameras.

The chromaticity and gradient based model is not specifically designed for outdoor street-view camera images like the MoG technique. Regardless Horprasert et al. [15] and Yuan et al [39] demonstrate that the chromaticity and gradient based model performs well under changes of illumination for the image.

McKenna et al. [24] demonstrate that the technique applies to tracking groups of people in an outdoor environment as well when using default parameters. Their camera positioning does not correspond to that of a typical street-view camera however. Instead of being positioned at an angle, their camera is positioned at an equal level to the objects of interest and at a distance shorter than that of a street-view camera.

These properties make this background subtraction technique less apt at foreground detection in the case of outdoor street-view cameras. The gradient step of the technique requires the objects of interest to be relatively large compared the other objects in the image [39]. Additionally, the single Gaussian distribution used to model the background is unable to capture multi-modal backgrounds.

Though the technique is invariant to illumination changes this does cause the technique to adapt to new objects entering the background very slowly. More importantly a single model is not
able to incorporate a dynamic background object such as a waving tree. In that case pixel values may alternate between green pixels that correspond to the leaves and blue pixels that correspond to the sky behind it. A single model can not capture both these background scenes.

The Eigenbackgrounds technique naturally applies to the outdoor street-view camera images. The eigenvectors used for the Principal Component Analysis are ordered by the amount of variance in the image that they describe. Therefore, picking a larger number for $m$ results in more details being included in the background matrix. This means that when $m$ is increased less objects and only smaller objects are marked as foreground. When $m$ is decreased the background matrix becomes more general, so that more objects are labelled as foreground.

Though the Eigenbackgrounds technique does not have a rate of adaptivity $\alpha$ the Eigenmatrix $\Phi$ may still be updated to adapt to illumination changes [26]. A problem arises when a moving object is captured in the training set used to calculate the Eigenmatrix. A foreground object in the same location as this moving object will not be completely captured as foreground when this Eigenmatrix is used [28]. This same property provides robustness against dynamic background objects such as waving trees however.

**Capability of Running in Real-time and Computational Efficiency**

The MoG technique was shown to run in real-time in the year 2000 by Stauffer & Grimson [32]. The improved technique by Zivkovic et al. [40] was shown to run even more efficiently in real-time on 2 Ghz PC in 2006.

Horprasert et al. [15] initially designed the chromaticity and gradient based model technique to run in real-time. They provided suggestions for speeding up the algorithm through parallelisation. McKenna et al. [24] used the technique in its system for tracking groups of humans. The entire system was able to run in real-time. Yuan et al [39] implemented a real-time human action recognition system that used the technique. Despite the use of the computationally heavy Canny edge detector [6] this system was also capable of running in real-time.

The Eigenbackgrounds technique was shown to provide good results while maintaining a higher computational efficiency than the MoG technique as was shown by Oliver et al. [26, 28] in 2000.

**Comparing the Techniques Based on the Three Criteria**

An overview of the techniques and discussed criteria is shown in Table ??.

When evaluating the first criterion it follows that the Eigenbackgrounds is by far most easy to set up. The technique has only a single parameter that requires finetuning such that an optimal value can easily be computed. It should not be forgotten that a suitable training set needs to be selected for the eigenmatrix to yield good results for outdoor background subtraction however.

The Mixture of Gaussians and Chromaticity and Gradient based Model approach require finetuning of respectively three and five parameters. As discussed previously this makes the process of finetuning these parameters and therefore setting up the technique much more time consuming. Though in the case of the Mixture of Gaussians approach finetuning of the parameters is obligatory to yield accurate results, the Chromaticity and Gradient model approach has a set of suitable default parameters. This makes finetuning the parameters an optional procedure for the latter technique allowing a simple initial setup.

