Master's Thesis
Multi-camera video surveillance system

Maarten Somhorst

January 20, 2012

Ministerie van Defensie

TU Delft Delft University of Technology

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science

Multi-camera video surveillance system

by Maarten Somhorst

Abstract

Since the stone age the human race seeks for strategies to extend its viewing range. With the rise of technology in the twentieth century, cameras are found to be a very useful tool to survey a large area with limited resources. With an increasing numbers of cameras, it becomes more difficult to watch every monitor and prevent incidents in the surveillance area. For the last decades, research seeks for possibilities to automatize the process of video surveillance.

For this thesis, we approach the surveillance task from the human perspective: we try to emulate what human operators do when they watch the monitors. To perform this task, state-of-the-art techniques from Computer Vision and Artificial Intelligence are applied. An object tracking technique called *P-N Learning* is used that enables the tracker to learn from its mistakes. The *Java Agent Development Framework (JADE)* is used to enable communication between agents in the *FIPA Agent Communication Language* standard.

A surveillance system model is designed that detects suspicious behavior in a non-public area. Its task is to alert the operators about suspicious events to give them the chance to investigate and take action. Two prototype applications are implemented and experiments are conducted to show the performance.

We showed the proof-of-concept of a system which is able to emulate operators and can potentially outperform a human being. Once the system knows what is considered suspicious behavior it can be automatically detected.

Master's thesis
Name: Maarten Somhorst
Study number: 1263196
Programme: Media and Knowledge Engineering
Graduation date: January 20, 2012

Committee members
Prof. Drs. Dr. L.J.M. Rothkrantz
Dr. ir. P. Wiggers
Ir H.J.A.M. Geers
Ir. I. Lefter

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Preface

This thesis is the result of my graduate project that finalizes the master programme Media and Knowledge Engineering. The project is titled *Multi-camera video surveillance system* and is performed on the University of Technology Delft, The Netherlands. The enthusiasm of Prof. L. Rothkrantz about video surveillance quickly infected me. Especially the reasoning and multi-agent aspects of the idea sounded interesting to me.

I would like to thank prof. L. Rothkrantz and ir. I. Lefter for their help during this graduation project. They have offered me a lot of time, experience, knowledge and creativity, which helped me through the project. Furthermore, I would like to thank my wife for her support and kind assistance.

Delft, January 14, 2012
Maarten Somhorst

Contents

1 Introduction

1.1 Problem Description: Surveillance

Video surveillance systems traditionally consist of cameras attached to monitor screens. These systems are installed to give an overview of a large area to a limited number of operators. The goal is to detect abnormal situations. Depending on the seriousness of the situation, action can be taken.

Operators often work in a room with lots of monitors like illustrated in figure 1.1. Their task is to watch constantly the monitors. If incidents happen, they warn the security or police. Some monitors show the video stream of a single camera and some show multiple streams on a single monitor simultaneously or sequentially.

However, in some areas the monitors are not watched constantly. Video recorders record the output of each camera. After an incident, the video footage can be used as evidence. One obvious disadvantage of this approach is that operators are not able to prevent incidents or limit their damage, since the videos are only watched afterwards. Another disadvantage is that it takes a significant amount of time to search for the right video images, especially when the suspect arrives at the scene hours before the incident and a large amount of cameras are involved.

1.2 Problem Description: Human Observers

The task of the human operator is not limited to watch the videos and react to abnormal situations. After training and acquiring experience, a human operator is able to incorporate the area context in order to judge the events on the monitor. Take for example a car that stops near a weapon depot. The event of a car stopping might not be abnormal, but the fact that it happens near

Figure 1.1: Control room.

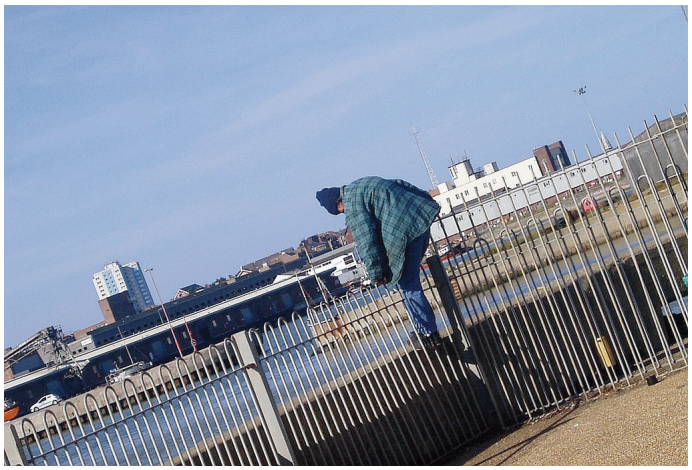

Figure 1.2: An example of a suspicious act.

a weapon depot might trigger the operator to give it some more attention. Building upon this example, an operator has a set of Regions of Interest (RIOs), that we define as: certain regions in the surveillance area that are potential targets of incidents. Examples of ROIs are parking lots, ATMs, the road to a nuclear reactor, and so on. For each of such ROIs, the operator has a list of what is normal and suspicious behavior. For example, it is normal if a car stops near an ATM and someone steps out. However, if a truck hits the ATM with a high speed, a robbery might be going on. Another example of an ROI is a fence: if people climb over a fence such as illustrated in figure 1.2, action should be taken. Besides incorporating context, human beings are very good at recognizing moving objects [4]. Even small motions on a large screen are not a problem for humans.

However, humans do make mistakes. Even more, it appears that they make a significant amount of mistakes when they are watching to surveillance monitors [8]. The main reason is the nature of the task: passively watching multiple monitor screens where nothing special happens for a long period of time. Research has been done to help the operators with the task to keep an overview [6]. However, if we could ease the task of the human operator by watching the monitor screens for him, the chance of preventing incidents is expected to increase significantly. Another problem is that hiring people is expensive, so the monitors are watched only when necessary. In contrast, computers can work for 24 hours a day and 7 days a week without large expenses.

1.3 Taken Approach

For this thesis, we approach the surveillance task from the human perspective: we try to emulate what human operators do when they watch the monitors. These activities include the following:

- Detecting objects: is something moving?
- Recognizing objects: is it a person or a car?
- Tracking objects: where is it going? which path does it take? at which speed?
- Reasoning about available data: is this object entering a forbidden area? is this car parking on a non-parking area? is this suspicious behavior?

To perform this task, state-of-the-art techniques from Computer Vision and Artificial Intelligence are applied. An object tracking technique called *P-N Learning* is used that enables the tracker to learn from its mistakes. The *Java Agent Development Framework (JADE)* is used to enable communication between agents in the *FIPA Agent Communication Language* standard.

A surveillance system model is designed that detects suspicious behavior in a non-public area. Its task is to alert the operators about suspicious events to give them the chance to investigate and take action. We focus ourselves on an area from the Nederlands Defensie Academie (NLDA) in Den Helder, The Netherlands. It is a military area surrounded by fences and authorized people can enter the area by foot or car through gates using an access card. With small adjustments, the proposed system is also applicable for other kinds of areas.

1.4 Relevance

Since the stone age the human race seeks for strategies to extend its viewing range [24]. With the rise of technology in the twentieth century, cameras are found to be a very useful tool to survey a large area with limited resources. Valera and Velastin [25] state that "Recent events, including major terrorist attacks, have led to an increased demand for security in society". Technological development provides cheaper cameras and surveillance systems. These factors cause the number of surveillance cameras to increase significantly. If we could ease the task of the human operator, or in some cases replace him, the chance of preventing incidents is expected to increase significantly [8].

Besides the societal relevance, this thesis is also scientifically relevant. In the project both Computer Vision and Artificial Intelligence are combined into one system model. Second, part of the model is implemented which shows the feasibility of a practical realization. Third, the produced software is build in such a way that it is reusable for future research.

Potentially there are many advantages in using computers instead of human operators. For one, computers are very scalable in terms of time. People tend to forget information that is not used regularly. In the case of surveillance, this means that a human observer might forget that he saw a certain car earlier today. In contrast, computers can be equipped with enough disk space to remember this kind of information for weeks or months. Second, computers are scalable in terms of the number of objects. Human observers might be able to understand and remember the interaction between ten or twenty objects that are currently in the area, but when a hundred or two thousand objects need to be monitored, people are not capable doing that. Third, because of the human limitations, multiple human operators are watching the monitors, but exchanging information is difficult. Computers might not be able to process more video streams than humans, but they can communicate constantly about what they see. Note that computers are not necessarily scalable to monitor any number of objects: there are limits to the processing power and disk space for a single unit. However, computers can be easily extended, whereas hiring more human operators make knowledge sharing even more difficult.

1.5 Methodology

The main goal of this graduation project is to design an automatic suspicious behavior detection model using multiple cameras and implement part of it. The following two research questions form the basis of this goal.

Question 1: How can we use a video surveillance camera to replace the human eye?

Question 2: If reasoning is applied to the data from multiple cameras, to what extend can we automatically detect suspicious behavior?

To answer these research questions, the following subgoals are identified and used as a basis for this graduation project:

- Perform a short literature study on automatic video surveillance to find out what the state-of-the-art is.
- Design a model of a system that is able to automatically detect suspicious behavior.
- Implement the video processing of a single video stream.
- Implement combining the information of multiple cameras to detect suspicious behavior.
- Perform experiments on both implementations and analyze the results.
- Give recommendations for future research.

During this graduation project two prototypes are implemented. The first prototype focuses on the processing of a single video stream. The processing of a single camera is the basis for each multi-camera surveillance system. The processing includes object detection, object classification and object tracking. The second prototype focuses on the reasoning by a central entity. Information from multiple cameras are combined to reason about events occurring in the surveillance area. This method of software development is called Software Prototyping: incomplete applications are created that shows the main purpose of the proposed system.

1.6 Outline

This report describes the graduation project that investigates multi-camera video surveillance systems. Chapter 2 includes a summary of research done in the area of automatic video surveillance. Chapter 3 sums up the requirements of the model, while chapter 4 elaborates on the design of the system model. Chapter 5 describes the resources that are used for the implementation and their background theory. Chapter 6 shows how we used these tools in our implementation. Chapter 7 describes the design of the agents that are part of the model. The implementation is described in chapter 8. Chapter 9 shows the performed experiments and their results. Finally, chapter 10 contains the conclusion and recommendation for future work.

2 Related Work

The field of video surveillance is very broad. Active research is going on in subjects like face recognition, 3D object modeling, multi-camera camera setups and human behavior analysis. This chapter contains summaries of several papers that describe fields which are close to the subject of this thesis. The set of papers include survey papers and papers that describe a particular system. For every paper it is also mentioned why the paper is chosen and our personal opinion is given.

The rest of this report includes a number of other papers that are applied in this thesis. They are not summarized in this chapter.

2.1 Smart Video Surveillance: Exploring the concept of multiscale spatiotemporal tracking - Hampapur et al (2005)

Motivation This article is chosen because it seems to give a good overview of tracking in the context of surveillance.

Summary This article [9] gives an overview of what aspects are important for large-scale video surveillance systems. The challenges that are pointed out are: combining multiple sources of information, automatic event detection and deploying systems with a large number of cameras (cost-wise). The article also shows some basic techniques like object detection, 2-D and 3-D object tracking, object classification and object structure analysis (specifically: head detection). Moreover, the structure of two designed systems are presented:

- Face cataloger, which aims at doing high-resolution face detection using 3-D head detection and a pan tilt zoom (PTZ) camera at the entrance of a restricted room.
- A system for long-term monitoring, which saves the surveillance video fragments in a database in such a way that it can be queried later on. During this saving process, the video fragments are analyzed by the *Smart Surveillance Engine* which detects, tracks and classifies the objects. Examples of queries: "give all vehicles that drove through *this* particular part of the camera sight" and "show all vehicles that drove to the east".

The last section includes a discussion of the errors such systems can make and how these systems can be evaluated.

Personal Notes This article gives a good overview of the field of video surveillance. It makes clear what are the basic techniques and mention different kind of implementations while not going into detail. Also, it mentions several issues that have to be faced while designing a video surveillance system.

2.2 A multimodal car driver surveillance system in a military area - Lefter et al (2010)

This paper [15] describes the surveillance system that the authors envision for the NLDA area in Den Helder, The Netherlands. The system uses both video and audio signals to secure the area. It consists of two subsystems: a car-driver identity recognition system at the entrance of the restricted area and a car-driving behavior recognition system along the roads in the area. The paper describes how they analyze the behavior of the drivers, fuse the video and audio data and use an expert system to solve conflicts between audio and video. At the end they describe several scenarios that help them developing this expert system.

Personal Notes The thesis project originated from the ideas in this paper. Therefore, this paper gives a good idea of the context of this thesis.

2.3 TRECVID 2009 - Goals, Tasks, Data, Evaluation Mechanisms and Metrics - Over et al (2009)

Motivation The TREC Video Retrieval Evaluation (TRECVID) is a conference series that started in 2001. The goal is "to encourage research in information retrieval by providing a large test collection, uniform scoring procedures, and a forum for organizations interested in comparing their results" . This paper is chosen because TRECVID is often mentioned in video surveillance literature and it is expected that we can use this paper to find other papers with good performing surveillance systems.

Summary This paper [19] gives an overview of the TREC Video Retrieval Evaluation (TRECVID) 2009. The goal of this evaluation was "to promote progress in content-based exploitation of digital video via open, metrics-based evaluation". In 2009, sixty three research teams submitted a video recognition system. There were four tasks: high-level feature extraction, search (fully automatic, manually assisted, or interactive), copy detection and surveillance event detection. Each system had to be able to do at least one of these tasks. During the evaluation, the submitted systems are tested and the results are presented in this paper. For our graduation project, the last task is interesting: surveillance event detection. The goal of this task is to recognize particular visual events of people. Ten types of events were specified. The data that is used consists of multiple synchronized camera views.

Personal Notes The most important page of this paper was the one with the Surveillance Event Detection task result graph. This was the only way to see what implementation did well during TRECVID 2009. The bar diagram was not very handy, because it is not made clear what parties did well overall and which did not. The diagram showed per event how well each party did, but each party was able to detect a different set of events. Nevertheless, we picked two papers on basis of this diagram. Because the Toshiba team is responsible for the best run, their paper is summarized next. Also, the paper of the Shanghai Jiao Tong University is reviewed below. They did well for the events *TakePicture* and *PersonRuns*, which might be useful for our project.

2.4 Toshiba at TRECVID 2009: Surveillance Event Detection Task - Yokoi et al (2009)

Motivation This paper is chosen because it is responsible for the best run in the TRECVID 2009 conference.

Summary This paper [31] presents the system Toshiba provided to the TRECVID 2009. It explains how the following four components are implemented:

- Change detection.
- Human detection.
- Human tracking.
- Event detection.

Their goal was to detect three events: PersonRuns, ElevatorNoEntr and OpposingFlow. PersonRuns is the event of a running person. A system that detects an ElevatorNoEntr event, sees a person waiting for an elevator, but he does not enter it when it arrives. The last event, OpposingFlow, is described as *"someone moves through a controlled access door opposite to the normal flow of traffic"*.

Personal Notes This is a detailed and mathematical paper. The components are well explained and very straightforward. However, there is not enough information to reproduce each of the components.

2.5 Shanghai Jiao Tong University participation in high-level feature extraction and surveillance event detection at TRECVID 2009 - X. Yang (2009)

Motivation This paper is chosen because the team manages to perform well at the events *TakePicture* and *PersonRuns*.

Summary This paper [30] describes the system that the authors made for the TRECVID 2009 conference. They "explored several novel technologies to help detecting high-level concepts". There are four main steps in their high-level feature extraction framework:

- Low level feature extraction. Global, local and *other* features are extracted from the input video with a total of eight features.
- Model. Support Vector Machines (SVMs) are used as classifiers. They train one SVM classifier for each low-level feature based on TRECVID 2009 development data.
- Ranking. The results of trained models are combined by *simple average fusion* and *linear weighted fusion*.
- Re-ranking. Textual information is extracted using automatic speech recognition and the information bottle principle.

The global low level features include features that characterize the image based on color, textures, shape or a combination of those. The local low level features are extracted using *Scale-invariant feature transform* (SIFT). *Difference of Gaussian* is used to describe a keyframe as a *bag-of-visual-words*. The main issue here is determining the size of the vocabulary. The *other* feature used is called *Space-Time Interest Points* (STIP). It computes locations and descriptors for space-time interest points in video. Besides these global, local and *other* features, two additional features are used for some of the recognition tasks: extraction of *Region of Interest* (ROI) and the *Face feature*. The ROI extraction aims at locating people's body parts. It uses *Conditional Random Field* and *Pyramid Histogram of Oriented Gradients*. The *Face feature* is created to detect female faces. The first step in *Face feature* extraction is to detect a human face on the image. The second step is implemented by using *Local Binary Pattern* features as input for a SVM classifier in order to determine if a face is male or female.

The rest of this paper shortly describes how they detect each of the eight events they can detect. At the end the experimental results are shown in the form of a table and a *Detection Error Tradeoff* curve plot.

Personal Notes The paper describes the methods and techniques they used to build their system and detect the events, but they do not give enough information to reproduce it.

2.6 A Survey on Visual Surveillance of Object Motion and Behaviors - Hu et al (2004)

Motivation This survey is chosen because it appears as a extensive survey of the video surveillance research field. Also the topic of *object motions and behaviors* promises useful information for this thesis.

Summary This survey [10] gives an overview of the developments in the field of visual surveillance in dynamic scenes. The focus is on applications involving surveillance of people or vehicles. Five stages are identified that are common for most surveillance systems. The following subsections summarize the topics discussed per stage.

2.6.1 Motion detection

This stage is divided into three processes: environment modeling, motion segmentation and object classification.

A Environment modeling.
Environmental models can be classified into 2-D and 3-D models. Several problems are discussed while using fixed, pure translation (PT) cameras and mobile cameras for 2-D. Current work on 3-D models is still limited to indoor applications.

B Motion segmentation.
The following approaches for motion segmentation are explained:

- Background subtraction.
- Temporal differencing.
- Optical flow.

C Object classification.
Two main categories of approaches are mentioned:

- Shape-based classification.
- Motion-based classification.

2.6.2 Object tracking

Tracking algorithms are divided into four main categories:

A Region-based tracking.
An object is tracked according to variations of the image regions corresponding to the moving objects.

B Active contour-based tracking.
These algorithms track objects by representing their outlines as bounding contours and updating these contours dynamically in successive frames.

C Feature-based tracking.
A distinction is made between global feature-based algorithms, local feature-based algorithms and dependence-graph-based algorithms.

D Model-based tracking.
The following three main issues are extensively described:

- Construction of human body models.
- Representation of prior knowledge of motion models and motion constraints.
- Prediction and search strategies.

2.6.3 Understanding and description of behaviors

Behavior understanding involves the analysis and recognition of motion patterns and the production of high-level descriptions of actions and interactions.

A Behavior understanding.
The following major existing methods for behavior understanding are described:

- Dynamic time warping.
- Finite-state machine.
- HMMs.
- Time-delay neural network.
- Syntactic techniques.
- Non-deterministic finite automaton.
- Self-organizing neural network.

B Natural language description of behaviors.
Two main categories of behavior description methods are explained: statistical models and formalized reasoning.

2.6.4 Personal identification for visual surveillance

Regarding visual surveillance it is important to identify people. Human face and gait detection is used to accomplish that. The paper describes the following major methods for gait recognition in recent research:

- Model-based methods.
- Statistical methods.
- Physical-parameter-based methods.
- Spatio-temporal motion-based methods.
- Fusion of gait with other biometrics.

2.6.5 Fusion of data from multiple cameras

This last stage is described by pointing out the following problems regarding multicamera surveillance:

- Camera installation.
- Camera calibration.
- Object matching.
- Automated camera switching.
- Data fusion.

The last part of this survey paper mentions subjects that deserve future research.

Personal Notes This survey gives a nice overview of the field of video surveillance. It mentions many basic techniques and discusses how several papers used these techniques. It makes clear which issues arise when building multicamera surveillance systems. Furthermore, it gives insight in what areas are well developed and which are not.

2.7 Survey on Contemporary Remote Surveillance Systems for Public Safety - T. Räty (2010)

Motivation This paper is summarized because it gives an overview of the history of video surveillance and mentions the current topics in research.

Summary This paper [22] reviews the historical development of video surveillance and describes the three generations of surveillance systems. The focus is on generic surveillance, which is applicable to public safety.

Public safety and homeland security are substantial concerns for governments worldwide. Recent events like terrorist attacks have increased the demand for security. Therefore, there is a growing interest for surveillance applications. The progress in technological development enable

deploying surveillance systems at reasonable costs. Due to the growing number of cameras, surveillance by humans become unfeasible and automatic intelligent surveillance systems are needed.

"A surveillance system can be defined as a technological tool that assists humans by offering an extended perception and reasoning capability about situations of interest that occur in the monitored environments. Human perception and reasoning are restricted by the capabilities and limits of human senses and mind to simultaneously collect, process, and store limited amount of data."

2.7.1 First-generation surveillance systems

The 1GSSs (1960-1980) uses analog technology to monitor the environment. Video data from a collection of cameras were being transported to a control room where human operators observed the environment using TV screens. The disadvantages of these systems are: significant miss rate of events of interest due to the small attention span of operators and the difficulties of analog video communication (high-bandwidth requirements, poor allocation flexibility and inefficient image distribution and storage).

2.7.2 Second-generation surveillance systems

2GSSs (1980-2000) has benefits over 1GSSs, as 2GSSs use techniques of the early progression of digital video communications. "The majority of research efforts during the period of the 2GSSs have been used in the development of automated real-time event detection techniques for video surveillance. The availability of automated methods would significantly ease the monitoring of large sites with multiple cameras as the automated event detection enables prefiltering and the presentation of the main events."

2.7.3 Third-generation surveillance systems

In 3GSSs (2000-present), the current generation, only digital components are used. Furthermore, it focuses on handling large number of cameras to cover large areas, distribution of processing capabilities and combining different types of sensors (microphones, thermal cameras, etc). The main objective is to ease efficient data communication, management and extraction of events in real-time video from a large collection of sensors. "The main application areas for the 3GSSs are in the region of public monitoring. This is required by the rapid growth of metropolitan localities and by the increasing need to offer enhanced safety and security to the general public."

The rest of this subsection describes the aspects of 3GSSs that the author finds notable.

Multiple sensor-enabled environments "Recently, there have been studies on data fusion techniques to tolerate information sharing that results from different types of sensors." Examples of sensors are: microphones, thermal cameras, radar, automatic identification systems and global position system receivers.

Junejo et al [12] state that a single camera is not sufficient in monitoring a large area and propose a practical framework for an automatically configurable network of non-overlapping camera that provides sufficient monitoring capabilities.

Video surveillance The primary goals of intelligent visual surveillance systems are to offer an automatic interpretation of scenes and to understand and predict the actions and interactions of the observed objects.

"The automation of all or part of the process of manual monitoring would obviously offer dramatic benefits, ranging from a capability to alert an operator of potential event of interest, to a completely automatic detection and analysis system. However, the dependability of automated detection systems is an essential issue, because frequent false alarms introduce skepticism in the operators, who quickly learn to disregard the system."

"Research interests have shifted from ordinary static image-based analysis to video-based dynamic monitoring and analysis. Researchers have advanced in addressing illumination, color, background, and perspective static aspects. They have advanced in tracking and analyzing shapes related to moving human bodies and moving cameras. They have improved activity analysis and control of multicamera systems."

"When multiple tracked objects are placed into groups with miscellaneous complexities of occlusion, tracking each individual object through crowds becomes a challenging task."

"Li et al. [17] state that the aim of multitarget tracking is to infer the target trajectories from image observations in a video. This poses a significant challenge in crowded environments where there are frequent occlusions and multiple targets have a similar appearance and intersecting trajectories."

"Basically, the approach of the detection of moving objects is through background subtraction that contains the model of the background and the detection of moving objects from those that differ from such a model. In comparison to other approaches, such as optical flow, this approach is computationally affordable for real-time applications. The main dilemma is its sensitivity to dynamic scene challenges and the subsequent need for background model adaptation through background maintenance. This type of a problem is known to be essential and demanding."

"A significant problem encountered in numerous surveillance systems are the changes in ambient light, particularly in an outdoor environment, where the lighting conditions varies. Thermal cameras have been utilized for imaging objects in the dark."

Audio Surveillance Several methods are presented for sound localization, detection, classification and tracking in sensor networks. Event detection systems have been built that solely use microphones.

Wireless sensor networks "Sensor networks usually are inexpensive wireless devices that are distribute over a region of interest. They are often battery powered and have restricted computation and communication capabilities. Every node is equipped with different of sensing modalities, such as acoustic, infrared and seismic. Networked sensors can collaborate to process and make deductions from the collected data and provide the user with access to continuous or selective observations of the environment."

Distributed Intelligence and Awareness "The 3GSSs use distributed intelligence functionality. An important design issue is to determine the granularity at which the tasks can be distributed based on available computational resources, network bandwidth, and task requirements."

Four tasks of surveillance are identified and shortly described:

- Event detection.
- Event representation.
- Event recognition.

- Event query.

The important questions in security are: "who are the people and vehicles in a space" (identity tracking), "where are the people in a space" (location tracking), and "what are the people/vehicles/objects in a space doing" (activity tracking) [9].

Bandini and Sartori [1] present the concept of a monitoring and control system (MCS) which aims at helping humans in decision making regarding problems which can occur in critical domains. This is done by gathering data of the monitored situation, detecting if abnormal events happen and acting on it, often by alarming the operator.

Architecture and middleware "The field of automated video surveillance is quite novel and the majority of contemporary approaches are engineered in an ad hoc manner. Recently, researchers have begun to consider architectures for video surveillance. Middleware that provides general support to video surveillance architectures is the logical next step."

Utilization of mobile robots "Currently, the development of a completely automated surveillance system based on mobile multifunctional robots is an active research area. Mobility and multifunctionality are generically adopted to reduce the amount of sensors required to cover a given region."

2.7.4 Discussion on current dilemmas in the 3GSSs

The paper identifies several aspects that are discovered during literature review.

Real-time distributed architecture It is important to develop a framework or methodology for designing wide-area surveillance systems. Furthermore, because of the high complexity of video surveillance networks in general, middleware is needed to reduce that complexity. A distributed multiagent approach maybe provide numerous benefits like:

- A large number of sensors might be deployed over a large area because of the low costs and intelligent cooperation between agents.
- Such a system is robust, because it keeps working if some agents fail.
- Performance is more flexible, because agents can help each other.

A dilemma related with architecture is that vision systems engineers have no systematic way to translate user system requirements to a detailed design. Now the design and analysis phases usually follow each other in a cycle until the system requirements are met.

The advantage of distributed systems over central systems is that there is an improvement in functionality, availability and autonomy of the surveillance systems. The disadvantage is that systems become more complex. One way to deal with this complexity is to decompose systems into component parts and functions.

Difficulties in video surveillance An issue that occur in crowded scenes is occlusion: objects become occluded by buildings, trees or other objects. Tracking and classification is more difficult when occlusion occurs. The use of multiple cameras can improve the performance of tracking and classification. Another method to gain better results is to use a human appearance model.

One dilemma in background subtraction is caused by the detection of false objects called *ghosts*. This occurs for example when a foreground object is stationary for a while and then moves away, which causes the system to label a part of the background as the moving object. This is undesirable and can be detected using a ghost detection algorithm.

Research is being done that focus on automatically extracting information from video data in order to reduce the load on human operators. However, due to computational cost of these systems, the usage in real-time is limited.

Furthermore, effort is done by the computer vision and artificial intelligence community to develop automated systems for monitoring people, vehicle and other objects. The aim is to build systems that detect unusual activity and warn the human operator.

Awareness and intelligence "Context-aware applications are needed to support personalization and adaptation based on context awareness." These applications must be capable of explaining their actions to the user.

There are two types of dilemmas regarding multi-sensor surveillance system: the fusion of data in an optimal matter and the optimization of the global management using the individual sensors.

Wireless networks and their applicability In order to use wireless sensor networks, many technological issues need to be resolved. Real-time surveillance systems generate a lot of data, which can cause network congestion, especially when there is a small amount of bandwidth available. In order to have a system that can cope with that, error control mechanisms need to be implemented.

Energy efficiency of remote sensors "Novel solutions are needed to handle demanding restrictions of video surveillance systems, both in terms of communication bandwidth and computing power."

Dilemmas in scalability "Implementing an intelligent, scalable, and distributed video surveillance system remains a research problem. Researchers have not paid too much attention on the scalability of video surveillance systems. They typically utilize a centralized architecture and assume the availability of all the required system resources, such as computational power and network bandwidth."

"A substantial pitfall in incorporating intelligent functions into real-world systems is the lack of robustness, the inability to test and validate these systems under a variety of use cases, and the lack of quantification of the performance of the system."

Location difficulties "Complementary information in the form of maps and live video streaming can assist in locating the problematic zone and act quickly and with knowledge of the situation."

Challenges in privacy Video surveillance opposes privacy problems. As many system nowadays transmits networks open public networks, information protection is an important issue. Research has been done to protect the privacy of people that are monitored by removing personally identifiable information.

Personal Notes A nice feature of this paper is that it identifies distinct generations in the development of video surveillance. However, the differences between them are not made very clear and the first two generations are not discussed extensively. For the third generation, many useful aspects are explained. In our opinion, the downside of this number of aspects is that the paper is not very coherent. The identification of the 3GGSs' aspects suggests that the author presents the material very structured, but the content of those aspects often seem to have nothing in common. A lot of research is mentioned, but they are not linked to each other. Nevertheless, many interesting subjects in the field of video surveillance are presented which gives a relatively good overview of this field.

2.8 Real-time Crowd Motion Analysis - N. Ihaddadene (2008)

Motivation This paper is chosen because it tries to detect abnormal behavior.

Summary This paper [11] describes work that has been performed for the MIAUCE project. This project "aims to investigate and develop techniques to analyze the multi-modal behavior of users within the context of real applications"[7]. A part of this project has made an attempt to automatically detect abnormal behavior at the exits of airport escalators using video cameras, which is described in this paper. The aim is to automatically detect congestions at the escalator exits.

The algorithm that is proposed has four steps:

1. Motion heat map creation.
   A motion heat map is used to identify the regions on the camera image with a high motion activity. It is a 2D histogram that is built from the accumulation of binary blobs of moving objects, which are extracted using background subtraction.

2. Features detection and tracking.
   During this step, a set of points of interest (POIs) are extracted from each input frame. These points are tracked over the next frames using optical flow techniques. This results in a set of vectors which contain the following properties of each feature: the X and Y coordinates, the motion direction and the feature distance between of two consecutive frames.

3. Direction map building.
   For this step, a direction map is built which indicates the average motion direction for each region of the camera image. This average motion detection is visualized by a vector for every region.

4. Abnormality estimation.
   A measure is defined that "describes how much the optical flow vectors are organized or cluttered in the frame". It is the scalar product of the normalized values of the following four factors: motion area ratio, direction variance, motion magnitude variance and direction histogram peaks. The resulting measure is compared to a specific threshold to determine if the current situation is normal or abnormal. This threshold needs to be configured, because it varies depending on camera position, escalator type and position.

The authors state that the results are *very satisfactory*, without mentioning them explicitly.

Personal Notes This paper clearly describes the steps that the authors took to detect abnormal behavior. Although the steps are not described in detail, it looks like it is feasible to reproduce the algorithm. Unfortunately, no experiments are shown that show the performance. The paper gives the impression that it is a good algorithm to detect abnormal movements (like congestion), but since it is only applied to escalators, it might not be applicable directly on another domain like detecting abnormal behavior on roads.

2.9 Tracking many objects with many sensors - H. Pasula, S. Russell, M. Ostland and Y. Ritov (1999)

Motivation This paper was referenced by the book *Artificial Intelligence: A Modern Approach* by S. Russell and P. Norvig which described the possibility of tracking highway traffic using multiple cameras. The multi-camera aspect is interesting, because it is a challenge how to track one object using several non-overlapping cameras and it is an important aspect of this thesis.

Summary This paper [20] builds on previous work from Huang and Russel which "provided a probabilistic analysis and a threshold-based approximation algorithm for the case of multiple objects detected by two spatially separated sensors". This paper focuses on the case of many sensors. For more than two observations, the objects' *intrinsic* properties needs to be measured. Also, this paper replaces "Huang and Russell's threshold algorithm for object identification with a polynomial-time approximation scheme based on a Markov Chain Monte Carlo (MCMC) simulation".

Two experiments are conducted using data from a freeway simulator. The first compares the algorithm with the one from Huang and Russel. The network in this experiment contains three cameras. The main conclusion here is that the previous algorithm cannot handle color noise, while the current algorithm handles it very well. The second experiment deals with a more realistic scenario with nine cameras that secure an area. "The aim is to estimate the origin-destination counts between the two entry points and the three exit points". The MCMC algorithm outperforms that Huang-Russell algorithm. However, both algorithms are unable to accurately determine the full trajectories for high levels of noise.

The authors mention in the *Summary and Future Work* section that they are working towards applying the approach in the real world such as traffic surveillance.

Personal Notes The approach in this paper looks very promising, but is not ready to use at all. Firstly, the algorithm still lacks the ability to determine the full trajectories with high levels of noise. Secondly, the algorithm is only applied at computer-generated data. Thirdly, the authors mention a missing extension: handling realistic problems such as missing observations.

2.10 Conclusion

It is shown that the field of video surveillance is very broad: it has many aspects that have all kinds of difficulties of their own. The papers describing surveillance systems often give not enough details to reproduce it and also do not publish their software on the Internet.

According to Räty [22], researchers have begun to consider architectures for video surveillance systems. In this thesis we propose an architecture for a multi-camera video surveillance system.

Furthermore, basic video surveillance techniques are combined with reasoning techniques and a state-of-the-art tracking technique called *P-N learning*. This combination resulted in an implementation and experiments are performed that show the performance of (a large part of) the system.

3 Requirements

This chapter describes what is to be expected from the surveillance system model. The main properties of the model are described in section 3.1. Assumptions about the area can be found in section 3.2. A Region of Interest analysis is given in section 3.3 and the chapter closes with a few words about the applicability in section 3.4.

3.1 Main Properties of the Model

The purpose of the model is to detect suspicious behavior. Therefore, it is required to observe the world and extract relevant information that can be used to draw conclusions. We would like to be alerted if something suspicious happens. In other cases, the model should not give any output. First we will describe the requirements if we would use a single camera. After that, the requirements are extended to the situation where a network of cameras is used. Finally, general requirements of the model will be mentioned.

Single Camera Situation First of all, we restrict ourselves to fixed cameras. Moving cameras that are for example installed on cars or Pan-Tilt-Zoom cameras are not taken into account. They would add extra complexity when determining the absolute position of the observable object(s). Second, most cameras have a frame rate of 25-50 frames per second. However, our model is not required to process every single frame, since objects are not expected to change significantly in $1/25$ second. A frame rate of one frame per second should be enough to capture all relevant events. Third, while processing the video, a small buffer is needed to save the frame. The processing should be finished before the next frame arrives. Fourth, the processing unit is able to detect movement and track the object as long as it is in sight.

Multiple Cameras Situation The model is extended to multiple cameras, because suspicious events often take place in a space that a single camera cannot cover. Some communication platform is needed between the units that process the video. Using the communicated information, conclusions should be drawn about how suspicious an object is. This requires some matching strategy, since an object must be tracked among multiple cameras. Because suspicious events also take place over a certain amount of time, history has to be taken into account when drawing conclusions. We feel that it is enough to take into account the last 30 minutes while determining how suspicious a person or car is.

The Model in General As mentioned in section 1.2, humans are able to incorporate context into the judgment of how suspicious an event is. We require the model to cope with context too. A list of ROIs is supplied and for each ROI a list of suspicious events are identified. Using this list it can for example be detected that it is suspicious for a car to park near a weapon depot.

Table 3.1: Summary of the concrete model requirements.

Description	Requirement
Camera location and orientation	Fixed
Frame rate	At least 1 frame per second
Buffer size	The maximum size of 1 frame
History buffer size	30 minutes

When suspicious behavior is detected, the model should alert the security operator. Then the security operator can observe the situation on its monitors and decide if (s)he should take action. The model should not give too many false positives. However, it is even more important that the system does not miss important events. For example, in the case that a bomb has been planted in the surveillance area, the system should never ignore that or label it as 'lightly suspicious'.

During this project, a model of a complete system will be designed. Since it is impossible to implement the whole system within the time schedule, some parts are selected to be implemented.

Table 3.1 summarizes the requirements mentioned above for which choices are made. The following list summarizes the elements that are required for a complete multi-camera video surveillance system, but for which no specific choices are made at this point:

- The video processing unit should be able to detect movement and track objects.
- A communication platform is needed.
- To track objects in space, some matching strategy is needed.
- Context is provided by a set of ROIs and the events that are suspicious there.
- The system should alert the operator when something suspicious happens and should not give too much false positives.
- A complete surveillance system model is designed, but only parts are implemented.

3.2 Area Assumptions

Designing a surveillance model that suites every possible area is a difficult task. Every area has different properties and security threads. Therefore, as mentioned before, we focus on a military area of the Nederlands Defensie Academie (NLDA) located in Den Helder, The Netherlands. A map of the area can be seen in figure 3.1. The area is surrounded by fences and only authorized people can enter the area using an access card. At the main entrance, security guards are able to give guests access to the area. The other two entrances are solely secured by cameras.

The functions of the buildings on the area are not publicly available. Therefore, we randomly labeled some buildings in order to define the ROIs. The blue labeled building on figure 3.1 is chosen to be the weapon depot, and the orange labeled building is for working and sleeping. Using this information we can for example state that the weapon depot is a Region of Interest where it is suspicious if someone parks a car.

The current setup of the cameras is not shown on the map, as they are not known. Nevertheless, we assume that they do not have an overlapping field of view and that they do not cover

Figure 3.1: The NLDA area.

the whole area. Furthermore, the data is sparse in the sense that the number of objects observed by one camera at one particular time instance is limited.

To summarize, these are the assumptions about the area:

- It is a military area that is surrounded by fences.
- Certain ROIs are identified, each with a list of events that are suspicious.
- The cameras do not have an overlapping field of view.
- The data is sparse: a limited number of objects are observable by one camera at one time instance.

Note that although we focus on the military area in Den Helder, this system is applicable to a broad variety of areas.

3.3 Regions of Interest

This section includes a list of Regions of Interest (ROIs) we constructed to illustrate how ROIs can be defined. After this list, we select a subset of ROIs which will be used in the rest of this report.

1. **ROI entrance barrier (see figure 3.2).**
   Visitors by car are supposed to have an access card. Showing this access card to an electronic reader opens the barrier automatically. Visitors without access card are supposed to push a button and communicate with the Surveillance Officer.

Figure 3.2: Region of Interest 1: entrance barrier. From [15].

(a) As soon as a car stops in front of the barrier, a camera takes recordings. These recordings are analyzed by a car template recognizer. This recognizer is using a special *Articial Neural Network* using the *Neocognitron architecture*. In general the recognition rate is 92 %, but given the fact that a limited number of cars are authorized to enter the area the recognition rate approaches 100%. In case an unknown car wants to enter the area, even if the driver has an access card, the driver gets a warning to visit the Officer of the Guard. See [15] for more information.
Technology: video/image analysis, car template recognizer (Neocognitron).

(b) It is not allowed to touch or move the barrier by human or by car. In case humans try to lift the barrier or demolish it by car this is observed by the camera.
Technology: barrier touch sensors, video/image analysis.

(c) A car blocking the entrance or exit for a longer time is not allowed. This is also observed by the camera.
Technology: video/image analysis.

(d) Humans jumping over the barrier or enter without using the access card is again observed by the surveillance system.
Technology: video/image analysis.

(e) The military guard is protected by the system by detecting aggression against him. The surveillance system observes visitors that (try to) kick, push or otherwise harm the guard.
Technology: video/image analysis, audio processing.

(f) Only one car is allowed to pass the barrier every turn. In case two cars pass the barrier at the same time, this is observed by the system.
Technology: video/image analysis.

2. **ROI street.**
Visitors in the military area have to take care of the traffic rules.

(a) Entering a one-way street is not allowed.
Technology: video/image analysis.

(b) Exceeding the speed limit is not allowed.
Technology: video/image analysis.

(c) Driving with lights turned off is not allowed during night.
Technology: video/image analysis, infrared camera.

(d) Crossing uninterrupted white lines or driving off-road is not allowed.
Technology: video/image analysis.

(e) Parking is only allowed at special places. Cars stopping for a long time are suspicious, for except on a parking place.
Technology: video/image analysis.

3. **ROI harbor or arsenal.**

(a) Stopping is not allowed.
Technology: video/image analysis.

(b) Leaving the car is not allowed.
Technology: video/image analysis.

(c) Taking pictures or filming is not allowed.
Technology: video/image analysis.

4. **ROI fence/gate.**

(a) Climbing a gate is not allowed.
Technology: video/image analysis.

(b) Taking pictures from military objects is not allowed.
Technology: video/image analysis.

5. **ROI Royal Naval College (KIM) area.**

(a) Cars have to be tracked from entrance to exit, disappearing cars or cars parking for a longer time is suspicious.
Technology: video/image analysis, car template recognizer, track matching, reasoning in place and time.

(b) Identity of cars (and possible car-driver) can be assessed by license plate. Some parts of the military are have limited access. Cars without license plate have no access.
Technology: video/image analysis, car template recognizer at entrance/exit, face recognition, reasoning.

(c) Visitors asking for entrance permission are supposed to take the shortest route to their destination.
Technology: video/image analysis, reasoning.

(d) Cars driving around for a longer time without clear destination are suspicious.
Technology: video/image analysis, track matching, reasoning in time and place.

(e) Based on recorded data, the most probable route for every car driver is known. The probability of taking a turn at a crossing is known. Abnormal routes are considered suspicious.
Technology: video/image analysis, track analysis, reasoning.

(f) Attempts to demolish cameras are suspicious behavior (climbing in lamppost, throwing objects, explosions of cameras recorded by other cameras in overlapping areas.
Technology: video/image analysis.

(g) Aggressive acts against personal or objects is suspicious, including gestures and shouting.
Technology: video/image analysis.

(h) People hanging around (outside smoke areas) or walking around for a long time on unexpected places (parking lot, around special buildings, looking into windows of cars or buildings).
Technology: video/image analysis, crowd analysis.

(i) People or objects falling or laying on the ground.
Technology: video/image analysis, reasoning in place and time.

(j) People grouping together into crowds.
Technology: video/image analysis, crowd analysis.

In this thesis we consider automatic surveillance system using video and image analysis. We focus on the NLDA area by taking a subset of the ROIs above. Actually, we take a subset of the suspicious behaviors within these ROIs, which we call *scenarios* from now on. The following scenarios are required to be detected by the proposed system.

A. Car exceeds speed limit.

B. Person runs.

C. Car drives where it is forbidden to drive.

D. Person walks where it is forbidden to walk.

E. Car or person stops where stopping is not allowed.

F. Car parks on a non-parking area.

To determine if it is *forbidden to drive/walk*, the information from the ROIs is used: if it is forbidden to drive in a certain ROI and a car is driving around there, this is considered suspicious behavior. For example, it might not be allowed to drive around on a tennis court or on the grass that separates the road from the fences. In the same way, it is also not allowed to stop or park a car at certain ROIs.

3.4 Applicability

Since the surveillance system model describes a security system, failures can have big consequences. That means that if a component of the system brakes down, the rest of the system components should continue their tasks as good as possible. This should be considered during the design of the model.

We will have a quick look at the feasibility of a real implementation of the model in section 4.2. Therefore, we also take into account the consequences in the real world. It is not the main focus to design an easy and cheap system that every company could implement, but we will avoid designing a model that is impossible to realize.

4 Design: Surveillance System Model

Now that we have described the requirements we can focus on the system model design. Section 4.1 describes the software architecture and section 4.2 handles the physical design.

4.1 Software architecture

This section describes the software architecture. Before the final version is shown, two other architectures are described to illustrate why the final version is chosen.