When evaluating the second criterion it becomes apparent that each of the techniques is capable of adapting to slow changes in illumination in the outdoor street-view camera images. The MoG technique may adapt by updating its models. In the case of a single background model adapting would take as long as the previous background image was present [40]. The multi-modal nature of the background model allows much faster adapting based on the selected value for $n_f$. The chro-
Table E.1: an overview of the three techniques based on the three discussed criteria

<table>
<thead>
<tr>
<th>Technique</th>
<th>Selection of parameters</th>
<th>Robustness to outdoor environment</th>
<th>Computational efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixture of Gaussians</td>
<td>Three parameters, Costly finetuning</td>
<td>Excellent adaption to (multi-modal) outdoor environments</td>
<td>Runs in real-time, Mediocre efficiency</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chromaticity and Gradient based</td>
<td>Five parameters, Finetuning extremely</td>
<td>Best at close range/low angle, not typical for street-view cameras,</td>
<td>Runs in real-time, Lower efficiency</td>
</tr>
<tr>
<td>model</td>
<td>difficult, Suitable defaults exist</td>
<td>Can not handle multi-modal backgrounds</td>
<td></td>
</tr>
<tr>
<td>Eigenbackgrounds</td>
<td>Single parameter, Easily finetuned,</td>
<td>Naturally applies to street-view by detecting smaller objects,</td>
<td>Runs in real-time, Highest efficiency</td>
</tr>
<tr>
<td></td>
<td>Selection of training set</td>
<td>Carefully selected training sets handle multi-modal backgrounds</td>
<td></td>
</tr>
</tbody>
</table>

The chromaticity and gradient based model is naturally robust to illumination changes due to the brightness invariant chromaticity components that are used for the background model. The Eigenmatrix of the Eigenbackgrounds method can be updated over time to adapt to illumination changes as well.

The chromaticity and gradient based model is invariant to more fast changes in illumination such as shadow as well due to its chromaticity components. The MoG does not naturally provide means for shadow detection. The improved version of the technique that was introduced by Zivokovic et al. [40] does have shadow detection mechanisms. Oliver et al. [26] provide no technique for detecting shadows when using Eigenbackgrounds.

The chromaticity and gradient based background model is less robust to dynamic backgrounds such as moving trees or objects leaving and entering the background scene. The MoG technique may adapt to dynamic backgrounds by setting its parameters to allow for multi-modal backgrounds as described previously. The Eigenbackgrounds technique more naturally adapt to this property of outdoor street-view cameras due to the concept of Principal Components Analysis.

Overall, the Eigenbackgrounds technique most naturally applies to the case of outdoor street-view camera image background subtraction. It requires selection of only a single parameter and requires less computational performance than the other two techniques. The Eigenbackgrounds technique has no means for shadow detection and may result in erroneous foreground masks when moving objects are captured in the frames used to calculate the Eigenmatrix. To achieve the best results the Mixture of Gaussians technique is more suitable. The different parameters that may be set provide flexibility to achieve good results, while the technique itself is most robust to dynamic background scenes. The chromaticity and gradient based model is runnable in real-time but least computationally efficient of the three techniques. Additionally, it has problems adapting to dynamic backgrounds as only a single Gaussian distribution is used for modelling the background. The gradient step of the technique is also less efficient when objects of interest are further away, which is a property of street-view camera images.
Finally it becomes apparent from the run time discussion that each of the three techniques meet the third criterion and are capable of performing in real-time. Especially when using modern quad-core computers and making advantage of parallelisation to perform pixel-by-pixel operations these techniques may achieve framerates of over 30 frames per second at 320 x 240 image resolutions. This leaves sufficient computational resources for subsequent image processing processes such as human recognition to further process the detected foreground objects.

Conclusion

In this article three different background subtraction techniques were compared to determine which technique is most suitable for detecting foreground objects in an outdoor street-view camera image in real-time. First, an introduction of background subtraction was given. Next, the problems that arise when performing background subtraction on images of outdoor street-view cameras were identified. The three techniques were then introduced: the Mixture of Gaussians technique which models the background for each pixel by using a Gaussian mixture model, the Chromaticity and Gradient based model which models the background for each pixel by using two Gaussians based on chromaticity and gradient values and finally the Eigenbackgrounds technique which performs a Principal Component Analysis on the image.