4.1.1 Software Architecture 1

Figure 4.1 shows one possible architecture of the software. A group of cameras is connected to a

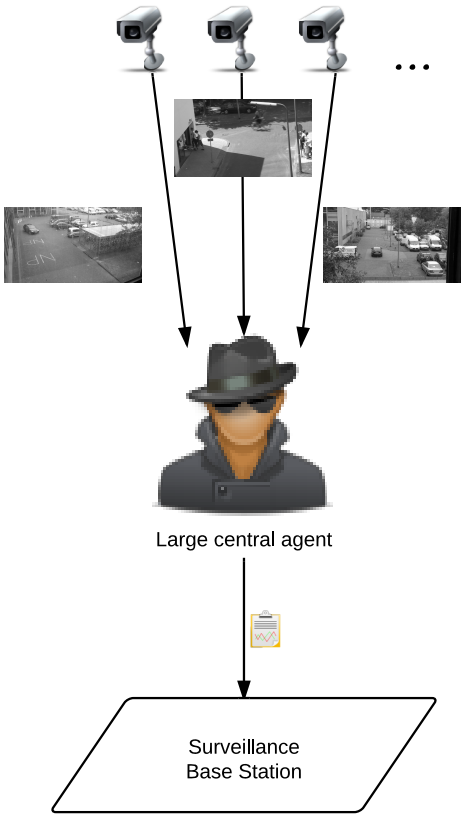

Figure 4.1: Software architecture number 1.

large central agent. This large agent processes all video streams, detects suspicious behavior and sends alerts to the Surveillance Base Station. In other words, this agent performs all processing tasks of the system. This approach has a few disadvantages. First of all, the agent is a single point of failure: if this agent fails, the whole system stops working. Second, dividing the responsibilities over multiple agents would make it easier to design, implement and maintain. Third, since processing (potentially large amounts of) video requires much processing power, the system may be unable to alert the Surveillance Base Station on time due to the lack of processing power for reasoning. That is why the first version of the architecture is rejected.

4.1.2 Software Architecture 2

Figure 4.2 shows another possible architecture. This architecture shows an agent network of three kinds of agents. Each of them is described below.

Observation Agent Receives the video signal, processes it and sends object information to the Reasoning Agent. While processing it, it determines the track of the object, the class of the object (Car/Person). Furthermore, it asks the Identification Agent for a *unique identifier* (ID) of this object. Each Observation Agent knows which other Observation Agent processes a video stream that is physically close. To take some load off the Identification Agent, object descriptions are sent to the closest Observation Agents, since the just-discovered objects are likely to be observed by the neighbor Observation Agents too.

Identification Agent Handles requests from the Observation Agents. Internally, this agent has a list of all objects it encountered. For each object, characteristics are known such as class, color, shape, etc. These characteristics are concluded from the information from the Observation Agents. For example, an Observation Agent could send a single image of an object to the Identification Agent. Then the Identification Agent matches it with the known object to see if it is a new object or not. Finally, it sends a unique identifier to the Observation Agent which can be used in the system (for example to describe the object to the Reasoning Agent). The advantage of such an agent is that there is only one entity that is responsible for matching objects. Knowledge about the map is asked to the Reasoning Agent.

Reasoning Agent Receives the object descriptions and trajectories from the Observation Agent and reasons about this. Mainly, it checks if the object has shown suspicious behavior using a few predefined suspicious events. Depending on the repetition and seriousness of these events, an alert is sent to the Surveillance Base Station.

This system still has a single point of failure: the Reasoning Agent. However, the system can easily be extended so that for example the Observation Agents can send alerts by themselves in certain situations. Imagine that the Identification Agent and the Reasoning Agent crashed and a person enters the gate without permission. If one of the Observation Agents see this and alert the Surveillance Base Station, the guards can still react to this situation. Another advantage of this architecture is that responsibilities are separated among different software entities. The Observation Agent is responsible for processing the video, the Identification Agent takes care of the object identification and the Reasoning Agent puts together this information and alerts the

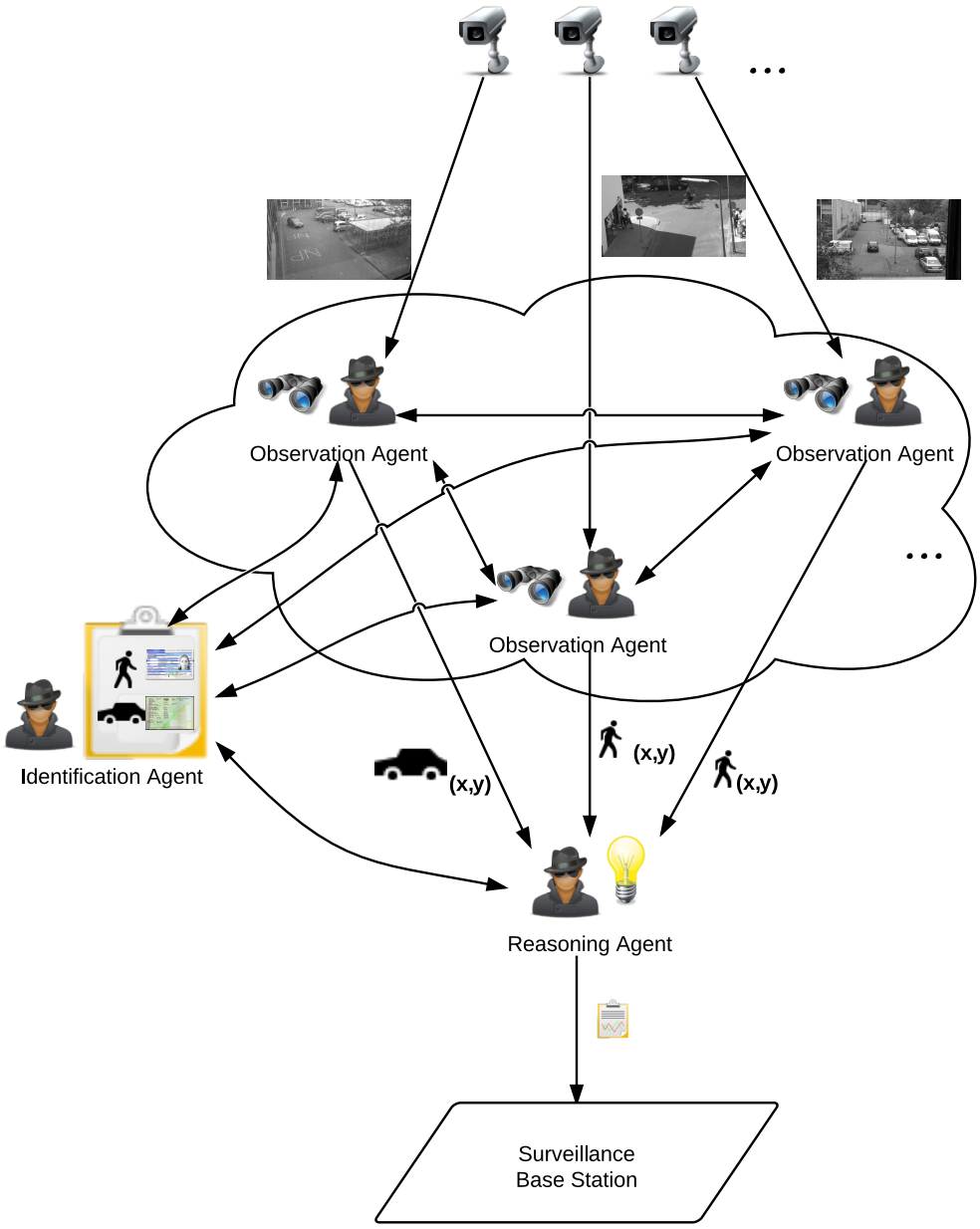

Figure 4.2: Software architecture number 2.

Surveillance Base Station when necessary.

4.1.3 Final Software Architecture

The final software architecture is shown in figure 4.3 on the next page. One difference with the second architecture is that the task of the Identification Agent is adopted by the Reasoning Agent. This way the Reasoning Agent can reconsider the identity of objects based on new information. Another difference with the previous architecture is the introduction of the Alerting Agent. This

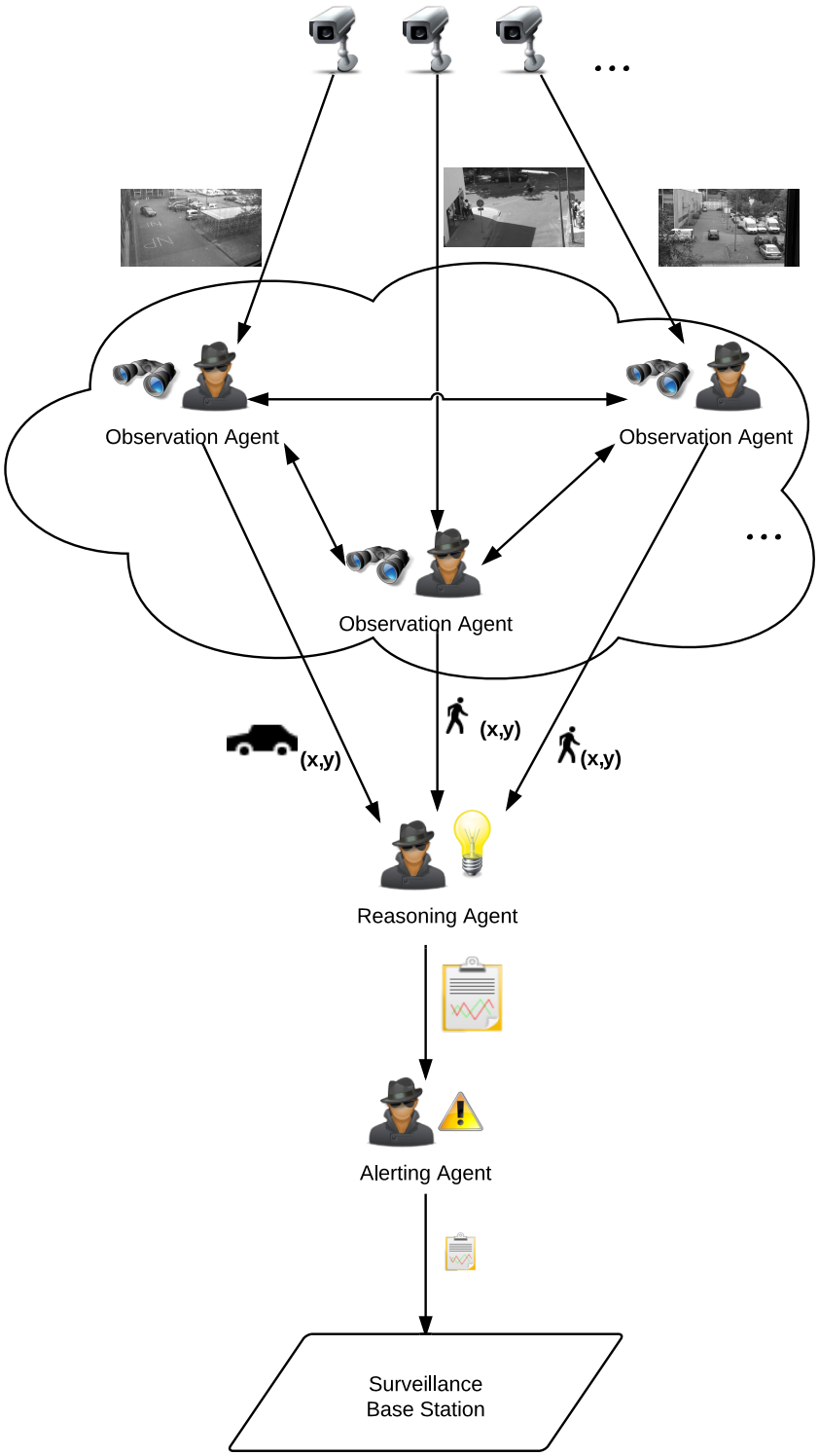

Figure 4.3: Architecture of the system

agent has the task to alert the Surveillance Base Station and functions as an interface between the system and the security guards. The main advantage is that the Alerting Agent can filter events to prevent swamping the guards. This architecture is considered as the best architecture because the agent have clear responsibilities, the Reasoning Agent has a complete overview and the operators are not swamped with alerts.

One important consideration in the design is the intelligence of the Observation Agent. We have chosen to make it reasonably intelligent: it extracts information and sends it, but it also knows about its neighbors and the direction of the object. The agent could however be designed to act even smarter. For example, the Observation Agents could buffer the trajectory information of each object: it sends a short notification to the Reasoning Agent when it detects an object, but sends nothing else until the object disappears or does something interesting. As long as the object disappears at some moment in time, the Reasoning Agents receives the same information, but the network is not transferring all these small messages with single observations. Another advantage of this approach is that implementing self-alerting Observation Agents is more easy in this case, because it has knowledge about what events are 'interesting'. However, when the set of suspicious events in the Reasoning Agent changes, the set of 'interesting' events in the Observation Agents is very likely to change too, which is a disadvantage in terms of software maintainability. Furthermore, the difference and correspondence between 'interesting' and 'suspicious' events might be vague and therefore may potentially cause errors. In other words, the taken approach is very suitable, since it separates the responsibilities *objectively observing the environment* and *detecting suspicious events*.

4.2 Physical Feasibility

The previous section elaborated on the architecture of the model. This section loses some words about the physical placement of the agents, since the system operates over a wide area. To realize the model that is described above, changes are needed at the NLDA. This section is aimed at exploring the feasibility of implementing the system in the real world. The first subsection below gives information about the existing equipment and what is required for the new system. The two subsections thereafter elaborate on two possible configurations.

4.2.1 Requirements

We assume that a working security system is installed. Security guards monitor the camera images and recorders save the video for later analysis. For the cost estimations in the following sections, we assume that the following hardware is installed:

- Twenty five analog CCTV tube cameras.
- Coax cables from the cameras to the rest of the equipment.
- Monitors that show the camera output.
- Recorders that capture the video for later usage.
- Workstation for the security guard(s).
- Enough networking equipment (switches and UTP cables).

Furthermore, we assume that there is a way to receive the same video as the monitors and recorders get.

The new system must provide processing power to all three different kinds of agents defined in section 4.1. We assume that the Reasoning Agent and Alerting Agent each can run on a recent quad-core server. The current implementation of the Observation Agent runs with about eight frames per second on a 3 Ghz Dual Core processor from 2007 (using one core). It is assumed that one recent quad-core server can handle four Observation Agent instances with eight frames per second, which is higher than the requirement defined in section 3.1. Note that resolution and image size greatly influence the performance, which means that extra processing power might be needed to do video preprocessing. This is not taken into account.

4.2.2 Central Processing

One solution is to place several computers in the Surveillance Base Station and let them digitalize, record and analyze the video streams. This solution is illustrated in figure 4.4.

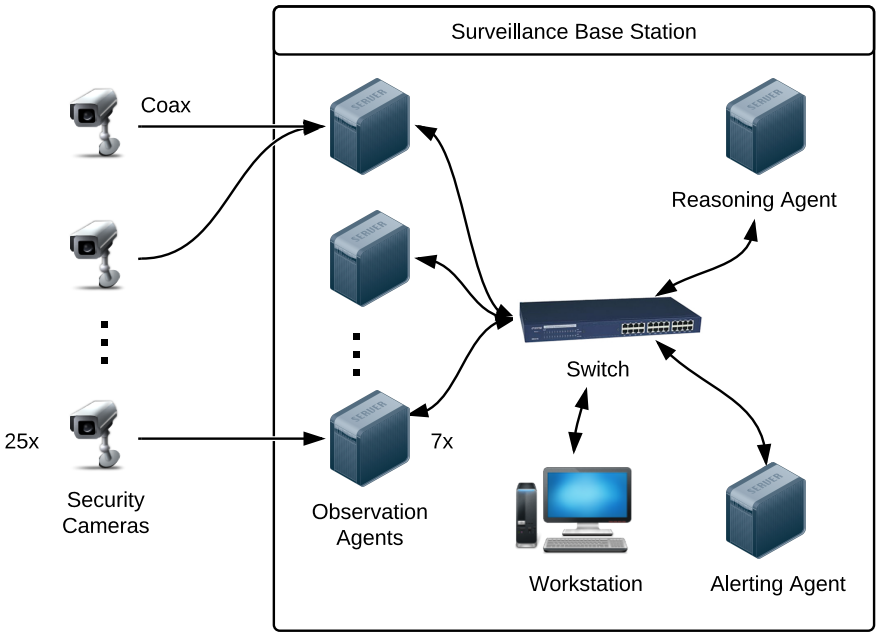

Figure 4.4: Central processing.

All cameras are attached to a server which runs multiple Observation Agents. A total number of seven servers are needed, because each server can manage a maximum of four cameras as mentioned in section 4.2.1. The rest of the servers run the other two agents. The Alerting Agent delivers its Alerts via the Workstation to the security guards, so these machines are placed on the same network.

Table 4.1 estimates the costs of this solution.

Each Observation Agent server needs a 4-channel PC card in order to read the camera output.

Table 4.1: Estimated costs of central processing.

Description	Model	Price ($)	Amount	Total price ($)
Quad-core servers for small businesses	HP ProLiant ML110 G6	499.00	9	4,491.00
PC card for video capture	Protech 4ch 120 fps MPG DVR Board	399.95	7	2,799.65
Total				**$ 7,290.65**

4.2.3 Distributed Processing

Another approach is to process the video near the cameras. Then each camera would be connected to a processing unit which analyzes the signal and sends it to the Surveillance Base Station. Figure 4.5 illustrates this solution.

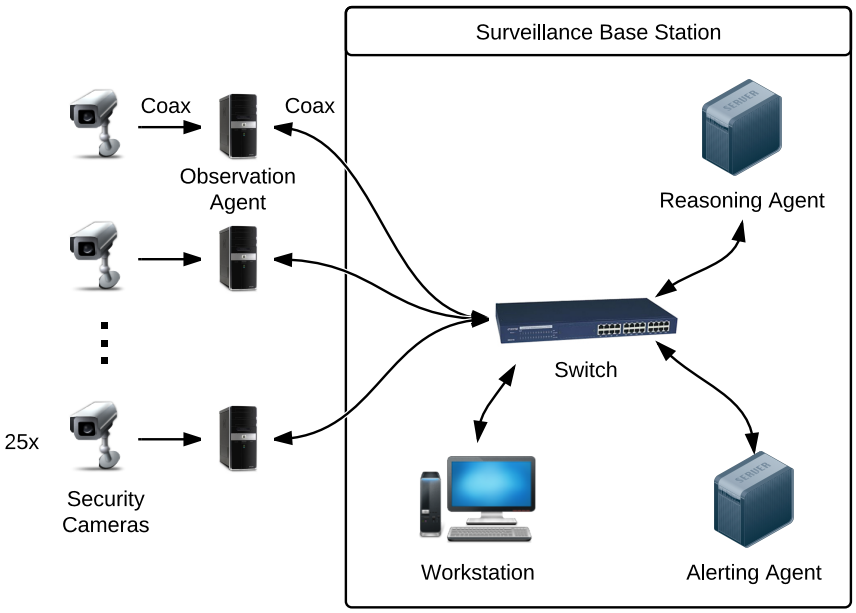

Figure 4.5: Distributed processing.

In this case, each camera is connected to a processing unit which is able to process one camera. This processing unit (a HP desktop PC in the cost estimation below) connects to the network in the Surveillance Base Station using the existing coax cables. To enable networking over coax, adapters are needed on each end of the cable. The rest of the system is similar to the central processing solution.

Table 4.2 estimates the costs of this solution.

Note that a solution has to be found to get the video from the processing unit to the monitors and/or video recorder on the Surveillance Base Station. One solution might be to stream the video via the network, which would require extra equipment to read the streams and supply it to the monitors and/or video recorder.

Table 4.2: Estimated costs for the distributed processing.

Description	Model	Price ($)	#	Total price ($)
Video processing unit for each camera	HP 500B Microtower PC	344.00	25	8,600.00
PC card for each computer	Elyssa Corporation EDV-XVAV Home Video Capture Card	53.13	25	1,328.25
Cable from camera to PC		10.00	25	250.00
Ethernet over coax adapter	Veracity VHW-HW	170.00	50	8,500.00
Quad-core servers for the two other agents	HP ProLiant ML110 G6	499.00	2	998.00
Total				**$ 19,676.25**

4.2.4 Discussion

Comparing both solutions result in the following advantages of each solution. Note that these advantages are disadvantages for the other solution.

Advantages of Central Processing:

- Only the Surveillance Base Station has to undergo installation activities.
- Its price: $7,789.65 instead of $20,175.25, which is 159% more expensive.

Advantages of Distributed Processing:

- No single point of failure: if for example the Surveillance Base Station has a power failure, the Observation Agents will still work. To utilize this advantage, the Alerting Agent and networking switches should be placed outside the Surveillance Base Station and it should be able to reach the security guards other than using the workstation. Furthermore, the Observation Agents should be able to communicate to the Alerting Agent directly.

The choice of which solution to use depends on the requirements: if placing equipment near the cameras is no problem and sabotage is a big issue, distributed processing might be the best solution. If not, central processing is. We can however conclude that the proposed system is realistic and can be implemented for less than $8,000 of hardware costs.

The choice of which physical solution to use has an influence on the software design, although the influence is limited. Imagine that the centralized approach is chosen. Then it does not matter how our software is organized: distributed or centralized, because distributed software can often run on a single machine. The limitation comes into play when a distributed solution is chosen. Large investments are done to install video processing units across the surveillance area, so it is expected that the software is distributed too. A centralized application would not suffice, because the video processing units will be useless and the central servers might not have enough processing power to run the application. To conclude, the most flexible way of designing a surveillance application is to make it a distributed application that can also run on a single system with little effort.

4.2.5 Other options

To make this chapter complete, other possible options are listed below including the reasons why they are not applied above.

- Make the video processing units wireless. A wireless PC card costs about $\$50$, so it will save $((2 * 170) - 50) * 25 = \$7,250$ if a wireless network is already present. However, this would make the system less stable since wireless networks are influenced by noise and make sabotage easier.
- Avoid using the coax cable by using usual UTP cables for networking. However, this is very expensive since these cables need to be put under the ground.
- Replace the CCTV cameras by IP cameras, which support networking over Ethernet by default (wired or wireless) and have higher resolutions. This is a good option if the CCTV cameras are not suitable any more. However, it is not needed for the system to work and outdoor IP cameras cost about $300 - $500 per piece.

5 Resources

In order to build a fully functional automatic surveillance system, several components are necessary. This chapter gives an overview of the resources that are used for the system. Because implementing the whole system is outside the scope of this thesis, several existing tools are used to be able to create a working system. These existing tools will be described in the first five sections. Section 5.6 closes this chapter with a description of the datasets we use to test the Observation Agent.

5.1 Object Tracking

5.1.1 Introduction

For human operators it is important to see how objects move through the video. For example, it is important to know how fast a person or a car is moving. As mentioned before, humans are generally very capable of tracking objects, even if they move very fast. Our brain predicts where the object is expected to be in the next moment in time. It is able to do that because we observe the trajectory of the person and know the context, for example: a person cannot move through a concrete wall. For a video surveillance system to work properly, it must be able to track objects too.

For this tracking task we use a tool called Predator [14, 13], also known as OpenTLD. Section 5.1.2 explains the algorithm behind this tool and section 5.1.3 describes the tool itself.

5.1.2 P-N Learning

The software tool Predator uses a new technique called P-N learning [14]. The main idea is that the tracker learns appearances of the object (positive examples) and its direct neighborhood (negative examples). The algorithm is able to learn from its mistakes.

The task of the P-N learning algorithm is to learn a classifier that labels each unlabeled sample from a feature set using an a priori labeled set. This task comes down to estimating the parameter θ which is learned from a training set, like with supervised learning. However, unlike supervised learning, P-N learning iteratively augments the training set by examples from the constraints of unlabeled data. During the training procedure, each iteration assigns labels to unlabeled examples using the classifiers trained in the previous iteration. "The constraints are then used to verify if the labels assigned by the classifier are in line with the assumptions made about the data. The example labels that violate the constraints are corrected and added to the training set. The iteration is finished by retraining the classifier with the updated training set. This procedure iterates until convergence or other stopping criterion" [14]. The conclusion is that the imperfect positive and negative examples cancel their errors under certain conditions.

This algorithm is applied to video object detection. The problem is stated as follows: "given a

single example of an object, learn an object detector on-line from unlabeled video sequence"[14]. An object detector is used which uses the scanning window strategy. A Lucas-Kanade tracker is used to track the object from frame to frame. This tracker is evaluated using the confidence determined by a normalized cross-correlation between the tracked patch and the patch selected in the first frame.

Processing the video sequence is done as follows: for each frame, both the detector and the tracker find the location(s) of the object. "The patches close to the trajectory given by the tracker and detections far away from this trajectory are used as positive and negative examples, respectively"[14]. If the trajectory is considered valid (the tracker has a 80% confidence in the last frame), these examples are used to update the detector. However, if the trajectory is invalid, the examples are discarded and the tracker is re-initialized.

The algorithm is applied to synthetic data and on real data. The former experiment shows that using both P and N examples make that errors cancel out, whereas using either P or N give worse results. The experiment on real data is performed on six video sequences that are all used in experiments of five other papers. For five out of six video sequences, the algorithm manages to keep track of the object longer than algorithms described in all other papers.

5.1.3 Predator

Predator is released under the GNU GPL version 3 in April 2011. It is a result of the PhD research done by Z. Kalal at the University of Surrey Guildford, United Kingdom. It is written in MATLAB and C++ and makes use of OpenCV [3].

As described above, the algorithm needs one sample before it can be initialized. For Predator it means that the user needs to indicate where the target object is in the first frame of the image sequence. The user can do this manually in the dialog window showed in figure 5.1. Here the car in the foreground is selected by the user. After double clicking inside the rectangle, the algorithm will start tracking playing the movie and tracking the car. The rectangle will be called the bounding

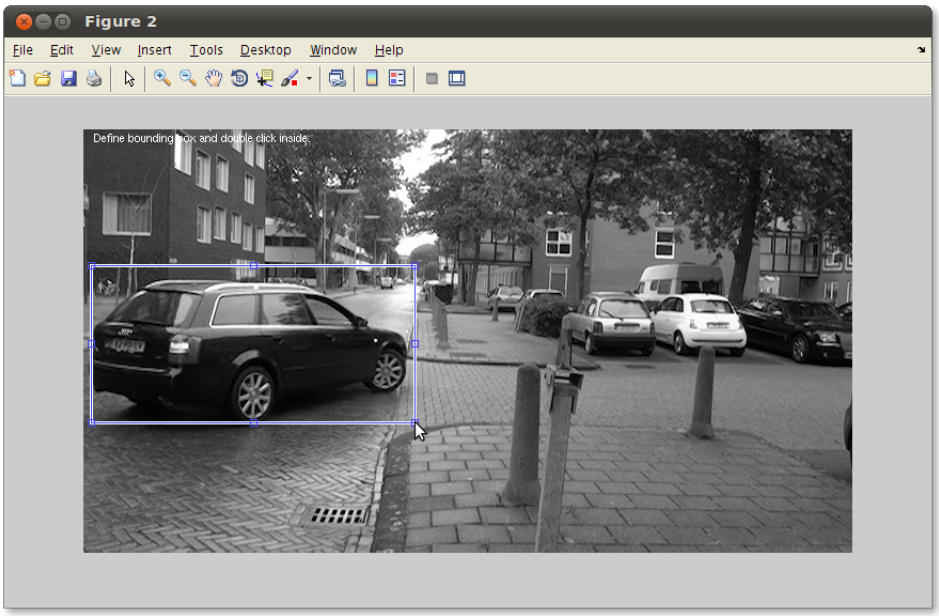

Figure 5.1: Predator: dialog window for manually defining the initial bounding box.

box from now on. This proces is automated in the implementation described in section 8.1.1.

A second way to provide the bounding box is via a text file. An example dataset that is included with Predator includes such a text file. If this dataset is fed into Predator, it starts tracking without asking user input. Figure 5.2 shows how tracking with Predator looks like. Screenshots of three frames are taken. On the left of each frame the negative examples are shown and on the right the positive. The number of examples grow through the tracking process, which indicates that it is learning on the way. In the picture, the motor bike is surrounded by a bounding box. The algorithm is able to track the motorbike most of the time, although the camera is shaking and the object sometimes is not fully visible when it takes a jump. Note that the frame rate that is indicated for each frame includes the time that is needed to create this screenshot. These screenshots are taken on a 3 GHz dual core computer. The example dataset contains color image frames, but Predator converts them to gray scale in the process.

The various options of Predator include an option to show the track of the object. Figure 5.3 shows the track of the motorbike until frame #40.

Predator contains 3062 lines of code[1] of which 1702 is MATLAB code and 1253 is C++ code. Before running Predator, a MATLAB script needs to be executed to compile the C++ code. An example script is supplied which enables the user to run Predator with the motorbike example or with the input from a webcam.

5.2 Object Classification

5.2.1 Introduction

A video surveillance system needs to be able to distinguish different kinds of objects in order to do intelligent reasoning. Imagine that the system observes an object with a speed of 15 km/h. If it is classified as a car, a cyclist or a bird, this is a normal speed. However, if a person without a bike would travel with this speed, (s)he is running and something might be wrong. Furthermore, classification might be useful to determine if the object is interesting. In most situations, animals like birds or dogs are not interesting. A camera above a highway might only be interested in cars, trucks and buses. And an airport security system might want to recognize bags to detect if they are left unattended.

With classification we mean the algorithmic procedure often applied in pattern recognition. During this procedure, a dataset is collected with a large amount of examples. These are labeled with the ground truth. For example, if one would like to detect if the picture contains a face, large amounts of pictures with a face are collected and they are labeled as 'face'. In order to learn the classifier how non-face images look like, one also collects a large amount of images without a face and label them 'non-face'. Then the classifier is fed with this dataset. The classifier uses some algorithm to learn the difference between 'face' and 'non-face' images. Often about 10% of the dataset is not fed into the classifier, because it is used to test the classifier afterwards.

5.2.2 VLfeat

To implement an Object Classifier, an open source library VLfeat[2] is used. This library contains implementations of several computer vision algorithms. Since it is written in both C and MATLAB, it is a good combination with the MATLAB tracker. The source comes with a sample script called

[1] Calculated with CLOC 1.55, downloaded from http://cloc.sourceforge.net

[2] http://www.vlfeat.org/

Caltech-101 classification that trains and tests a classifier on the Caltech-101 dataset[3]. This dataset has pictures of objects in 101 different classes. Each class consist of 40-800 images. The

[3] http://www.vision.caltech.edu/Image_ Datasets/Caltech101/

Figure 5.2: Predator: overview of tracking the motorbike example.

Figure 5.3: Predator: track of the motorbike until frame #40.

sample script downloads this dataset, trains a classifier with 5 classes and tests it. How we used this sample application to implement the Object Classifier is described in section 8.1.3.

5.3 Reasoning

To make intelligent decisions, the Reasoning Agent needs some kind of reasoning mechanism. This mechanism has some requirements. For one, time should be taken into account. If an object acts suspicious for a period of 15 minutes, it is more suspicious than an object that acts the same over a 48 hour period. Second, we would like to express a degree of how suspicious a person is at a certain moment, since we want to avoid lots of false alerts. We introduce the principle *suspicious rate*: a degree of how suspicious the object acted during a certain time window. Since one action is less suspicious than another one, we need some kind of weighting to express that difference between actions. This section is structured as follows: section 5.3.1 describes how expert systems are considered as reasoning mechanism. Section 5.3.2 explains why Bayesian Networks are chosen and shows an example. Section 5.3.3 describes the noisy-OR gate. Section 5.3.4 describes Dynamic Bayesian Networks and shows an example. Section 5.3.5 closes with a description of the Bayes Net Toolbox for MATLAB.

5.3.1 Expert Systems

A reasoning mechanism that is often used is an Expert System. Basic Expert Systems are constructed with the knowledge from field experts. This knowledge is encoded in rules that are collected in a rule base. When input is fed to the Expert System, the conditions of some rules might be true and conclusions are drawn. With these conclusions, other rules might fire, and so on. A major advantage of Expert Systems is that it can explain why a specific conclusion is drawn. In our case, it could tell us why it draws the conclusion why object X is suspicious. However, standard Expert Systems can only cope with discrete values, which makes the output discrete. Since we need a suspicious rate as output, Fuzzy Expert Systems could be seen as a solution. This

kind of system can cope with concepts like 'very' or 'extremely', so we could work with outputs like 'a bit suspicious' or 'very suspicious'. However, the threshold of what is suspicious and what is very suspicious is defined within the system. We would like the Alerting Agent to determine when an object is suspicious enough to alert the operators and not the Reasoning Agent.

5.3.2 Bayesian Networks

A good example of reasoning mechanisms where numbers are used is the field of Probabilistic Reasoning. The Bayesian Network model is used for reasoning with uncertainty based on formal rules of probability theory [26]. Figure 5.4 shows an example of a simple Bayesian Network (BN). A BN is a directed acyclic graph which show the relation between random variables. In this case,

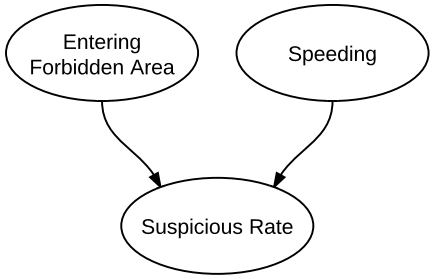

Figure 5.4: Example of a simple Bayesian Network.

Entering Forbidden Area and *Speeding* cause suspiciousness, which we express in the *Suspicious Rate*. In this thesis we solely consider boolean nodes which means that each node can take the value true (T) or false (F). A Condition Probability Distribution defines the probability that the node is true or false given the state of its parents:

$$P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i))$$

[23]. The probabilities $P(x_i \mid \text{parents}(X_i))$ are defined by the Conditional Probability Table of each node. Table 5.1 shows the probability tables for the nodes from figure 5.4.

Table 5.1: Conditional Probability Tables for the Bayesian Network example.

(a) Entering Forbidden Area (X_1)

P(X_1=T)	$P(X_1 = F)$
0.5	0.5

(b) Speeding (X_2)

P(X_2=T)	$P(X_2 = F)$
0.5	0.5

(c) Suspicous Rate (X_3)

X_1	X_2	$P(X_3 = T)$
T	T	0.9
T	F	0.2
F	T	0.3
F	F	0.0

Given the network and the CPTs it is defined that if both *Entering Forbidden Area* and *Speeding* are true, $P(X_3 = T) = 0.9$. In other words: if it is observed that the object is both *Entering Forbidden Area* and *Speeding*, the *Suspicious Rate* is 0.9. Similarly, if only *Entering Forbidden Area* is true, the *Suspicious Rate* is 0.2, etc.

During the reasoning process, evidence is allocated to nodes. Using a process called *inference*, conclusions are drawn.

5.3.3 Noisy-OR

In this thesis, the CPT of the *Suspicious Rate* is determined by a noisy-OR model [21, 5]. Wiggers et al [27] have used this model for reasoning in a decision support system. The noisy-OR model simplifies defining the CPT: instead of defining 2^k conditional probabilities, a noisy-OR needs just k probabilities, where k is the number of parent nodes. For each parent a *linking probability* is defined:

$$P_i = P(Y = T|X_i = T, \{X_j = F\}_{j=1,j\neq i}^{n}).$$

Figure 5.5 shows an example with the linking probabilities.

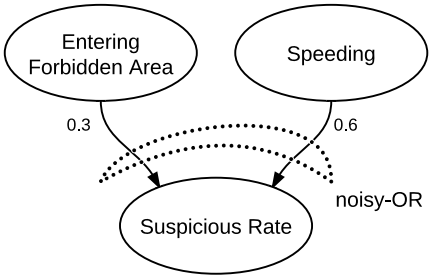

Figure 5.5: Example of a simple Bayesian Network with a noisy-OR.

In the general case, imagine parent node X_i is set to T and the other parent nodes are set to F. The probability that its child Y is set to T is equal to $(1 - P_1)$. The disadvantage is that only binary nodes can be used. To calculate the conditional probability of a general noisy-OR node given the probabilities of their parents, the following formula is used :

$$P(Y = T|X_1, ..., X_n) = 1 - \prod_{i:X_i=T} (1 - P_i)$$

For the example of figure 5.5 it means that

$$P(X_3 = T \mid X_1, X_2) = 1 - (1 - 0.3)(1 - 0.6) = 0.72.$$

5.3.4 Dynamic Bayesian Networks

Since we have to take time into account during reasoning, Dynamic Bayesian Networks (DBNs) are used [18]. A DBN can be defined by two BNs: a prior model and a transition model [26]. Figure 5.6 shows an example of such a DBN definition.

The *Suspicious Rate* node from time slice 2 has an extra parent: the *Suspicious Rate* from slice 1. This way the *Suspicious Rate* is incorporated in the judgment of the next moment in time. In other words, if the *Entering Forbidden Area* and *Speeding* nodes are true in both slices, the *Suspicious Rate* of slice 2 is higher than the rate of slice 1. Using this definition, a DBN inference algorithm can also reason about consecutive time slices, since the linking probabilities are not assumed to change over time.

Note that the linking probabilities from figure 5.6 are illustrative: the probabilities used in our system are defined in section 9.2.1.

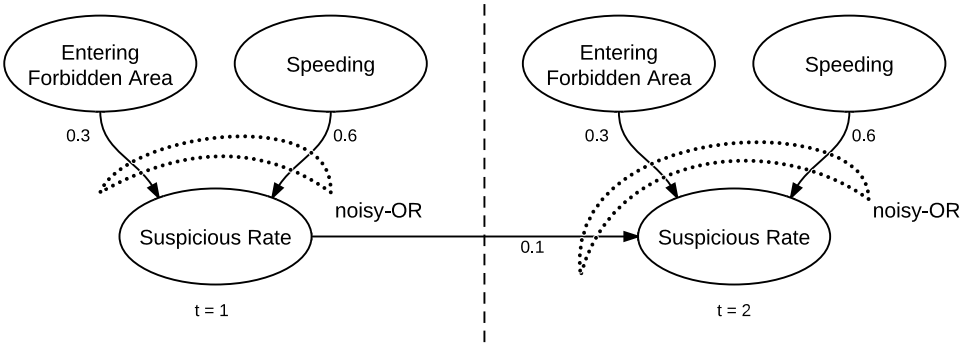

Figure 5.6: Example of a simple Dynamic Bayesian Network with a noisy-OR.

5.3.5 Bayes Net Toolbox

BNT is an open source MATLAB tool written by Kevin Murphy. This particular tool is chosen because it supports both Dynamic Bayesian Networks and noisy-OR gates, it is programmed in MATLAB, the author has written multiple Bayesian Networks papers and the documentation is clear. It does not come with a GUI, but the user can visualize his Bayesian Networks with third-party tools or other MATLAB toolboxes.

5.4 Agents

5.4.1 What is an Agent?

Before we explain what agent framework is used, it is explained what we mean by an agent. An agent in the context of artificial context can be defined as:

> "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [23].

An example of an agent is a robotic arm in a factory: it perceives its environment trough cameras or pressure sensors and works on the product through its arm. A completely different agent would be a virtual negotiation agent that perceives its environment (the bids and the virtual attitude of the opponent) and acts upon its environment by placing bids or walking away. Also human being and animals are agents, according to this definition.

The kind of agent we use is sometimes referred to as a software agent: it is a piece of software that acts like an agent. In other words: it perceives its environment and acts upon it. However, we require the agents to have another property that is common in agent theory: *independency*. After configuration is done, the system (and therefore the agents) should be able to work independently of a user. An exception is the Alerting Agent, which can be implemented such that it alerts the user according to their current preferences.

5.4.2 Agent Communication Framework

Since the agents need to communicate and we want to implement it in a standarized way, an agent framework needed to be chosen. For this the master's thesis from M. Beelen is used [2]. Beelen compares different agent platforms. His thesis "explores the possibilities of an application on a handheld device, providing users of public transportation with an up-to-date advice based on

the current delay situation on the railroad network in The Netherlands." Four agent architectures are described and compared:

- GALAXY.
- Open Agent Architecture.
- COUGAAR.
- Foundation for Intelligent Physical Agents (FIPA):
  "FIPA tries to establish a common standard for agents and agent applications. FIPA does not specify how the specifications should be implemented, but only specifies the functional behaviour which the implementation should have."

Three different FIPA implementations are described:

- Tryllian Agent Development Kit (Tryllian ADK).
- FIPA-OS.
- Java Agent DEvelopment framework (JADE).

The advantages of JADE are described as follows: "The user base of JADE is very large, among which companies like British Telecom, France Telecom, KPN and Philips and universities like Imperial College of Londen, University of Helsinky and Parma. Several add-ons exists for the JADE platform implementation. Most notably are the LEAP add-on, which combined with JADE forms a platform for mobile devices, the Jadex add-on, which makes it possible to design agents based on the Beliefs, Desires and Intentions (BDI) model, and the HTTP MTP add-on, which allows agent communication over the HTTP protocol."

The comparison of the agent architectures and their implementations can be summarized as follows:

- The disadvantage of GALAXY is that it does not support distributed applications.
- The disadvantages of the Open Agent Architecture is that it is written in the C language, it does not allow direct communication between agents and it has an unclear license for commercial use.
- "The main advantages of the FIPA standard are that it has different implementations with their specific benefits and licenses and does not limit agent communications but rather defines how agents should communicate with each other."
- There are no restrictions on the use of JADE, LEAP or Jadex because of licenses.
- JADE has the largest user base and the most advanced documentation and development tools among the described FIPA implementations.
- JADE has extensive possibilities for creating distributed platforms.
- COUGAAR is a very robust and scalable platform.
- COUGAAR lacks the advantages of a large user base like JADE has.

- JADE is better documented and easier to use compared to COUGAAR.
- COUGAAR does not comply with the FIPA standard, which makes it impossible to communicate with other FIPA agent platforms.
- "COUGAAR offers an interesting alternative architecture for the behaviour oriented or BDI agent models."
- "One of the problems with JADE is how to share information between different agents."
- "An interesting option is to use JADE and COUGAAR simultaneously to combine the strengths of both and reduce the disadvantages."

Beelen has chosen JADE as agent framework. The video surveillance system from this thesis has similarities with the application from Beelen: it is also a distributed agent network. Furthermore, JADE is still under development (last modification is very recent). Therefore, JADE is chosen for this thesis as well.

5.5 Simulation

5.5.1 Introduction

One possible test setup to test the Reasoning Agent is making several simultaneous video recordings, feed it to the Observation Agents and observe how the Reasoning Agent handles it. However, multiple cameras will be needed for simultaneous video recordings and for both simultaneous and non-simultaneous recordings a lot of data is generated. Furthermore, when a new scenario needs to be tested, new recordings have to be made, which is a very time consuming task. Finally, there may arise other problems like bad tracking results which is caused by bad recordings or failing Observation Agents, which have nothing to do with the Reasoning Agent itself.

To avoid the problems mentioned above, we designed a simulator with which we can simulate cars and people moving through the area. By doing this, the input of the Reasoning Agent is 100% correct and we can focus on the performance of the Reasoning Agent alone. Several cameras (Observation Agents) are positioned in the area. The input of the dummy Observation Agents is very simple: a car or person is seen at a certain position. This information is directly communicated with the Reasoning Agent and this agent will reason about it, which we can check afterwards.

An important consideration is what information is communicated exactly. The simulator knows the identity of each object, but the Reasoning Agent does not receive this information. In the real world, it also receives just the local information from the Observation Agents and it tries to match the different observations to see which observations belongs to the same object. Therefore, the dummy Observation Agents communicate three things to the Reasoning Agent: the class of the object, the local identification number and the position of the object. The 'local identificaton number' is explained in section 7.2. This amount of information is also available in a 'real' Observation Agent, for except that the class and the position of the object may not be accurate.

The path of the car and/or person has to be predefined by the user, just like the position and orientation of the cameras. Afterwards, it can be checked if the Reasoning Agent alerted the user as expected.