The techniques were then compared based on three criteria: Is the technique simple to set up regarding the parameters that must be set? Is the technique robust regarding the problems introduced by the outdoor street-view property of the cameras that were described in Section E? Is the technique computationally efficient enough to run in real-time?

The Eigenbackgrounds technique was found to yield good results and as it only has a single parameter it remains simple to set up. Additionally it is more computationally efficient than the other two techniques and extends naturally to outdoor cameras when a correct training set is selected. The Mixture of Gaussians method provides the most accurate results and is most robust to the problems imposed by the outdoor street-view camera images due to the flexibility provided by its parameters and capability to handle multi-modal backgrounds. The parameters require time-consuming fine-tuning however, making it more difficult to set up. The Chromaticity and Gradient Based Model the is least computationally efficient. Though it has many parameters, suitable defaults exist making the fine-tuning optional. Additionally, this model is least capable of adapting to the dynamic multi-modal backgrounds of outdoor street-view camera images and performs best at distances and angles that do not correspond to those of outdoor street-view cameras.

If more time were available the study should be extended to evaluate the different techniques in practice by testing each technique on identical testing video material and comparing the results to a group truth. This would additionally provide a precise performance benchmark.
Appendix F - Literature Survey 2: Classification Techniques

This appendix gives an overview on the most important solutions to the problem of classifying binary pixel regions in images as humans. Four techniques are discussed followed by a comparison: a silhouette-based classifier, a shape-based neural network, a pixel-based neural network and a Mahalanobis distance classifier. The first, second and last technique are feature-based, meaning that they calculate one or more characteristic values out of the given input image that are then used in the classification process. The third is pixel-based, meaning that classification is directly based on the pixels of the input image.

Introduction

Detecting humans in images is an important problem with useful applications. For example, in order for an automatic visual surveillance system to recognize and reason about suspicious or unwanted behavior, it is essential that it has the capability of detecting humans. Another example is related to autonomous vehicles and the problem of detecting people traversing a pedestrian crossing.

Throughout the last decades several algorithms have been developed for detecting objects in a sequence of captured camera images. These techniques are referred to as background segmentation\(^1\). An overview of the most commonly used techniques is given in [18]. Once the moving objects are detected the next step is to classify these as belonging to a certain group or being of a particular type. This is referred to as object classification.

Finding a robust and well-performing algorithm for this problem that can be executed by a computer proves to be a challenging task [10]. People can take on different poses or may be only partially visible in the image. Furthermore, the context of the observed scene has often to be taken into account.

This paper gives an overview and comparison on the most important solutions to the problem of object classification. More specifically four commonly used techniques will be described, followed by a discussion comparing the various strengths and limitations of each technique to see if one can be declared as overall winner. The techniques in this papers each approach the problem from a another perspective and use different mathematical concepts to analyze and reason about the problem.

The remainder of this paper will be structured as follows. Section 2 will formalize the problem of object classification and section 3 will discuss the four classification techniques. Section 4 will compare these techniques and describe their main advantages and associated limitations. In Section 5 the paper will be concluded.

\(^1\)Sometimes one can also encounter the term foreground detection.
Problem description and formalisation

The problem of object classification can generally be described as follows. Given a stream of colored images captured from a camera, we want to determine for each image which pixel regions correspond to moving humans in the scene observed by the camera. This problem can be divided in two parts. The first determines the pixels belonging to moving objects, by deleting all the background pixels. This is referred to as background segmentation (see [18, 13] for surveys on these subject). The second, the part this paper will focus on, classifies the detected pixel regions as representing a human or not. This is referred to as object classification.

Background segmentation techniques return a binary image. This is an image containing only white and black pixels (respectively 1 or 0). Black pixels refer to the background and white pixels to the foreground, i.e. moving object(s) detected from the captured sequence of colored input images. Every detected foreground object in this binary image is also referred to as a blob.

Bounding boxes can be drawn around the detected blobs, to indicate the presence of a detected object. Figure F.1 shows an example of a binary image containing the blobs of a walking person and a bird, surrounded by their corresponding bounding boxes.