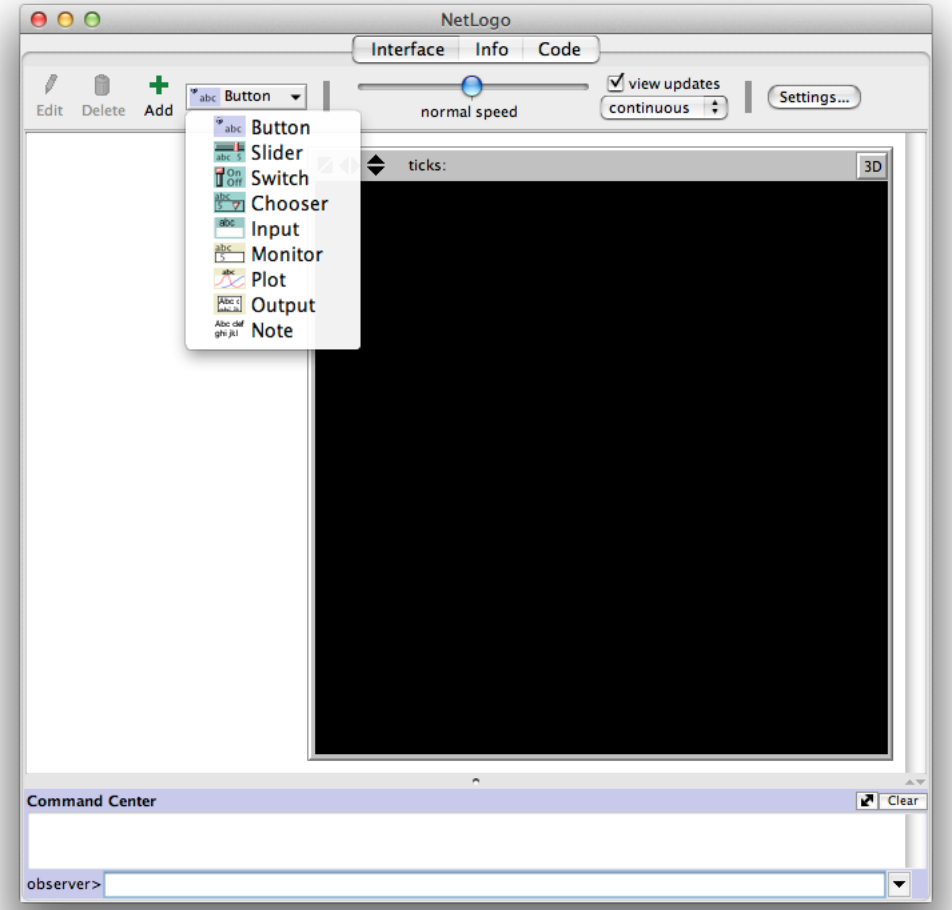

Figure 5.7: NetLogo with an empty project file.

5.5.2 NetLogo

To implement a simulator, the tool NetLogo [29] is used. "NetLogo is a multi-agent programmable modeling environment" [28]. Figure 5.7 shows the interface of NetLogo when it just started and the *Button* dropdown menu is clicked. In the *Interface* tab (which is enabled in the figure) the interface of the NetLogo model can be defined by the user. The dropdown menu shows that different kinds of elements can be added to the interface: buttons, sliders, switches, etc. The *Code* tab gives the user the chance to program the behavior of those buttons, sliders, switches, etc. The black screen is called the *World* of the model. It shows the state of the model. In NetLogo, an agent is called a *turtle*. They are able to move around in the World. The World consist of square elements called *patches*. Section 6.4 shows how our project file looks like.

5.6 Data

5.6.1 Our Datasets

To test the Observation Agent we've captured several videos with cars and people. Figure 5.8 shows the first dataset.

Figure 5.8: Impression of dataset 1.

The car accelerates and drives away from the camera. This dataset is an exception because the first frame already contains the object. To compensate for this, the background image is constructed in such a way that the car is not visible at all.

Figure 5.9 shows the second dataset.

Figure 5.9: Impression of dataset 2.

This dataset shows how a person walks to his car.

The third dataset is shown in figure 5.10.

The frame sequence shows a car that drives towards the camera. In the first frame, the car is behind trees and other car. So the object does not enter the frame via one of the borders, like the other datasets.

Figure 5.10: Impression of dataset 3.

Figure 5.11 shows dataset 4.

Figure 5.11: Impression of dataset 4.

This dataset shows a highway. For this thesis we focus on the car that appears on the right lane in the second frame of figure 5.11.

The last dataset is shown in figure 5.12. It shows a person that walks from left to right.

5.6.2 Privacy

The datasets described above are created and chosen such that they do not reveal sensitive information about the NLDA. Even more, none of these datasets are recorded on the NLDA area. Furthermore, the privacy of the drivers and pedestrians are respected: they are not recognizable or permission is provided if the license plate is readable.

Figure 5.12: Impression of dataset 5.

6 Computational Reasoning

Chapter 5 described the resources we use for our implementation. Before the internal structure of the agents will be described in the next chapter, this chapter will explain how we applied the resources.

6.1 Training the Object Classifier

To enable the system to classify each object we have trained a classifier using VLfeat, a library that is described in section 5.2. Three classes are defined: CARS, PEOPLE and OTHER. The first two classes speak for themselves. The OTHER class is introduced to cope with photos of potential backgrounds: roads, trees, walls, etc. To train the classifier, images are downloaded from all kinds of scientific sources. The classes CARS and PEOPLE consist of about 830 images each and the class OTHER has 380 images.

Before we could start training, the existing *Caltech-101 classification* application needed to be adjusted. We will not go into detail how it is adjusted, but the coupling with the Observation Agent will be described in section 8.1.3. It is interesting to mention that the script extracts PHOW features from each image, that a k-means clustering algorithm is used and that the classifier is a Support Vector Machine (SVM).

Since the smallest class contains 380 images, a classifier is trained on 342 images and tested on 38 (90% training data, 10% test data). The result was an accuracy of 70.18%. More tests show that if the number of used images is lowered, the accuracy increased. That indicates overfitting of the classifier. Eventually 60 training images are used and 10 test images which resulted in an accuracy of 83.33%. The train/test ratio is changed in order to have a low number of training image and a sufficient amount of test images. With more effort a classifier with an accuracy above 95% is feasible. However, with the limited amount of data and time, the current classifier performance of 83.33% suffices. Section 9.1.3 shows how the classifier performs on the object pictures from our datasets.

6.2 Dynamic Bayesian Networks

Section 5.3.4 described Dynamic Bayesian Networks in general. This section elaborates on how DBNs are applied in the Reasoning Agent. Section 6.2.1 explains the DBNs that are used and section 6.2.2 describes how the Bayes Net Toolbox is used.

6.2.1 Reasoning Agent DBNs

Figure 6.1 shows the Dynamic Bayesian Network for a car. Two time slices are shown. Both slices have four nodes, but only the *Suspicious Rate* node is shown from slice 2. All nodes are

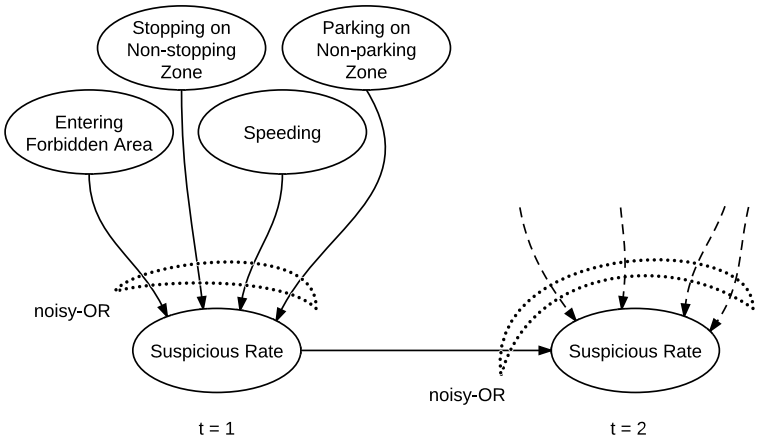

Figure 6.1: Dynamic Bayesian Network for a car.

binary nodes: they can either take the value true or false. The first node, *Entering Forbidden Area* represents the action that a car has entered a forbidden area. *Stopping on a non-stopping zone* means that the car has a speed of zero where this is not allowed. The node *Speeding* resembles the act of driving faster than the speed limit. The last node, *Parking on non-parking zone* is true if the car is standing still for a long period where this is not allowed.

Figure 6.2 shows the Dynamic Bayesian Network for a person. Only the nodes *Running* is different from the car network. It represents the act of a person that is running.

6.2.2 Combining BNT and the Reasoning Agent

An attempt is made to implement the Reasoning Agent in such a way that we can easily replace tools that are used. Regarding BNT it means that we should be able to replace BNT by another MATLAB toolbox which can build Dynamic Bayesian Networks. Therefor, a small set of MATLAB functions are created that are used by the (Java) Reasoning Agent. These MATLAB functions use the BNT toolbox, but if another toolbox is introduce, only these functions need to be rewritten. If a non-MATLAB tool is chosen for the Reasoning Agent, we cannot use these functions, but

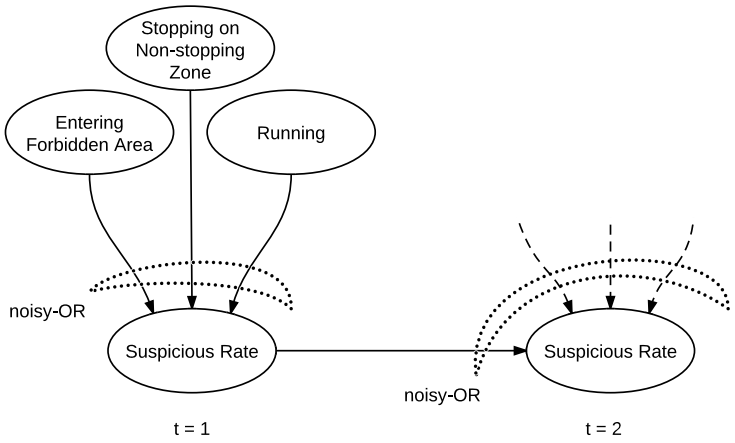

Figure 6.2: Dynamic Bayesian Network for a person.

that is out of the scope of this section.

The following functions are written to combine the Reasoning Agent with BNT:

- [bnet, engine] = create_reasoningagent_dbn(inhibit): create a DBN for the Reasoning Agent with the inhibit probabilities.
- [ev] = create_evidence_array(bnet): create an empty evidence array for the given DBN.
- [ev, engine] = add_evidence(engine, ev, t, newEvidence): add new evidence to the evidence array of the DBN.
- [s] = get_suspiciousness_of_slide(engine, t): returns the suspiciousness rate of slide t given the engine.

These functions are called in this order. The functions add_evidence and get_suspiciousness_of_slide are often called multiple times to add more evidence and acquire the new suspiciousness rate.

6.3 JADE

Section 5.4.2 describes why JADE is chosen and provides some details. This section explains how JADE agents are organized. Figure 6.3 illustrates how the JADE agents of this thesis are organized. A JADE network can consist of one or more platforms. Each platform contains a *Main Container* and zero or more regular *containers*. A container can contain one or multiple agents. The Main Container always contains the Agent Management System (AMS) and the Directory Facilitator (DF). In short, the AMS provides a naming service and is the authority of the platform while the DF helps agents find each other. Each regular container within a platform is registered with the Main Container.

The agent network in the example figure runs on three hosts. Host 1 contains the Main Container and the other two hosts have a container with an Observation Agent. The agents communicate via the network. New Observation Agents can be added to the regular containers if they run on the same host. If a new host is introduced, it creates a new container for his Observation Agent(s). The Alerting Agent is not shown in the figure.

As quickly mentioned above, JADE networks can contain multiple platforms. Agents can communicate with each other regardless of the platform they are a member of. The main advantage of this feature is that a JADE platform can be connected to non-JADE platforms as long as they obey the FIPA specification. In order to keep it simple we have used a single platform.

6.4 NetLogo

The simulator uses the tool NetLogo described in section 5.5.2. A NetLogo project is created that performs and visualizes the simulation, shown in figure 6.4. This project file contains code that loads the map properties, the object configuration and the camera configuration. Furthermore, it contains code that move the objects through the world given its path. The object information is passed on to a custom NetLogo extension that communicates the information to the Reasoning Agent via the JADE framework.

The NetLogo simulator needs three configuration files: a map definition file, an object configuration file and a camera configuration file. These will be described below.

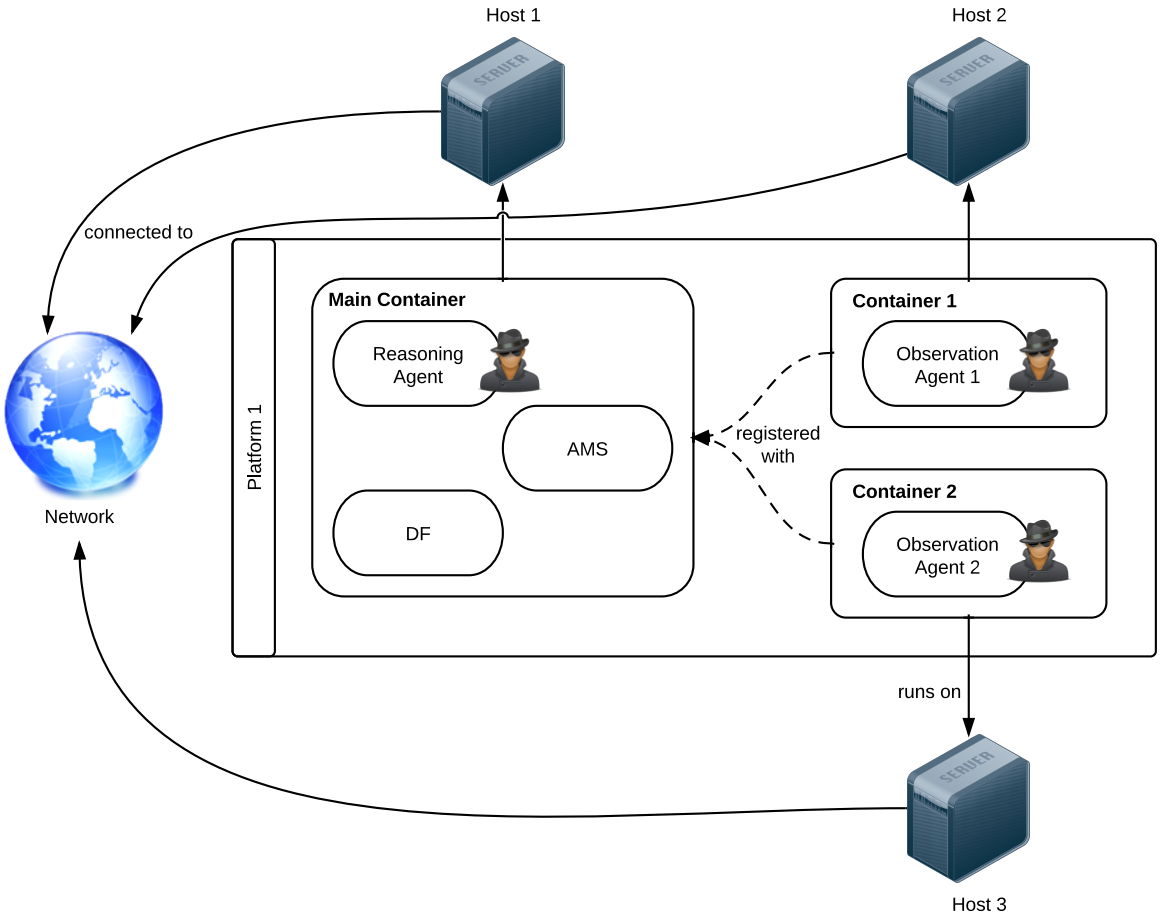

Figure 6.3: Structure of the JADE network.

Map Definition File The area is modeled as a matrix in a text file. The matrix contains the following integers:

- 0: building - cannot drive or walk here.
- 1: road - a person or car can drive or walk here.

The NetLogo project file loads this file and constructs the map accordingly. Zeros are mapped to the color green (grass) and ones are mapped to the color black (asphalt).

Object Configuration File The paths of the objects are contained in a configuration file. The best way to explain the structure of the file is to give an example. The definition of a car can look like this:

```
[audi]
class=car
position=8 29
heading=N
path=LRRL9TLS
```

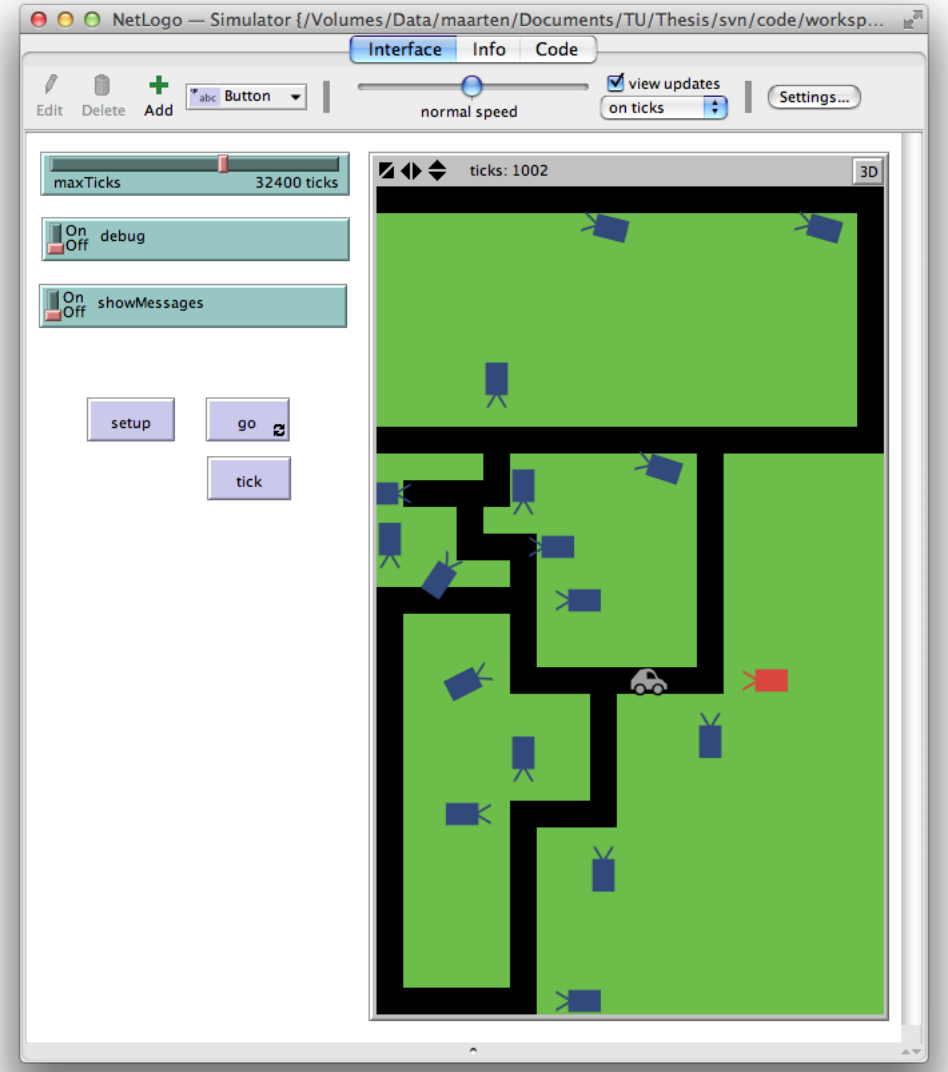

Figure 6.4: NetLogo project file with a simplified version of the NLDA map.

The first line includes the name of the object between square brackets. This name should be unique in this file, but is not (yet) used in the implementation. The second line defines the class, which can be 'car' or 'person'. The starting position of the object is given in the third line. This position is defined in world coordinates that originates in the lower left corner of the world. Finally, the fourth line defines the path of this object. The following tokens can be used (where n is an integer like '1' or '12002'):

- L: turn left at the next crossing.
- R: turn right at the next crossing.
- G: go straight ahead at the next crossing.

- S: stop.
- nW: stop and wait for n seconds.
- T: turn around.
- H: hide.

When an object stands still, a L, R or G makes it move again. If a number is used before a L, R, G, S, T or H, it means that the action must be executed after that amount of meters. For example, '9L2S4W' means: after 9 meters, turn left at the next crossing, then after 2 meters, stop and wait for 4 seconds. The action *hide* means that the object is temporarily not observable by the camera. With this action we simulate mistakes from the video processing and objects that are hiding intentionally.

The World of the NLDA area is 18x30 patches, which corresponds to the real size of 180x300 meters[1]. This configuration file is expected to contain world coordinates, because it makes the placing of the object much easier. Note that decimal numbers can be entered if an increased precision is needed.

Camera Configuration File A definition of a camera can look like this:

```
[camera1]
position=8 29
patches=1 30 8 31
```

The first two lines speak for themselves. The last line defines the 1-D row of patches the camera observes. Each path between those coordinates is added to a list and if an object moves over one of those patches, the camera observes the object. For the coordinates the same holds as for the object configuration file: using world coordinates make the configuration easier.

[1] Measured using Google Earth `http://earth.google.com/`.

7 Design: Structure of the Agents

Chapter 4 described the surveillance system model. This chapter will zoom in on the internal structure of the agents themselves.

7.1 Overview

The system model described in figure 4.3 consists of several agents. Figure 7.1 repeats this model, but here the agents are schematically displayed. The following sections zoom in on the structure of each agent.

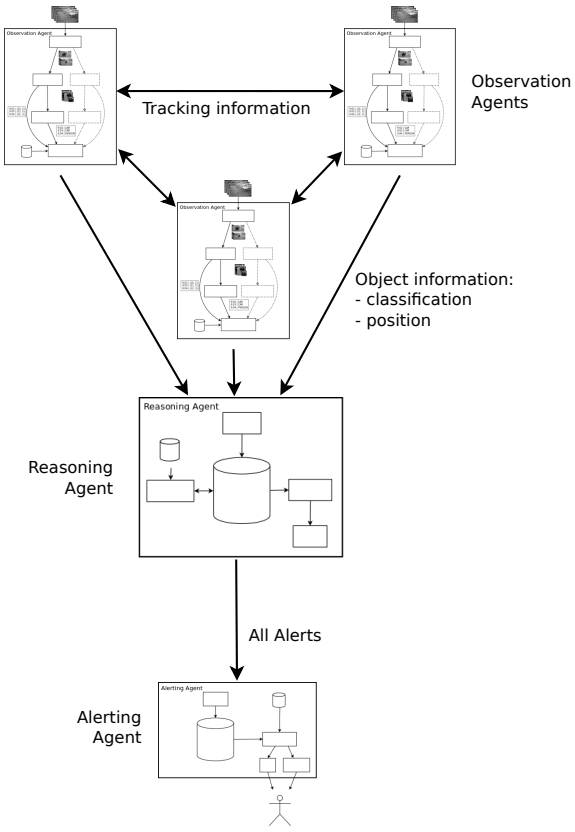

Figure 7.1: System Model with schematically displayed agents.