![Figure F.1: example of a binary image containing two detected blobs and their associated bounding boxes. (Original images: [17] and [1])](image)

Every object classification technique discussed in this paper assumes a cropped binary image of the blob as its input and returns an identifier indicating the determined type of the object. In the case of human classification the output will be human or non-human. See Figure F.2 for an example.

Image classification techniques can be divided in two categories: feature-based and pixel-based classification. Feature-based techniques first calculate one or more characteristic values out of the given input image. Subsequently, these values are given as input to a specific classifier, which returns the type of the object. Pixel-based techniques do not calculate any features in advance but instead base their classification directly on the pixels of the input image.

Classification techniques

This Section discusses four classifiers: a silhouette-based classifier, a shape-based neural network, a pixel-base neural network and a Mahalanobis distance classifier. The first, second and last technique are all feature-based. The third is pixel-based.

The field of object classification is broad and driven by a lot of research. The techniques in this paper have been carefully selected based on their representation and analyzation of the problem.
Several techniques could not described in this paper but are definitely worth mentioning for possible further reading. Firstly, silhouette-based descriptors are often based on Fourier descriptors which describe a silhouette shape by a fixed number of lowest frequency components [29]. Another commonly used technique uses optical flow features in combination with a Gaussian model to detect moving objects [38]. Lastly, a very promising technique uses Histograms of Oriented Gradient (HOG) features in combination with a State Vector Machine (SVM) classifier to detect moving humans [10].

**Silhouette-based classifier**

This technique classifies the binary blob image by looking at its contour and comparing this to the contours of previously collected images of which the human poses are already known. This technique is described in more detail in [11]. The description below follows the same reasoning.

First the centroid, $C = (x_c, y_c)$, of the binary input image $f$ is calculated as follows:

$$
\text{area} = \sum_{x=0}^{\text{width}} \sum_{y=0}^{\text{height}} f(x, y) \\
x_c = \frac{\sum_{x=0}^{\text{width}} \sum_{y=0}^{\text{height}} xf(x, y)}{\text{area}} \\
y_c = \frac{\sum_{y=0}^{\text{height}} \sum_{x=0}^{\text{width}} yf(x, y)}{\text{area}}
$$

(S.1)

Next, the contour of the object region is extracted from the binary image. This can be achieved by setting all pixels whose 4-connected neighbors are all 1 to 0. Another well-known algorithm uses freeman chain codes for contour detection, which is discussed in [35]. An example is given in Figure F.3.

Let the image have $n$ boundary points $p_i$, ordered from the boundary point $p_1$ above the centroid in a clockwise manner, and define $B$ as the list of these ordered boundary points: $B = \{p_1, ..., p_n\}$. From this list a so called distance signal [11] $D = \{d_1, ..., d_n\}$ can be generated containing the
distance between each boundary point $p_i$ and the centroid $C$:

$$d_i = d(C, p_i) \quad i = 1, 2, ..., n$$  \hspace{1cm} (F.3)

In order to compare a series of signals belonging to different sized object images, the length of each signal has to be reduced to the same length. Suppose this length is some predefined constant $K$. The new signal is then computed as follows:

$$\hat{D}[i] = D \left[ i \times \frac{n}{K} \right] \quad i = 1, 2, ..., K$$  \hspace{1cm} (F.4)

Next, the acquired signal $\hat{D}$ is normalized:

$$\hat{D}_N[i] = \frac{\hat{D}[i]}{\sum_{i=1}^{n} \hat{D}[i]}$$  \hspace{1cm} (F.5)

An illustration of a normalized distance signal is given in Figure F.4.

The signal from equation (F.5) is the descriptor that will be used to compare a blob with a predefined database. This database contains numerous similar descriptors from a variety of representative human poses together with a corresponding label indicating their type. This database is constructed offline by manually assigning type labels to a set of extracted boundary descriptors.

The comparison of the newly acquired object descriptor to the pre-established database uses a pairwise distance formula to measure the similarity between two descriptors. Suppose $P$ is a new object and let $Q$ denote an object from the database. Let their descriptors be $\hat{D}_{N_P}$ and $\hat{D}_{N_Q}$ respectively. The similarity between those two objects is then expressed by computing the distance $d_{PQ}$ between their two descriptors as follows:

$$d_{PQ} = \sum_{i=1}^{K} |\hat{D}_{N_P}[i] - \hat{D}_{N_Q}[i]|$$  \hspace{1cm} (F.6)

To determine the type of object $P$ its descriptor can now be compared to all object descriptors.
Figure F.4: illustration of a normalized centroid-boundary distance signal, as function of the boundary points

in the database. The type label of object Q is assigned to object P if and only if there is no other object in the database whose descriptor distance to P is smaller than that to Q, and the descriptor distance is below some predefined threshold T. More formally, the following two conditions must hold:

1. \( d_{PQ} \leq d_{PI} \) for every object I in the database
2. \( d_{PQ} \leq T \)

Shape-based neural network

This technique first calculates a list of shape parameters which are then supplied as input to a pre-trained neural network classifier. This network has a unique output value for every object type that can be distinguished. In the case of human classification there is one output node whose value is binary: 1 if the object is of type human and 0 otherwise.

A complete discussion and background on neural networks can be found in [25, 30].

Initially, after determining the number of hidden layers and the size of each layer (number of neurons), every weight of the neural network is uniformly assigned a random number between two predefined boundaries.

Next, the network can be trained. This is done by supplying it a series of input values of which the expected output is already known. Each time the output calculated by the network is different from the expected output an error function determines the difference between them. In order to correct for the error that was made, the determined error difference is used to change the weights of the network, in other words: the network learns from its mistakes. A technique that is widely used to achieve this is the backpropagation algorithm [30]. This algorithm propagates from right to left back through the network, adjusting the weights of every synaps while passing them by, according to some fixed set of rules.

Table F.1: Shape parameters and their formula

<table>
<thead>
<tr>
<th>Shape parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formfactor: ( \frac{4\pi \times \text{area}}{\text{perimeter}^2} )</td>
</tr>
<tr>
<td>Roundness: ( \frac{1}{\sqrt{1+4 \times \frac{\text{height}}{\text{width}}} - 1} )</td>
</tr>
<tr>
<td>Aspect ratio: ( \frac{\text{height}}{\text{width}} )</td>
</tr>
<tr>
<td>Eccentricity: ( \sqrt{1 - \left( \frac{\text{height}}{\text{width}} \right)^2} )</td>
</tr>
<tr>
<td>Parallelism: ( \frac{\text{height}}{\text{width}} )</td>
</tr>
<tr>
<td>Convexity: ( \frac{\text{area of the convex hull}}{\text{area of the object}} )</td>
</tr>
<tr>
<td>Convexity: ( \frac{\text{area of the object}}{\text{area of the convex hull}} )</td>
</tr>
<tr>
<td>Area: ( \text{area of the object} )</td>
</tr>
<tr>
<td>Perimeter: ( \text{perimeter of the object} )</td>
</tr>
</tbody>
</table>

Appendix F - Literature Survey 2: Classification Techniques
Intelligent Multi-camera Video Surveillance, Final Report
supplied as input to the neural network. A global overview of the neural network is given in Figure F.5.

The network can be trained by providing it shape parameters computed from images with known object type, so the expected values of the neurons in the output layers are known (e.g. input parameters corresponding to a human image should result in an output of (1,0,0,0) in Figure F.5). If the output-values do not match with the expected output, then the weights of the network will need to be adjusted to correct for the error that was made, using for example the supervised back-propagation algorithm.

![Diagram of neural network](image)

**Figure F.5: global overview of the shape-based neural network**

**Pixel-based neural network classifier**

This technique is very similar to the previously described shape-based neural network. The only difference lies in the input layer of the network. The network doesn’t require any shape-parameters as its input, instead every input neuron corresponds with a pixel value of the binary input image. Figure F.6 illustrates this approach to the problem of human classification. Classification thus completely proceeds based on the original image information rather than on a set of calculated image features. During the training phase the network is now not supplied with shape parameters, as was previously the case, but instead it receives the original binary image pixel values as input. Every training image is processed by the network and when a classification error is made the weights are properly adjusted, using for example the backpropagation algorithm.