7.2 Observation Agent design

An Observation Agent processes the video that it receives from one camera. Its main task is to extract information about the objects that move around in sight of the camera. This information consist of the position of the object at each time instance and the classification: is it a car or a person? Figure 7.2 on the next page shows the structure of an Observation Agent. The following paragraphs describe the elements from this figure.

Input The agent receives a video stream from its camera.

Object Detector The Object Detector searches for moving objects, determines a bounding box and sends it to the Object Tracker.

Object Tracker The task of the Object Tracker is to follow the object in consecutive frames and to give the trajectory to the Communicator. A trajectory consists of a list with timestamped positions. In this example, a trajectory of three positions are shown. During the tracking process, the Object Tracker has collected multiple pictures of the object which is passed to the Object Classifier.

Object Classifier An Object Classifier determines what kind of object it is. In this example, object A is classified two times as a car and then as a person, which is of course a mistake of the classifier. This information is passed to the Communicator.

Communicator The Communicator combines the information and sends it to the Reasoning Agent. With predefined knowledge, the agent is able to determine if an object is headed to one of his neighbor Observation Agents. If that is the case, it sends the tracking information to its neighbor. This tracking information is useful for the Object Tracker of the other Observation Agent because it includes the different poses of the object that are observed by the first agent. The Communicator is a special module in the sense that it has access to all information inside the Observation Agent. For example, the tracking information which is sent to the neighbor agent is coming from the Object Tracker.

The dashed lines and modules represent the parallel tracking of a second object. The Object Detector is able to detect multiple objects in one frame. In this example two objects are detected: object A and B. Furthermore, the Object Detector is able to determine which object is being tracked by a tracker and which one is not. If not, a new tracker and a new classifier are initialized.

An aspect which is not shown in the figure is that the timestamped position and object classification is labeled with an internal object ID, in this case object A or B. This internal ID is useful for the Object Detector to keep track of which objects are present, but is also very useful for the Reasoning Agent. If one Observation Agent labels two objects to be object A and B, the Reasoning Agent trusts this judgment and will never conclude that object A and B are the same objects. This is because the Observation Agent is able to differentiate objects based on physical features while the Reasoning Agent is not.

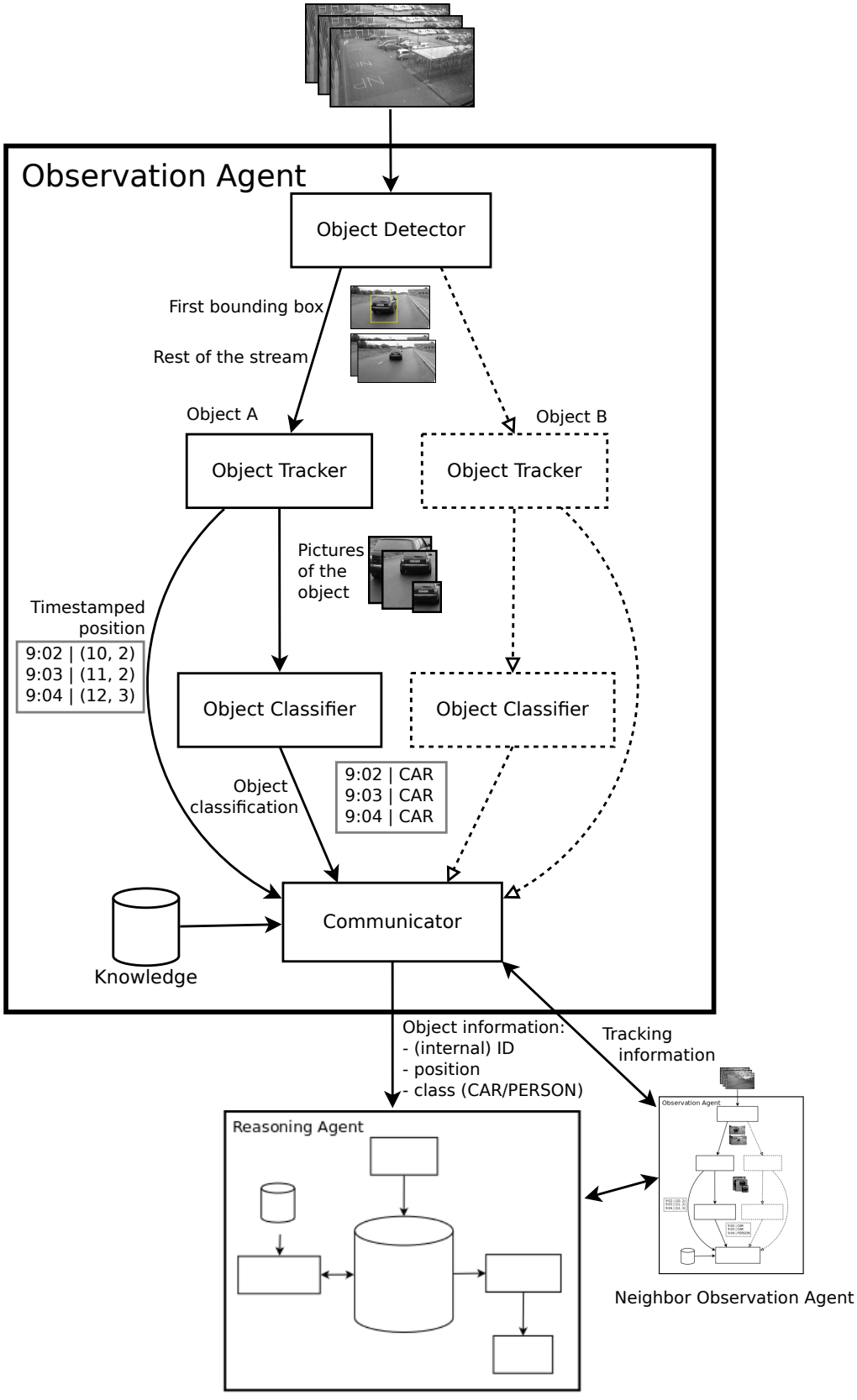

Figure 7.2: Structure of an Observation Agent.

7.3 Reasoning Agent design

The main task of the Reasoning Agent is to generate an alert in case of suspicious behavior. From the observations received from the Observation Agents, conclusions can be drawn upon the actions of the object. For example, the Reasoning Agent can determine if a car exceeds the speed limit or drives on an area where it is forbidden to drive. In section 3.3 six scenarios are defined which should be recognized by the system.

Most of the scenarios need predefined knowledge: it should be known where it is forbidden to walk, to stop or to park. Furthermore, information is needed about the positions of the cameras.

Figure 7.3 schematically shows the structure of the Reasoning Agent. A *Receiver* receives the object information from the Observation Agents and stores it in a central *Event Database*. Then *Interpreters* interpret this data and draw conclusions from what events have taken place exactly.

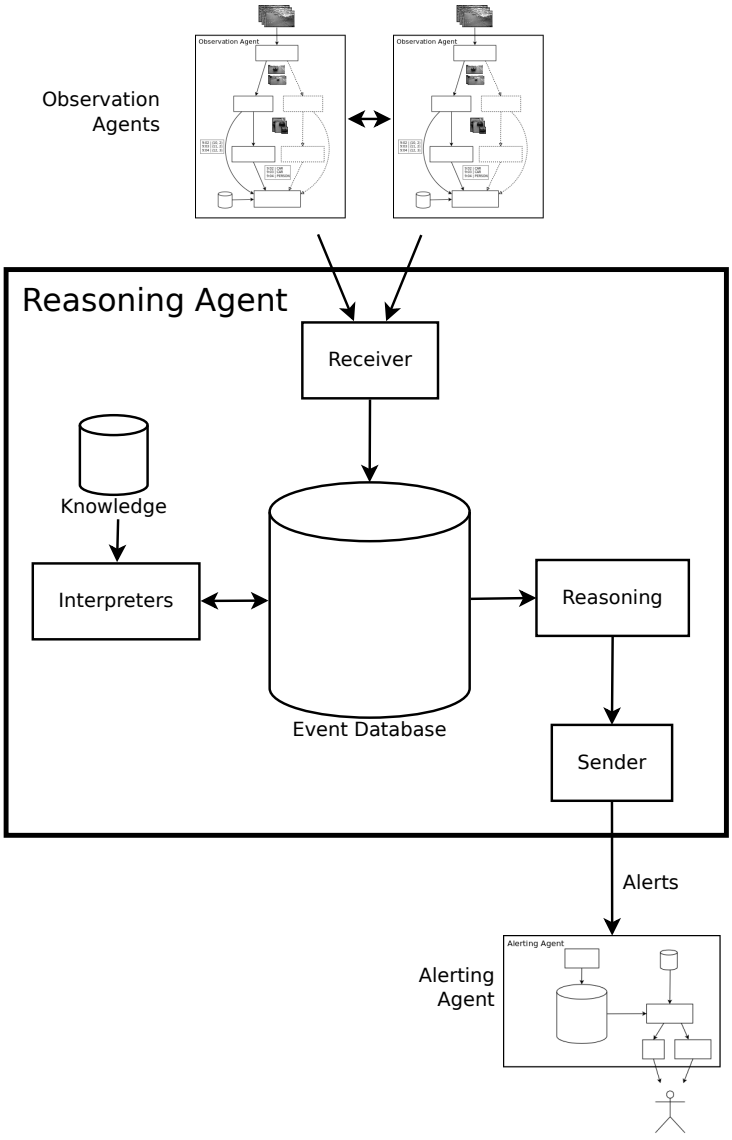

Figure 7.3: Structure of the Reasoning Agent.

For example: the object has been speeding or parked on a non-parking area. To draw these conclusions, predefined knowledge is needed like the speed limit and the ROIs. The *Reasoning* part reasons about the events and determines how suspicious each object is. When an alert needs to be send, this is done via the *Sender* to the Alerting Agent.

In order to provide more details of the Reasoning Agent, the software design is described here. The complete Reasoning Agent can be summarized by the UML diagram of figure 7.4. The function of the Receiver is assigned to the *ObservationWriter* class. The observations are written to the *Whiteboard*, which corresponds to the Event Database. The roles of the Interpreters are filled by the *IdentificationWriter* and the *ActionWriter*. They add more information to the *Whiteboard*. The *ReasoningEngine*, which encompasses the Reasoning part, reads the contents of the *Whiteboard* and then reasons about it. Conclusions (alerts) are sent to the Alerting Agent by the *ReasoningEngine*. The Sender is included in the *ReasoningEngine* for simplicity. The *BoardWiper* is introduced to limit the memory that Whiteboard occupies. The Whiteboard is required to have 30 minutes of data and the *BoardWiper* regularly cleans the old data. The rest of this section describes the most important parts of the design and gives more details.

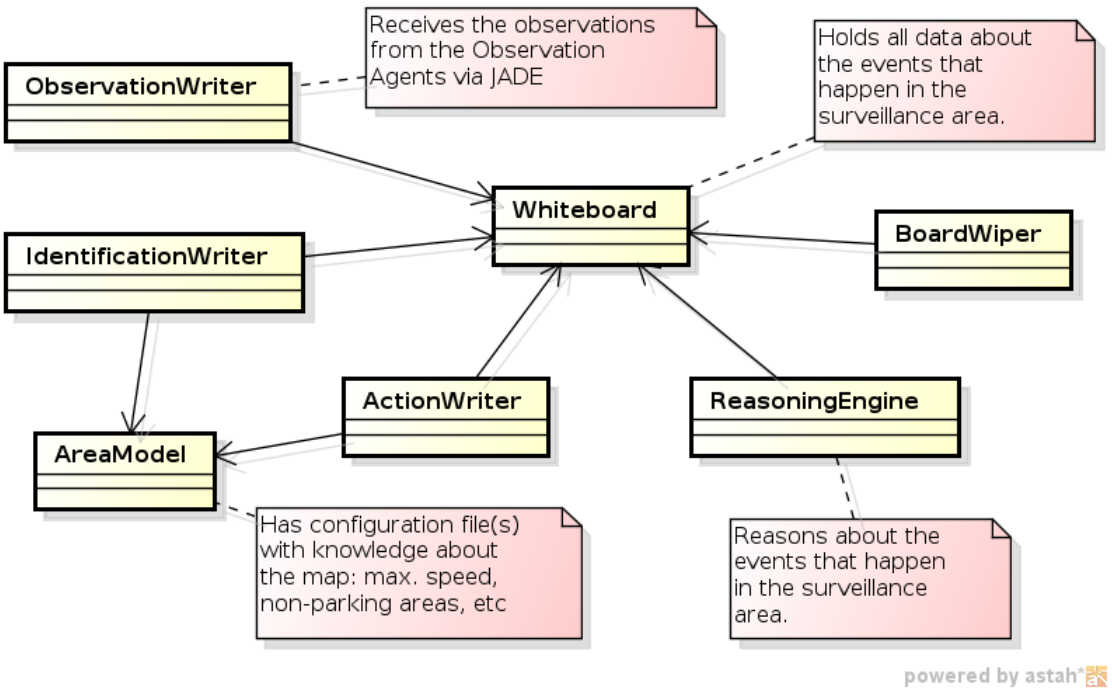

Figure 7.4: Reasoning Agent summary class diagram.

7.3.1 Whiteboard

This class gets it name from an element that is found in most offices: a white board. Occasionally, people use it to write or draw their ideas. Adjustments can be made by adding more information to it or removing some parts. This concept is used for the Reasoning Agent because we need a central place to save the data. Note that this concept is similar to the *Blackboard architectural model* used in the area of artificial intelligence.

Figure 7.5 shows an example of how information is added to the *Whiteboard*. Three stages

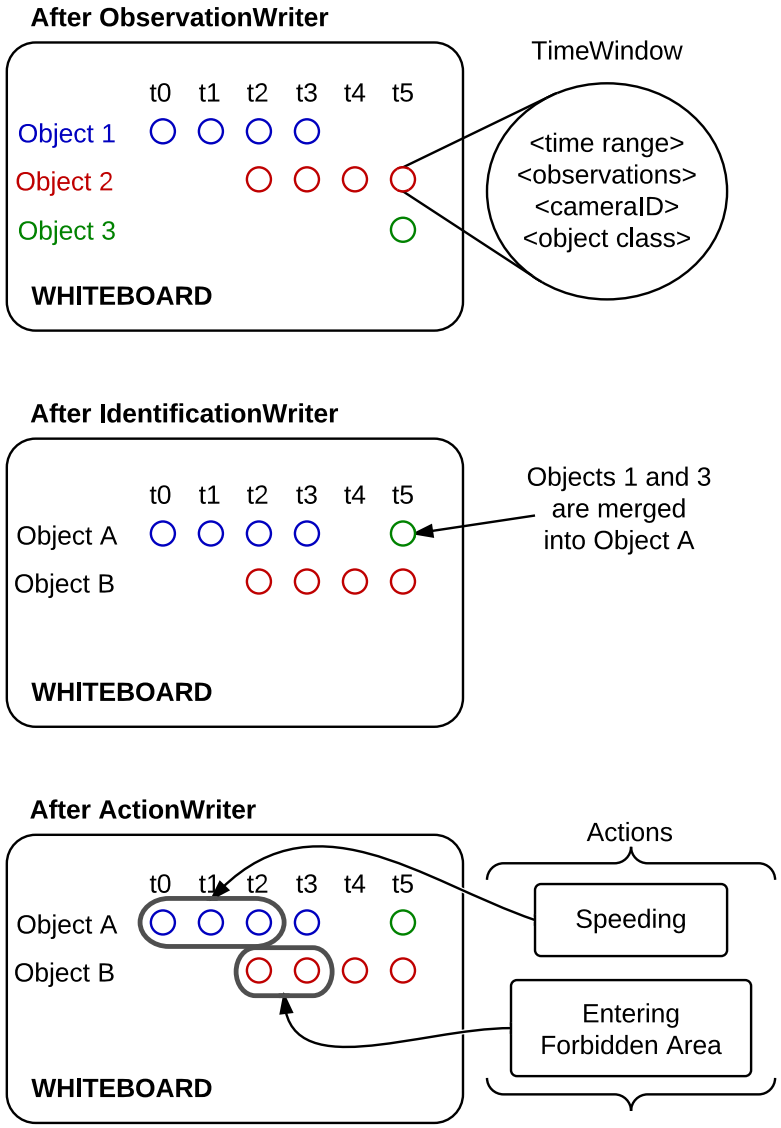

Figure 7.5: Illustration of the information flow on the Whiteboard.

are shown. The first stage illustrates the situation when the *ObservationWriter* has done its job. Six time steps are shown. Each time step takes two seconds. For each object and each time step, a *TimeWindow* object is created which holds all information for these two seconds for that particular object. A time range of two seconds is taken instead of the individual observations from the cameras, because it reduces the amount of information that has to be processed. For example the average speed can be determined, which can not be done for an individual observation. Three objects are shown, because the cameras identified three different objects. It is not shown which object is observed by which camera, so it is possible that all objects are observed by a single camera, by three different ones or something in between. At this point, the object ID for the ReasoningEngine is determined by a combination of the camera ID and the local object ID from the camera. In this case, all *TimeWindow*s with a blue color have observations with the same camera ID and local object ID from that camera (not shown in the figure). The red *TimeWindow*s

either have a different camera ID or the same camera ID and a different local object ID compared to the blue ones.

The second stage shows the situation after the *IdentificationWriter* analyzed the contents of the *Whiteboard* and has written the right identification to it. In this example, *IdentificationWriter* discovered that Object 1 and 3 actually are the same object and they are merged into object A. Object B contains exactly the same *TimeWindow*s as Object 2 in the first stage.

In the third stage, the result of the *ActionWriter* is shown. It analyzes each *TimeWindow* and adds *Action*s to it if a violation is detected. In this example, two actions are attached to five *TimeWindow*s.

To make this process run smoothly, some mechanism is needed to tell classes that the *Whiteboard* has changed. The Observer Pattern is used [16] as a model for this mechanism. This pattern describes the relationship between an Observer and Observable. The strength of the pattern is that the Observable has no knowledge about the Observers. The only requirement is that the Observers are added to the Observable and the Observers have a method that is called when the Observable has changed.

In the case of the Reasoning Agent, the *Whiteboard* is of course the Observable and the writers and readers are Observers. The standard Java library contains interfaces which make it very easy to implement this pattern. However, since the *Whiteboard* will be changed very frequently, it should be avoided that every class receives all updates. Therefore, a customized implementation is made. Every reader or writer is subscribed to a certain event. For example, if a new *TimeWindow* is written to the *Whiteboard*, the *Whiteboard* will only notify the writers and readers that are interested. This is illustrated in figure 7.6. It shows that the *IdentificationWriter*, *ActionWriter* and *ReasoningEngine* can be subscribed to the *MultiObservable* class for different Events using the method *addObserver(Event, WhiteboardObserver)*. An *Event* is an enumeration which will be explained in section 8.2. When the *Whiteboard* changes, the method *notifyObservers(Event, Object)* is called and the three aforementioned classes receive the update via the *update(Event, Object)* method they implemented. Note that the links between the three classes and the *MultiObservable* class are absent. This is because the classes are subscribed to the *MultiObservable* by another class. This has been done to make the classes as decoupled as possible. Compared to figure 7.4 it is noticeable that the *ObservationWriter* and *BoardWiper* are not shown. That is because these two classes are not interested in any changes on the *Whiteboard*. Also the links between the three classes and the *Whiteboard* are not shown, because this figure only illustrates how the classes receive their updates. Their link with the *Whiteboard* is related to the changes they make to that class and not the changes they receive.

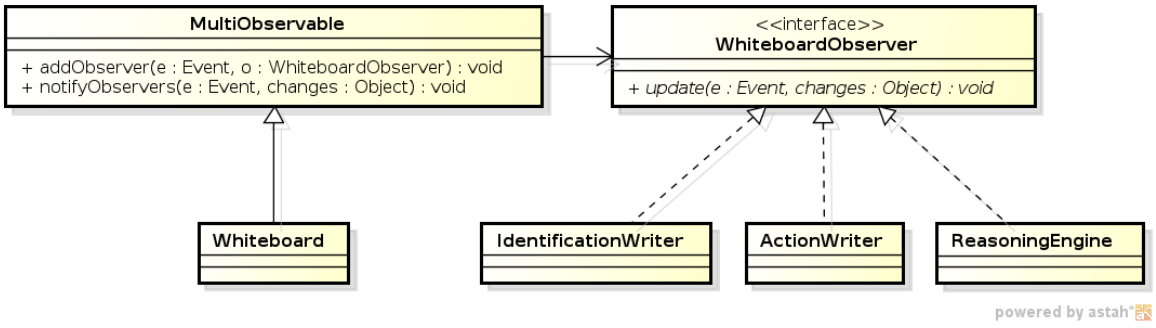

Figure 7.6: Class diagram that illustrates the Observer Pattern.

This raises the question how the writers make changes to the *Whiteboard*. If we would implement the *Whiteboard* like figure 7.4 suggests, the *Whiteboard* class would become a fat class and all classes would depend on a lot of public methods. The *Interface Segregation Principle* states that classes should not depend on *fat interfaces* . For our situation this means that the *Whiteboard* will implement several interfaces and the classes which use the *Whiteboard* are not given the complete *Whiteboard* with all the public methods, but instead a clean interface is given. In this way, the classes depend on just the small number of methods they really need. The interfaces are shown in figure 7.7.