Compared to a shape-based neural network a pixel-based network uses more of the input data to classify the images, but on the other hand also requires a considerably larger set of neurons to capture all this information.

**Mahalanobis distance classifier**

This classifier uses a set of shape features, similar to the ones in Table F.1. One additional assumption is made: for every type of input images (e.g. all human images), the values for a particular feature are considered to be drawn from a Gaussian distribution with certain mean and variance. In other words, the values fluctuate around a central value.
Say $k$ features are determined from every binary input image. Then a $k$-dimensional space can be constructed where each dimension (i.e. axis) corresponds to a particular feature. For every classification type a set of training images is collected and for each of these images an associated feature vector is calculated. This vector corresponds to a point in the constructed $k$-dimensional space.

If the features are carefully chosen and the set of training images is representative enough to distinguish each classification type, clearly separated clusters will arise in the feature space. Each of these clusters correspond to a classification type.

When a new binary image is supplied to this classifier it first gets transformed into a $k$-dimensional feature vector, representing a point in the feature space. Subsequently the distance from this point to every cluster is determined, using the Mahalanobis distance $[23, 12]$. This distance measure takes both the mean and variance of the clusters into account. Finally, the type of the cluster yielding the shortest distance is assigned to the new point if the distance is lower than some predefined threshold.

An example is given in Figure F.7. Here only two features are calculated, so $k = 2$. The variance of the clusters is visualized using ellipses which become increasingly more transparent when the points move away from the mean of the cluster. As can clearly be observed, the red cluster has a significant larger variance compared to the variance of the blue and green clusters. The effect of using the Mahalanobis distance measure to classify points instead of the ordinary Euclidean distance becomes clear when looking at point $P$ in the figure. Although the Euclidean distance from $P$ to the green cluster is much smaller than to the red cluster, the Mahalanobis distance from $P$ to the red clusters is the smallest and therefore $P$ is assigned to the red cluster.

**Comparison and discussion**

This Section compares the various strengths and limitations of the four techniques discussed in the previous section. The comparison looks at five characteristics: completeness, compactness, time efficiency, controllability and creativity. *Completeness* indicates whether the method makes actual use of all the provided input data in the classification process. The *compactness* is a characteristic
Figure F.7: a 2-dimensional feature space with three normally distributed clusters. The red cluster has a significantly larger variance compared to the variance of the blue and green clusters. The Mahalanobis distance classifier assigns point P to the red cluster (Original image: [8]).

for the total memory that is occupied by the method. The time efficiency property looks at the overall running time of the method. Controllability is the extend and ease to which the parameters of the technique can be adjusted and fine-tuned in order to reach an optimal performance level. Finally, creativity is the degree to which one has to be creative and inventive during the process of selecting parameters and/or descriptors in order for the technique to perform well.

**Silhouette-based classifier**

Completeness: Silhouette-based descriptors are able to capture the complete shape of the objects. The classifier uses all the shape information and can therefore reason about the integral human posture. A limitation that is related to this issue is the under- and oversampling that is required to create descriptors of equal length. This inevitable causes some information to get lost, proportional to the sampling size.

Compactness: The memory usage of silhouette-based classifiers is not highly efficient. In order for the classifier to perform reasonably its training database should contain a large number of representative human descriptors. For applications with strict memory constraints the size of the database may exceed the limits, making the technique ill-suited for that application.

As a side note, it should be noted that several algorithms have been developed to reduce the memory burden of this technique. One of these uses a dimensionality reduction technique called principal component analysis (PCA) to extract from the large, high-dimensional distance signal a small set of characteristic values, or principal components. These components are then used to compare different signals. A detailed description of this method is given in [37]. More information on PCA and dimensionality reduction in general can be found in [34].
**Time efficiency:** In terms of overall running time the technique does not score very high. For each new incoming frame the distance from the centroid to every boundary point has to be calculated. So the time required to calculate the descriptor is directly proportional with the size of the object boundary. For large images this may take awhile, making the technique unsuitable for real-time applications.