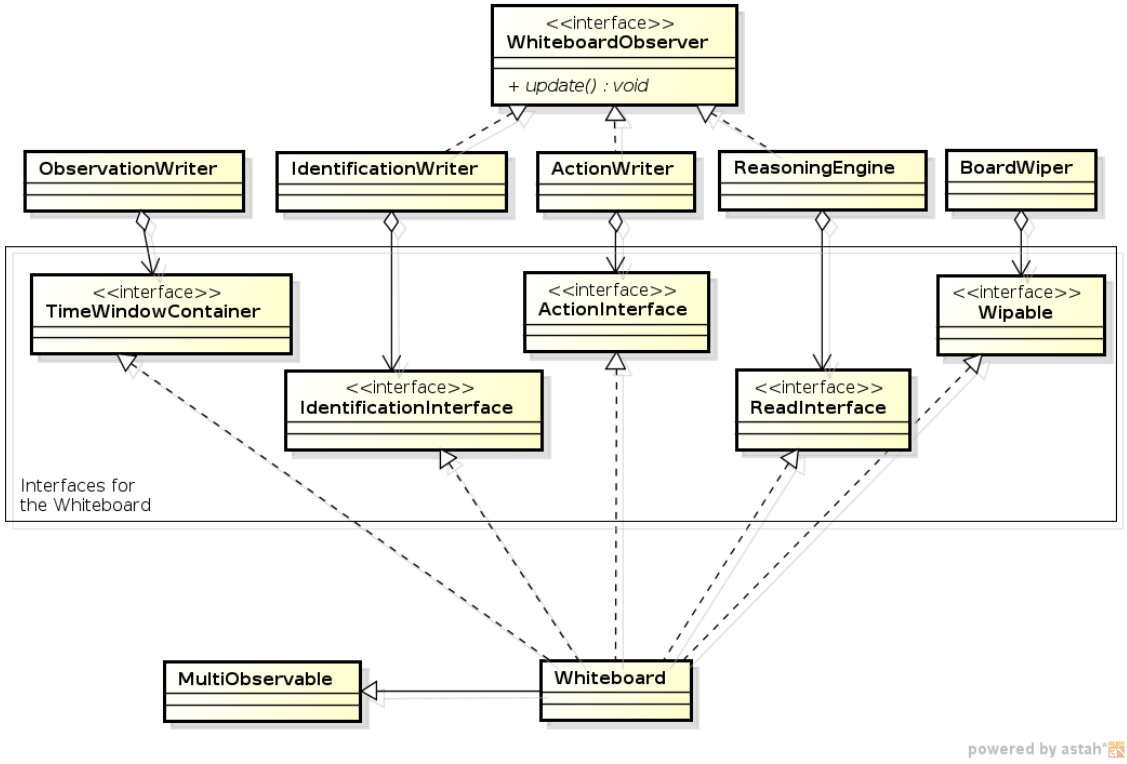

Figure 7.7: Class diagram that illustrates the interfaces.

7.3.2 Readers and writers

The ActionWriter reads from the Whiteboard and interprets the data. For example, it checks if the speed of the time window exceeds the speed limit. If so, it attaches an action to this time window. The area-specific data like a speedlimit is contained in the AreaModel.

The ReasoningEngine does not write to the Whiteboard, but only reads from it. Its task is to draw conclusions about the behavior of the objects. It builds upon the work of the Identification-Writer and the ActionWriter. It reads the object information and list of actions and fills a DBN for each different object. As described in 5.3.5, the Bayes Net Toolbox for MATLAB is used for these DBNs. In order to connect to MATLAB the tool Matlabcontrol[1] is chosen. This coupling from Java to MATLAB is designed such that it is relatively easy to replace the Matlabcontrol tool or the Bayes Net Toolbox. This design is shown in figure 7.8. The advantage of this design is

[1] `http://code.google.com/p/matlabcontrol/`

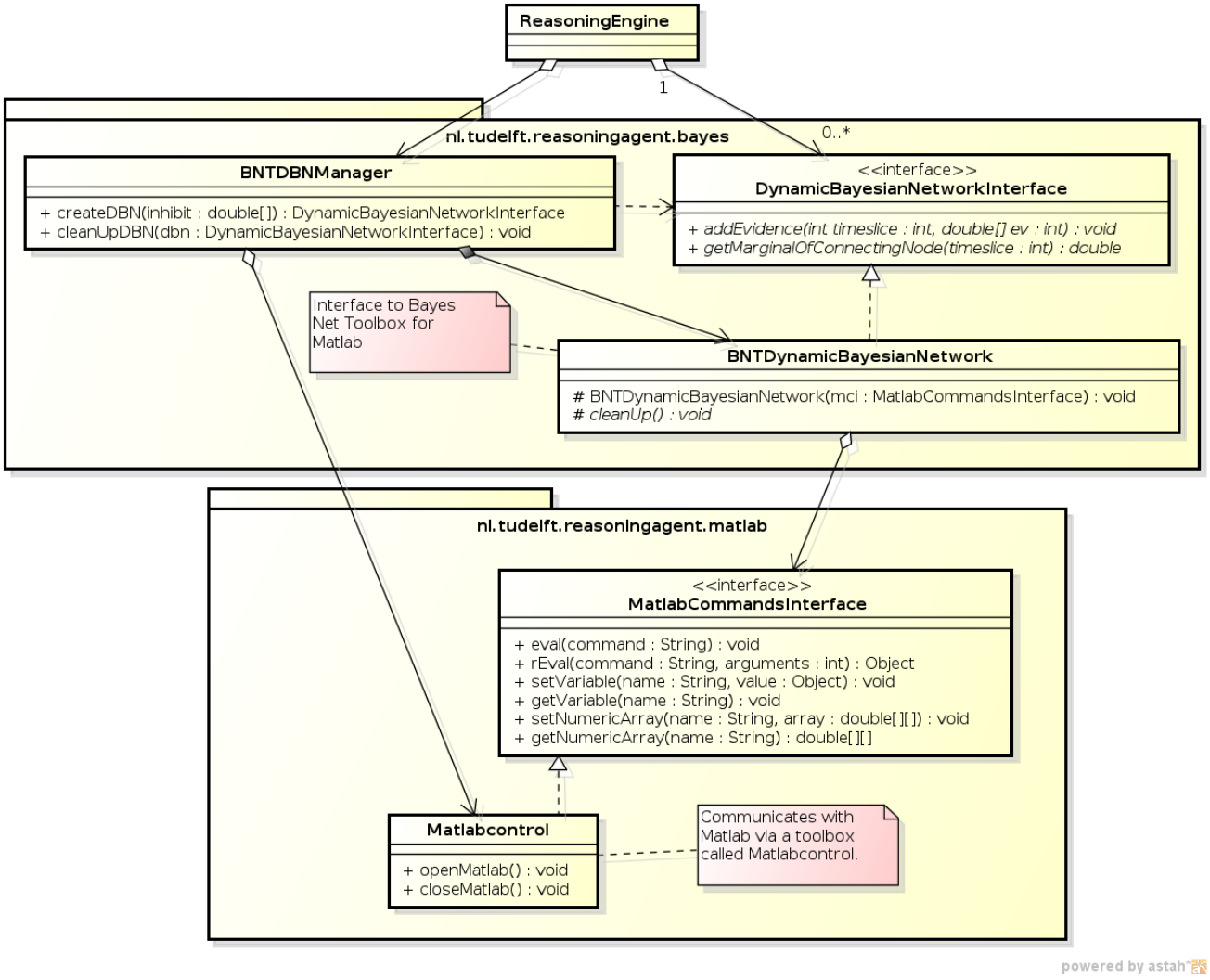

Figure 7.8: Class diagram that illustrates the coupling with BNT.

that the *BNTDynamicBayesianNetwork* could be replaced by a class that uses another MATLAB toolbox, as long as it implements the *DynamicBayesianNetworkInterface*. If in the future DBNs will be controlled by a toolbox that is not written in MATLAB, the class *Matlabcontrol* can be replaced by a class that controls this toolbox.

7.4 Alerting Agent details

The Alerting Agent receives alerts from the Reasoning Agent. The Alerting Agent is responsible for bringing the right alerts to the attention of the security guards. A simple Alerting Agent would just forward each alert to the Surveillance Base Station. Smarter Alerting Agents would sound an alarm for critical alerts, send normal messages for less critical alerts and repeat them if users do not respond.

'Sending messages' can be interpreted very broadly. It could send e-mails or SMS messages. Also the Alerting Agent can display messages using a web interface. Another useful output would

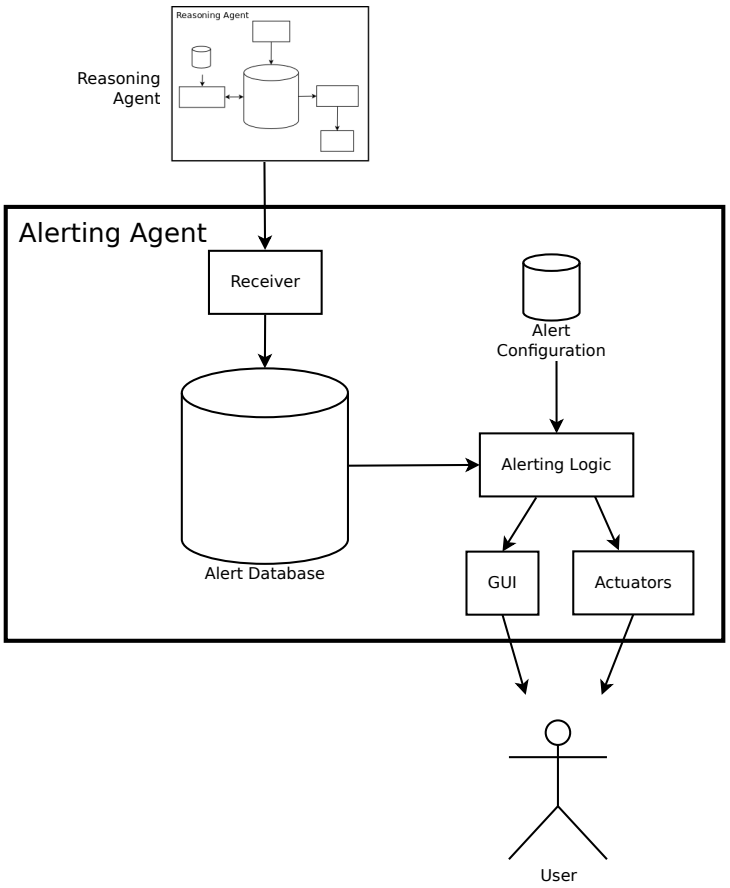

Figure 7.9: Structure of the Alerting Agent.

be an audible alarm.

The structure of the Alerting Agent is illustrated in figure 7.9. A *Receiver* receives all messages from the Reasoning Agent. They are entered in a central *Alert Database*. Then the module *Alerting Logic* uses logic and the *Alert Configuration* to determine if an alert must be forwarded. And if so, the *Alerting Logic* component decides if the alert is shown on the *GUI* or an *Actuator* is enabled. Via these two modules, the alert reaches the user. The main advantage of the Alerting Agent is that it knows about the alerts of the system via the Reasoning Agent and the user preferences via the *Alert Configuration*. The system could for example send the same alert multiple times per second, but the Alerting Agent can contain logic that makes sure that the user will be alerted with a lower frequency. Furthermore, the *GUI* can be created as such that there is a difference between high-priority alerts and low-priority alerts. Examples of an Actuator would be an alarm bell or a flashing light.

This agent is not implemented, so no class diagrams are made to design the Alerting Agent.

7.5 Summary

If one looks at the tasks that are prominent in the system, three main tasks can be identified as shown in figure 7.10.

Feature Extraction encompasses the extraction of position and class from the frame image,

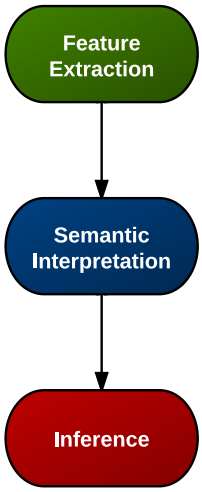

Figure 7.10: Main tasks in the surveillance system.

but also the speed and curvature that can be concluded from consecutive positions. *Semantic Interpretation* is the task of interpreting this class, speed and curvature to recognize suspicious behavior. The last task called *Inference* combines these recognized suspicious behavior over time and draws conclusions about them. One could consider the tasks *Semantic Interpretation* and *Inference* as the tasks that handle the reasoning.

Figure 7.11 shows how these tasks are distributed over the agents. The Observation Agents are obviously the first step in Feature Extraction: they observe objects in the environment and determine their position and class. When this object information reaches the Reasoning Agent, the ObservationWriter creates TimeWindows which calculate the speed and curvature for their Observations. Therefore, the first part of the Reasoning Agent is also colored green. The Semantic Interpretation is done by the IdentificationWriter and ActionWriter. The IdentificationWriter makes sure that the right observations are coupled to the right objects. The ActionWriter analyses the speed and curvature from the TimeWindows and conclude if they are parking, speeding, etc. If so, Actions are attached to the TimeWindow. Finally, the Inference is performed by the ReasoningEngine: it reads the actions from the TimeWindows and adds evidence to the Dynamic Bayesian Networks it possesses.

Table 7.1 gives a summary of the information flow throughout the system. It is shown how the data of one Observation Agent is interpreted by the Reasoning Agent and it reveals the inner workings of the agents.

The Observation Agent observes the camera images and sends a tuple to the Reasoning Agent. This tuple consist of time t_i, internal object ID o_i, object position p_i and class c_i. The ObservationWriter writes this information to the Whiteboard in the form of TimeWindows w_i for $i = 1, 2, 3$. Each TimeWindow calculates its speed and curvature. The IdentificationWriter interprets this information and checks if the two objects that the Observation Agent identified are really different objects. Then the ActionWriter applies its rules to determine the actions of a TimeWindow. Three rules are shown. The ReasoningEngine reads those actions, feeds evidence to the DBNs and reports the new suspicious rate of each object to the Alerting Agent. If this rate exceeds a certain threshold, the Alerting Agent will alert the operators.

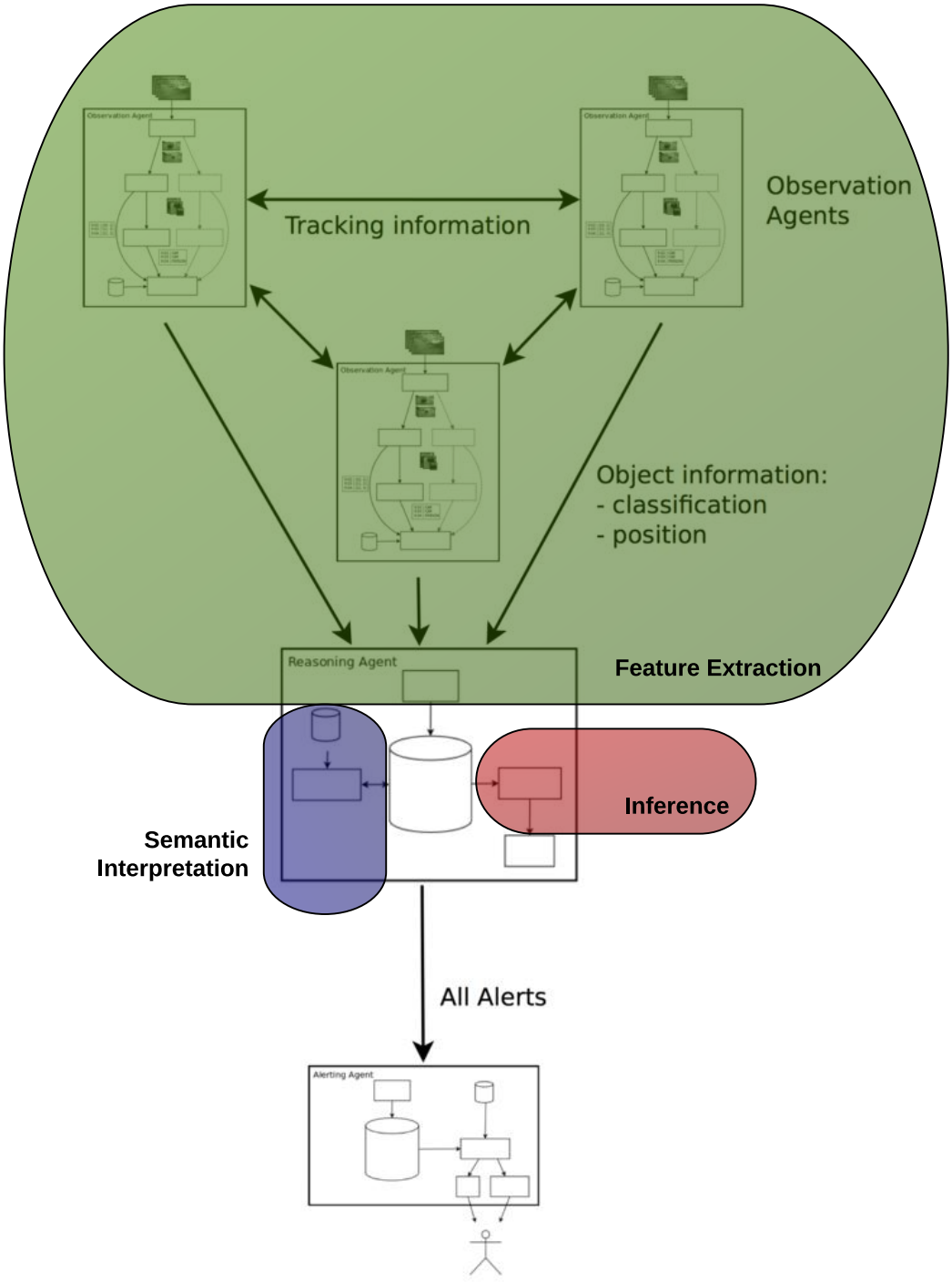

Figure 7.11: Distribution of the tasks among the surveillance system.

Table 7.1: Summary of the information flow in the system.

Observation Agent	**Reasoning Agent**				**Alerting Agent**
	ObservationWriter	IdentificationWriter	ActionWriter	ReasoningEngine	
(t_1, o_1, p_1, c_1) (t_2, o_1, p_2, c_2) (t_3, o_1, p_3, c_1)	TimeWindow w_1		if c_i == car and speed > limit → SPEEDING	evidence for DBN → suspicious rate	if suspicious rate > threshold, alert
(t_4, o_1, p_4, c_1) (t_5, o_1, p_5, c_1)	TimeWindow w_2		if $\exists\ p_i \in$ forbidden area → ENTERING FORBIDDEN AREA		
(t_6, o_2, p_6, c_1)	TimeWindow w_3	if $o_1 == o_2$, merge w_2 and w_3			
	For each TimeWindow: $\text{speed} := \frac{\sum_{i=1}^{N-1} \text{distance}(p_i, p_{i+1})}{t_N - t_1}$ curvature := ...		if c_i == car and $\exists\ p_j \in$ non-parking area and speed ≈ 0 for the previous 60 seconds → PARKING ON NON-PARKING ZONE ...		

8 Implementation

As mentioned in section 1.5, the implementation is divided in a single camera part and a multi-camera reasoning part. Section 8.1 describes the implementation details of the Observation Agent, which represents the single camera part. Section 8.2 describes the implementation details of the Reasoning Agent that belongs to the multi-camera reasoning part. Section 8.3 covers the choices made regarding JADE. Section 8.4 described how tests are performed during the implementation and section 8.5 closes with a list of features that would be a valuable addition to the implementation.

8.1 Observation Agent

As described in section 7.2, the Observation Agent consists of four modules called *Object Detector*, *Object Tracker*, *Object Classifier* and *Communicator*. The *Object Tracker* is taken as a basis of the Observation Agent. It is extended with several functionalities we will describe the following subsections. The module *Communicator* is not implemented.

8.1.1 Object Detector

For Object Detection we chose to implement Background Subtraction , because it quickly gives reasonable results. For this technique, a picture of the scene background picture should be available. This means that we assume the cameras are static. Furthermore, lighting variations might cause problems in the future.

This is how the Background Subtraction is implemented:

```
function detect_object
begin
  while ¬end of video stream do
        frame := get_next_frame(); // gets next frame of the video stream
        bb := get_bounding_box(frame);
        if bb not empty
          return bb;
        end
  end
end
```

```
function get_bounding_box(frame)
begin
   bg := load_background_image(); // loads background image from disk
   difference_image := bg − frame;
   return bounding box around largest object in difference_image;
end
```

In essence, the function `detect_object` waits for an object to appear and returns the bounding box if it perceives the object.

The object detector currently does not support the detection of multiple objects, because the Object Tracker (described below) does not support multiple objects either. When support will be added in the future, it means that the algorithm needs to make bounding boxes for the largest n blobs instead of just the largest, where n is the number of objects with sufficient size.

8.1.2 Determine Object Position

If the Object Tracker has determined the bounding box, we can use it to determine the position of the object. With the position of the object we mean where the object is located in the real world. Before this can be done, a map of the surveillance area is needed. To map a point on the camera image to the real world, a perspective mapping is performed. The user has to define four points on the camera image and supply the coordinates in the real world. These four points and their real world coordinates are saved to a text file called `perspective.txt`.

This is how determining the object position is implemented. To initialize, the following code is executed:

```
perspectiveFilename = [tld.source.input 'perspective.txt'];
if exist(perspectiveFilename, 'file')
    transformMatrix = get_perspective_matrix(perspectiveFilename);
end;
```

The field `tld.source.input` contains the path to the input video sequence. If the file `perspective.txt` exists, the perspective transformation matrix is determined using the self-written function `get_perspective_matrix`. This function uses the MATLAB function `maketform` and is not interesting enough to show here. For each frame, code from figure 8.1 is executed.

This piece of code appends the position of the object in the current frame to the file `trajectory.txt` in the output folder. The if-statement checks if the save-trajectory feature is enabled. The variable `tld` is the global struct that is used throughout the Predator code. We extract the bounding box from this variable, determine the center and determine the position (x,y), which is defined as the point on the center of the lower bounding box edge. Then the real world position (u,v) is determined using the MATLAB function `tformfwd` and the transformation matrix. If no transformation matrix is initialized because `perspective.txt` was absent, only the position on the image is taken. The position is appended to the trajectory file using the MATLAB function `dlmwrite`. An example of the contents of `trajectory.txt` is shown below.

```
26 277.47 259.47 -1.3068 34.489
27 277.88 259.81 -1.2962 34.455
```

```
% Save the trajectory of the object.
if isfield(tld.control, 'save_trajectory') && tld.control.save_trajectory == 1
    % Get the bounding box coordinates.
    bb = tld.bb(:,tld.source.idx(i)); % Watch out: this vector can contain negative values or NaNs.
    % Get the center of the bounding box.
    center = bb_center(tld.bb(:,tld.source.idx(i)));
    % Determine x and y.
    x = center(1);
    y = bb(4); % Take the Y from the lower right corner of the bb.

    % If the transformMatrix variable exists, transform (x,y) to global
    % coordinates (u,v) using the tranformation matrix determined
    % above.
    if exist('transformMatrix', 'var')
        [u,v] = tformfwd(transformMatrix,x,y);
        trajectory = [i x y u v];
    else
        % Just save the (x,y) local image coordinates.
        trajectory = [i x y];
    end;

    % Write the trajectory to disk.
    dlmwrite([tld.output 'trajectory.txt'],trajectory,'-append','delimiter',' ');
end
```

Figure 8.1: Piece of MATLAB code that is executed for each frame.