**Controllability:** The technique scores relatively low on controllability. One has only the freedom to change the distance measure that is used to compare two silhouette descriptors. Apart from that there are little or no parameters that can be changed or adjusted.

**Creativity:** One has not to be inventive to use this technique because all parts are uniquely and unambiguously defined.

**Shape-based neural network**

**Completeness:** In contrast to the previous silhouette-based classifier a shape-based classifier only reasons about a finite number of image features. The information loss that is accompanied with this representation is the main drawback of this technique. Choosing image features that are able to capture the differences between all the object types of the classifier is crucial for attaining high performance. It is, however, very difficult to guarantee a set of features will not result in large information loss, in light of the purpose of classification. One can only try to test the system with sufficient training data in order to convince himself the features are representative enough.

**Compactness:** The strength of this technique lies in the compactness and efficiency of the representation: every image is characterized completely by a small set of numbers.

**Time efficiency:** The running time of the method depends on the time required to calculate the shape features and the time required by the neural network to classify a new input vector. Usually the shape descriptors are all simple values that do not require a lot of computation time. Typical neural networks also often have high classification speeds. So overall, this technique can be considered reasonably time-efficient.

**Controllability:** Neural networks are often criticized of being a black-box model. It is very difficult to control or adjust the behavior of a trained neural network. The effect of changing one of the weights of the synapses is very hard, if not impossible, to predict. One can only train the network extensively in order to guarantee a good performance.

**Creativity:** One of the advantages of using a neural network is the absence of having to explicitly come up with reasoning or classification rules for solving the problem in question. One only needs to find an accurate input representation for the problem instance and provide the network with sufficient training examples. Throughout the training process the network will itself try to find the right structure in the data as it tries to minimize the classification error.

**Pixel-based neural network classifier**

**Completeness:** The advantage of a pixel-based neural network over the previous network is that more information of the image can be captured: where the previous network only had the ability to reason about a fixed number of image properties, this network takes into account every pixel of the input image. This makes the technique more flexible.

**Compactness:** Because the classifier uses the raw image data, without any feature extraction,
Table F.3: overview of the comparison between the four classification techniques

<table>
<thead>
<tr>
<th></th>
<th>Silhouette</th>
<th>Shape NN</th>
<th>Pixel NN</th>
<th>Mahalanobis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Completeness</strong></td>
<td>⊕⊕</td>
<td>⊖</td>
<td>⊕⊕</td>
<td>⊖</td>
</tr>
<tr>
<td><strong>Compactness</strong></td>
<td>⊖</td>
<td>⊕</td>
<td>⊖⊖</td>
<td>⊖</td>
</tr>
<tr>
<td><strong>Time efficiency</strong></td>
<td>⊖</td>
<td>⊕</td>
<td>⊖⊕</td>
<td>⊖</td>
</tr>
<tr>
<td><strong>Controllability</strong></td>
<td>⊖</td>
<td>⊖⊖</td>
<td>⊖⊖</td>
<td>⊖⊕</td>
</tr>
<tr>
<td><strong>Creativity</strong></td>
<td>⊖</td>
<td>⊖</td>
<td>⊖⊖</td>
<td>⊖</td>
</tr>
</tbody>
</table>

the technique does not score very high on compactness.

*Time efficiency, controllability and creativity:* At these points the technique does not differ with the previously described shape-based neural network.

**Mahalanobis distance classifier**

*Completeness and compactness:* In terms of completeness and compactness the statements from the previous section, on the strengths and limitations of using image features to represent and reason about the blob images, also apply here. In addition, one drawback is the fact that the classifier only considers the first two order moments (i.e. mean and variance) of the data. Some problems might require higher order moments to be also taken into consideration.

*Time efficiency:* Once the feature space is constructed it is very easy and fast to calculate the mahalanobis distance from a new feature point to every constructed cluster in the feature space. This makes the technique very suitable for real-time applications.

*Controllability:* An advantage of the Mahalanobis distance classifier is the controllability. The effects of changing the mean or variance, adding new features or adapting the distance measure are all easy to predict.

*Creativity:* One only needs some creativity in finding an accurate set of features that provide a good input representation. The rest is all clearly defined without much selection freedom.

**Discussion**

From the foregoing Sections it follows that every technique has its own set of advantages and limitations. Table F.3 gives an overview of the strengths and limitations of the four techniques, in light of the five predetermined characteristics.

From the table it becomes clear that it not easy to select one technique as the overall best for every situation. The choice of algorithm is strongly related to the context of the problem and application. Each application has its own set of constraints and requires a technique that best fits these requirements. Even in the case of human classification it is not directly clear which technique should be used. Applications where realtime operation is important, like camera surveillance systems, might benefit from one of the neural networks or the mahalanobis classifier. Critical systems in which a high level of control is essential, on the contrary, will not use a neural network but
might rather be interested in using the mahalanobis classifier. There might be applications that will need to be able to cope with changing light conditions, like outdoor security systems, while others may have different constraints on the human types that can be distinguished (e.g. distinguishing men, women and children v.s. only differentiating between human and non-human). Only when the scope and domain of the application in question are well-specified, one can start considering the different techniques and compare their qualities to reach a verdict on which technique should eventually be used.

Conclusion

This appendix focussed on providing an overview on the most important solutions to the problem of object classification, in the contest of human classification. Four techniques were discussed and compared: a silhouette-based classifier, a shape-based neural network, a pixel-base neural network and a Mahalanobis distance classifier. The first, second and last techniques are all feature-based, the third is pixel-based. Feature-based techniques first calculate one or more characteristic values out of the given input image and subsequently give these as input to a specific classifier, which returns the type of the object. Pixel-based techniques do not calculate any features in advance but instead base their classification directly on the pixels of the input image. Each method approached the problem from a different perspective. The comparison looked at five characteristics: completeness, compactness, time efficiency, controllability and creativity. The comparison showed that it is not directly possible to declare one technique as overall winner for every situation. The context of the problem and the constraints of the application in question thus greatly influence the choice on which technique should be applied.
Intermediate Feedback

[Aanbevelingen] De code van het systeem scoort net vijf sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code volgens onze score zeer goed onderhoudbaar is. De twee punten waarop een lagere score wordt behaald zijn Unit Interfacing en Unit Size.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. In dit systeem is de score met name lager aan de server-kant van de applicatie. Binnen de Java-code lijkt er hechte relatie te zijn tussen verschillende objecten zoals 'MotionBox', 'Camera' en 'Frame', deze objecten worden dan ook op verschillende plekken samen doorgegeven. De relatie tussen deze objecten is echter niet geheel duidelijk vanuit de code, het is aan te raden om te kijken of de relatie tussen deze objecten duidelijk te maken is binnen de code-base. Dit maakt het voor toekomstige programmeurs makkelijker om de code te begrijpen en te onderhouden.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de constructor van het 'Camera' object in de client-code, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '//Get the height of the camera above ground' en '//Get mac adress' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Over het algemeen scoort de code zeer goed op onze score voor onderhoudbaarheid, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code voor het Java-gedeelte is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt. Daarnaast is het ook aan te raden om ook de belangrijke onderdelen van de C++ code automatisch te testen om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Final Feedback

[Hermeting] In de tweede upload zien we dat de omvang van het systeem is gestegen, maar dat daarbij de score voor onderhoudbaarheid zeer licht is gedaald. Op zowel het gebied van Unit Interfacing als op het gebied van Unit Size zien we in deze upload een verbetering. Op het gebied
van Module Coupling, het percentage van de code wat relatief vaak wordt aangeroepen, zien we een daling van de score. Met name aan de Server-kant zijn er enkele grotere objecten die relatief vaak worden aangeroepen. Verder zien we dat aan zowel de Server- als aan de Client-kant de hoeveelheid test-code is toegenomen.

Uit deze observaties en de begeleidende documentatie kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Het is goed om te zien dat ondanks de forse groei van het systeem (met 59%) het hoge niveau van onderhoudbaarheid is behouden.