One requirement is that the camera does not move during the frame sequence. This piece of code saves the trajectory to disc, but with the same ease it can be saved to memory so that the Communicator module can send it to the Reasoning Agent.

8.1.3 Object Classifier

In section 6.1 is described how the Object Classifier is trained. This classifier is coupled with the Observation Agent by a single function called *classify_object*, which is shown in figure 8.2.

A picture of the object is given as parameter. If the classifier (model) is not loaded yet, some settings are loaded using the function *load_classifier_settings* and the model is loaded. Finally, the class and score is determined by calling the function *classify* that is built into the classifier during training. The meaning of the score variable is not found. It was expected that it would indicate the classifier's certainty, but occasionally the number is negative or NaN (Not a Number).

8.2 Reasoning Agent

The design of the Reasoning Agent is described in section 7.3. This section will summarize how the Reasoning Agent is implemented and it will highlight the most important implementation decisions.

The core of the Reasoning Agent is the Whiteboard: it contains all relevant data and most components read from or write to this class. During the initialization of the Reasoning Agent, the readers and writers are registered to the Whiteboard as shown in figure 8.3.
The first two lines create a Whiteboard and an ActionWriter. Note that the ActionWriter constructor expects an ActionInterface, which is implemented by the Whiteboard. The third line adds the ActionWriter (that implements the WhiteboardObserver interface) to the Whiteboard as observer for the OBJECT_MUTATION event. This piece of code could be shorter if the Action-

```
function [className, score] = classify_picture(im)
%CLASSIFY_PICTURE   Classifies the picture using a previously
%trained classifier.
%   The classifier is asumed to be trained using train_classifier. This
%   function uses the VLfeat library, so it is assumed that vl_setup is
%   already executed.
%
%   [C, S] = CLASSIFY_PICTURE(I) classifies image I and outputs classname C
%   and score S.
%
%   The code of this function is mostly copied from the PHOW_CALTECH101
%   example application provided with the VLfeat library.

global conf;

% Load settings and model (if not already loaded).
if ~exist('model','var')
    load_classifier_settings();

    % Load model.
    load(conf.modelPath);
end;

[className, score] = model.classify(model, im);

end
```

Figure 8.2: Source code of the MATLAB method classify_object.

```
Whiteboard whiteboard = new Whiteboard();
ActionWriter actionWriter = new ActionWriter(whiteboard);
whiteboard.addObserver(Event.OBJECT_MUTATION, actionWriter);
```

Figure 8.3: Registration with the Whiteboard.

Writer would add itself to the Whiteboard. However, it does not know about the Whiteboard, because it received an ActionInterface via this constructor. That is exactly why we initialize the ActionWriter in this way: it has no knowledge of the whole Whiteboard, but only about the limited set of methods it uses.

The following piece of code shows a very simple method that shows how an Event is generated inside the Whiteboard.

```
@Override
public void addTimeWindow(TimeWindow tw) {
    setChanged(Event.TIMEWINDOW_ADD);
    notifyObservers(Event.TIMEWINDOW_ADD, tw);
}
```

This method is called by the ObservationWriter if it constructed a new TimeWindow. The first line registers that something has changed that needs to be updated to the observers. The second line really notifies the observers about this new TimeWindow. Note that the Whiteboard does not save the TimeWindow, although that might be a logical step. With the current implementation, the Whiteboard does not need the TimeWindow in any point of time. Other Whiteboard methods are more complicated in the sense that they have to perform some actions on the incoming object

before they notify the observers.

8.3 Agent communication

8.4 Testing

Several tests are done during the implementation. Because of the nature of each agent, tests are done in different ways.

Before the Observation Agent existed, several tests are done with Predator. Datasets are downloaded or captured and fed to the tracker. Because the Object Detection was not implemented yet, the first frame had to contain the object and a bounding box must be defined. The main purpose of those tests were to see how Predator works and to see if it performs well enough.

During the development of the Observation Agent, some tests are done using debug lines. For example, when developing the Object Detector, a debug line printed how many different objects were found in the image and what the size of the largest object is. Other debug lines produced an image. These tests showed that a certain function works, but also helps in the configuration of the function. Sometimes a small MATLAB file is created that tests a certain functionality separately from the rest of the agent.

The most important tests with the Observation Agent is starting it and watch the result. Several different datasets are given to the Observation Agent to see how they are handled. A subset of those datasets is described in section 5.6.1 and the results are shown in section 9.1.

The Reasoning Agent is of course evaluated by the Simulator, but unit tests are created during the development to assure that individual units work properly. All unit tests are created using the JUnit testing framework[1]. Every relevant class has a unit test class which tests the relevant methods. This results in 19 tests spread over 8 test classes.

Most tests of the JADE classes are very simple: an attempt is made to create a JADE network, a message is sent and the receiver print this message.

8.5 Future Implementation

The previous sections described the performed implementations. This is a list of pieces that are not implemented, but would be useful for the system in the future:

- Video saving/handling: the tracker (and therefore, the Observation Agent) reads still images, while the cameras produce video streams. So software is needed to either enable the tracking to read streams or convert the streams to single images. In the latter case, the tracker still needs to be adjusted so it keeps checking the disk for new frames.
- Multiple object tracking: this requires the tracker to be extended. It uses one global variable to save everything, so this variable needs to be able to hold information of multiple objects. Furthermore, adding and removing objects should be implemented. Finally, the Object Detector module and the IdentificationWriter class need to be adjusted.
- Pan-Tilt-Zoom support: to expand the area that is under video surveillance without installing significantly more cameras, Pan-Tilt-Zoom cameras can be a great solution. However, the background subtraction algorithm as well as the perspective transformation to determine

[1] http://www.junit.org/

the object's position assume a fixed camera. However, if these two Observation Agent issues are solved, the system is able to handle Pan-Title-Zoom cameras.

- Section 3.1 stated that we require the system to have a frame rate of at least one frame per second. However, when something suspicious happens it might be interesting to temporarily increase the frame rate in order to supply the Reasoning Agent with more detailed information.
- Communication with the Observation Agent: currently, the simulator is able to communicate with the Reasoning Agent via JADE. However, the Observation Agent itself is not able to communicate at all. This is assumed to require little effort, since the JADE Java classes are already implemented for the Reasoning Agent and MATLAB supports calling Java classes.
- Alerting Agent: this agent is not implemented, as alerts are now just printed to the console. An alerting agent can manage the way alerts are presented to the Surveillance Base Station.
- Smarter Reasoning: the Reasoning Agent has a basic reasoning network, but it can easily be extended to do more complex reasoning. For example: more nodes can be added, continuous nodes nodes can be introduced and more actions can be invented.
- Object matching in the Reasoning Agent: currently the Reasoning Agent only can identify objects based on the naive assumptions that every observed object is already known, unless it is proved otherwise. It would be very useful if it could receive the object information from the Object Tracker and still images of the object. In this way, the agent can also distinguish objects based on their visual properties. Furthermore, identifying non-physical features would improve the system even more. To give an example: imagine two marines walking down the street. They look very similar due to their uniforms. However, the style of walking might be a feature that distinguishes the two. One significant advantage is that this feature is not camera-specific.

9 Experiments

This chapter describes the experiments that show the performance of the implemented prototypes. Section 9.1 describes the experiments done with the modules of the Observation Agent and section 9.2 describes the Reason Agent performance. Both sections describe how the experiments are executed and their results.

9.1 Observation Agent

As described in section 8.1, three modules are implemented: *Object Detector*, *Object Tracker* and *Object Classifier*. These modules are tested separately, since they have distinct tasks and are dependent on each other. The experiments are described below.

To test the modules of the Observation Agent, input is needed. Therefore, several videos are recorded that function as input datasets, as described in section 5.6.1.

9.1.1 Object Detector

Setup An Object Detector performs well if it detects most objects that are visible on the video. Since our implementation is based on background subtraction, we only consider 'new' objects: moving objects that were part of the background are avoided in the datasets. Furthermore, the Object Detector is expected to focus on the largest objects if multiple objects appear in the frame. Finally, given the implementation, we expect the Object Detector to detect objects that are at least two pixels away from the boundary of the frame.

Now that we know the context, we can describe the experiment. Most datasets start with at least the first frames without any objects. For those datasets, the first frame is labeled that contains an object that is at least two pixels away from the frame border. Then the frame sequence if fed to the Object Detector and it prints which frame is the first one that includes the full object. The (absolute) difference between the true and measured frame numbers is taken as performance measure.

Another property that can be considered is the quality of the bounding box. The Object Detector performs very good if the resulting bounding box is the smallest square around the object without excluding parts of the object. If a bounding box is chosen too large or too small, the performance is worse. However, we did not test this property because we limit ourselves to basic tests in this thesis.

Results Table 9.1 shows the results of the Object Detector experiments performed on the five datasets described in section 5.6.1. The result for dataset 3 is missing, because the object appeared in the middle of the frame. Of course, the object was detected, but it is difficult to define a ground truth.

Table 9.1: Results of the Object Detector experiments.

Dataset	Ground truth frame #	Detection frame #	Absolute difference
1	63	68	5
2	99	111	12
3	-	-	-
4	8	9	1
5	20	17	3
Mean			**5.25**

The difference of dataset 2 seems large: 12 frames. But the camera used to record the datasets had a frame rate of 25 frames per second. So in this case, the tracking started 12 / 25 = 0.48 seconds later than it should. This result is acceptable, especially when keeping in mind that the frame rate is required to be 1 fps and a normal system can process about 8 fps. Considering this analysis, the mean difference of 5.25 is a good result.

9.1.2 Object Tracker

Setup Kalal et al [14] compared the performance of Predator to other algorithms "in terms of the frame number after which the tracker doesn't recover from failure"[14]. Their purpose of this first experiment was to show that Predator outperforms other algorithms thanks to its learning features. The second experiment in that paper shows the internal performance of Predator which proves the advantage of the learning algorithm. In contrast, our purpose is to determine if Predator is suitable for a surveillance system. A disadvantage of their approach for the first experiment is that false positives are not taken into account: since each frame of the dataset includes the object, Predator is not being tested on frames where the object is absent. Therefore we introduce another way of measuring Predator's performance: the number of frames that Predator is correct. We define correct as: *Predator tracks (part of) the target object*. So incorrect would be if Predator starts tracking another object or the background. For our application, we are mainly interested in the position of the object. Therefore, we think it is not a problem if Predator tracks part of the object. Nevertheless, it will be mentioned in the results when it happens, because it might cause problems for the Object Classifier.

During this experiment, the Object Tracker receives a dataset with the object on the first frame and a (correct) bounding box around the object. Then we count the number of wrong frames and divide it by the total number of frames to calculate the error.

Results Table 9.2 shows the results of the Object Tracker experiments.

The error for dataset 1 is .0000, but about 61 frames show a bounding box that is significantly larger than the object: less than 50% of the bounding box is filled by the car.

The large number of wrong frames for dataset 2 is caused by the fact that the tracker starts tracking a part of the road when the person almost reaches its car. This phenomenon is observed for dataset 5 too and also datasets not described in this report show this weakness of Predator. A possible explanation is that Predator has learned too much about the background (road) that causes it to track the background when the actual object is getting smaller. Note that the error of

Table 9.2: Results of the Object Tracker experiments.

Dataset	Wrong frame count	Total frames	Error
1	0	90	.0000
2	228	553	.4123
3	0	469	.0000
4	0	142	.0000
5	77	473	.1628
Mean			**.1150**

datasets 2 and 5 most probably would be higher if the frame sequence is extended. Similarly, the error would be lower if the tracker is stopped when the object is not tracked correctly any more. Of course, Predator has its own mechanism to do that, but it might be possible to improve the performance in the future by using the data from the Object Detector.

Furthermore, during several experiments, the tracker does not smoothly track the object: sometimes it does not move, while the object does. Most of the times it recovers after a few frames and continues tracking the moving object.

Note that the datasets are captured with a frame rate of 25 fps. For future experiments, it would be useful to test Predator using frame rate used in the system, which would be between 1 and 8 fps. It is expected that Predator will not perform worse, because several demo movies on the Internet show that Predator is able to cope with rapid movements and shaking cameras. We might even observe a better performance when using a lower frame rate, because it is possible that Predator does not learn too much about the background any more.

The result is reasonable good: three datasets are correctly tracked and two datasets perform worse because of the background tracking problem. In the future this problem might be tackled using a lower frame rate and the information from the Object Detector. The beginning of dataset 5 nicely showed that Predator is able to learn: the first couple frames only tracks the front part of the person, but after a while Predator has learned how the full person looks like. Overall, Predator is considered as a good tracker to determine the object position.

9.1.3 Object Classifier

Setup Testing the Object Classifier brings some challenges. Feeding it with the contents of the bounding box of the tracker might give bad results if the tracker fails. Even more, imagine that the Tracker tracks a part of a car. Then the Object Tracker succeeds, but the Object Classifier most likely fails. A possible solution is to manually cut out the objects in case the Object Tracker fails or tracks part of an object. However, this can potentially take a lot of time. Furthermore, there might exist a difference between the way the Object Tracker 'cuts' the object from the frame and the way humans do that, which will influence the results in a negative way.

Another approach would be to make a test set of images from all kinds of sources. Actually, this is done to test the quality of the classifier, described in section 6.1. However, here we tested the performance of the classifier on the images that are similar to to the training images. We could download new images and see how the classifier performs, but that would indicate the performance of the new images instead of the output from surveillance cameras.

Therefore, another approach is taken. For each dataset, the contents of the bounding box is (automatically) extracted, which gives a series of pictures of the object, as long as the Object Tracker did a good job. Then only the *right pictures* are fed into the Object Classifier. 'Right picture' is defined as: *a picture mainly consisting of a large portion of the current object*. So a picture of a person and a tree is incorrect, since it should mainly contain the person. And a picture of the torso of a person is also incorrect, because a large portion of the person should be visible. Most pictures will contain a low number of pixels, but this should not influence the judgment of a 'right picture', as we consider the performance of the Object Classifier in the context of video surveillance.

Results Table 9.3 shows the results of the Object Classifier experiments.

The first observation is that the results are bad in general: on average, more than 61% of the pictures are wrongly classified. That means that the performance would be better if we randomly choose between cars and people.

The best result is gained with dataset 4 (about 7% wrongly classified). A close look to the pictures reveals that the classifier was able to classify it as a car, although the pictures become as small as 8x7 pixels. When looking at the results from this dataset, it appears that the last 63 classification scores are all NaNs. The meaning of these scores is not documented, but it might give an indication that the classifier was unable to classify the picture. These 63 pictures are 23x21 pixels or smaller. Regardless of the NaN, the pictures are correctly classified as a car. That is why the result with dataset 4 is very good. It is suspected that the classifier has a small bias in favor of the cars. When examining the results of dataset 5, it is observed that all wrongly classified frames have a score of NaN and are classified as cars. The only frame that has a valid score is classified correctly. Furthermore, this picture is the largest one with a dimension of 22x56. It is observed that all pictures of dataset 4 and 5 with a NaN score have a width or height lower than 22 pixels. The other datasets contain zero NaNs. We assume that the classifier is not able to classify pictures with a width or height lower than 22 pixels and that these pictures are always classified as cars.

Section 6.1 described the training process of the classifier. An accuracy of 83.33% if gained with the test set, which corresponds to an error of 0.1667. When using other data, an error of 0.25 or 0.3 is expected, so the mean error of 0.6113 is surprisingly large. We could have a look at the datasets 1, 2 and 3 that did not have any NaN scores. The mean error of those datasets is 0.6627. So this still indicates that the classifier performs badly. However, we can not draw firm conclusions, since we only looked at five datasets of which three have accurate results. Although

Table 9.3: Results of the Object Classifier experiments.

Dataset	Class	Wrong classification count	Total pictures	Error
1	Cars	69	89	.7753
2	People	97	163	.5951
3	Cars	257	416	.6178
4	Cars	10	141	.0709
5	People	382	383	.9974
Mean				**.6113**

the classifier has seen 1192 pictures in total, essentially it has seen five different images with small variations.

9.1.4 Conclusion

The Observation Agent performs well with Object Detection and Object Tracking. The perspective mapping described in section 8.1.2 is not tested, but the detection and tracking give a good basis to deliver accurate position information. The Object Detector performance is bad and should be improved by using a better dataset or another tool. Also more real data is needed to make firm conclusions about the performance.

9.2 Reasoning Agent

This section describes the experiments performed with the Reasoning Agent using the simulator, as shown in figure 9.1. The process of testing the Reasoning Agent actually also involves configuration of the agent. In order to let configuration and testing not be intertwined, a calibration is described in section 9.2.1 that performs a few tests and allows configuration and tweaking. Section 9.2.2 validates if the Reasoning Agent acts as expected in other situations, given that configuration. Section 9.2.3 closes with a conclusion.

9.2.1 Experiment 1: Calibration

The parts of the Reasoning Agent that are being configured during the calibration are the weights of the Dynamic Bayesian Network and the threshold at which we decide that the object is suspicious. While configuring, the suspicious rate is monitored to speed up the process.
Before describing the calibration experiments, some parameters and properties are enumerated:

- The speed limit is 50 km/h (for cars).
- A person is considered *running* when its speed is higher than or equal to 9 km/h.
- An object is considered not moving if the speed of the object is equal to or below 0.5 km/h.
- A car is considered *parking* if it does not move for two minutes.
- The alert threshold is set to 0.5. So if an object has a suspicious rate of 0.5 or more, it is considered as an alert. The Reasoning Agent continuously prints the probability so we can monitor it.
- Another thing that we can monitor is the actions that are attached to the TimeWindow by the ActionWriter. With this information we can conclude if the Reasoning Agent did a good job interpreting the incoming information.

The calibration contains the following tests:

1. Car drives around for 30 minutes with 49 km/h. Expected result: no alerts.
2. Car is driving 60 km/h in sight of the cameras. Expected result: no results up till 30 seconds after the start. After 30 seconds: at least one alert.

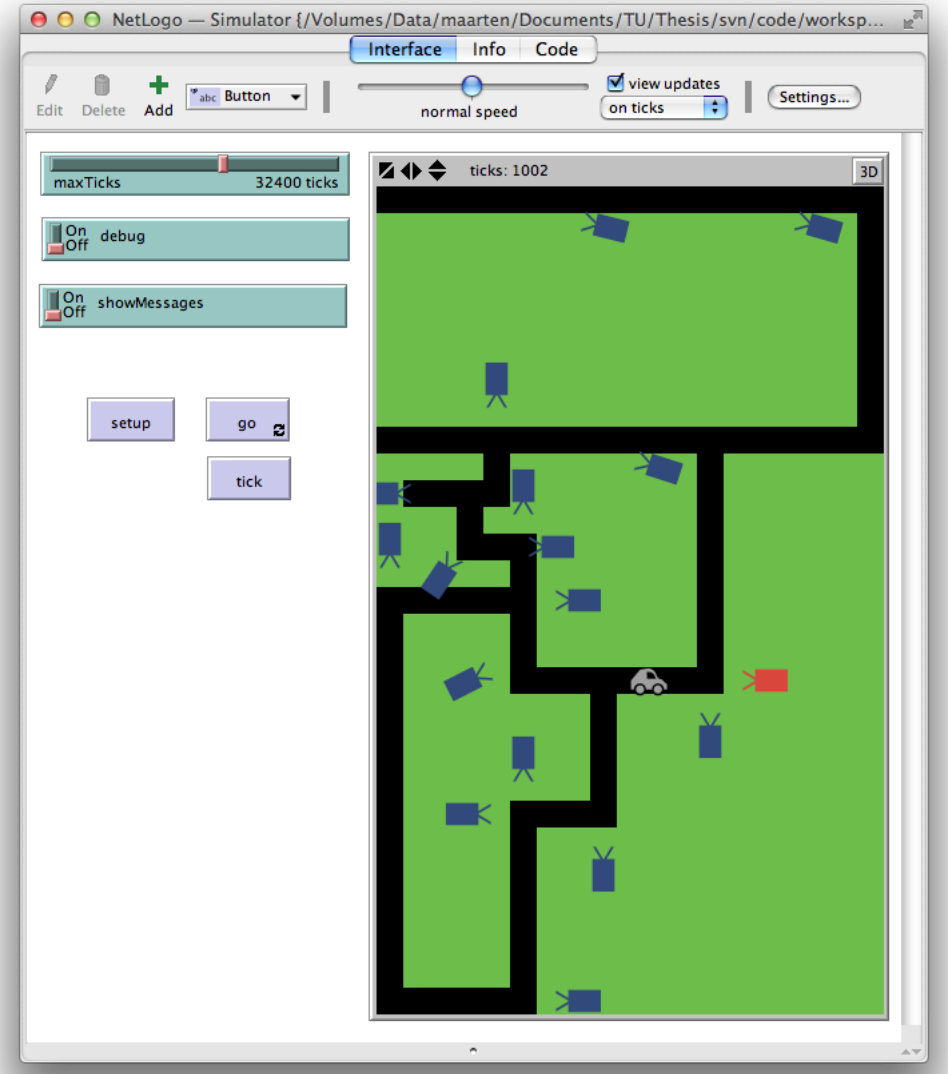

Figure 9.1: NetLogo project file which is used to test the Reasoning Agent.

3. Car enters a forbidden area for 20 seconds. Five minutes later, the object stops at a non-stopping area. Expected result: no alerts until 5:20 minutes. After that an alert should be given.

4. Car stops on non-parking area and waits for 3 minutes before it starts driving again. Expected results: after 2 minutes, the Reasoning Agent should know that the car is parking. After 2.5 minutes, alerts are given. After 3 minutes no actions are shown any more and the suspicious rate slowly drops.

Notes:

- If we expect an alert, it does not matter how many alerts are displayed exactly. The reason

for this is that the Alerting Agent will serve as a layer between the Reasoning Agent and the user to allow or deny repeated messages.

- For the Reasoning Agent, there is no difference between a speed of 51 km/h or 102 km/h, because an object can be speeding within a time window or not. However, for the simulation it matters because 'speeding for 30 seconds in sight of a camera' requires the double amount of road in sight of cameras when driving with 102 km/h compared to speeding with 51 km/h.

Results The most important result of the calibration are the DBN probabilities. The alerting threshold is fixed at 0.5 and only the probabilities are tweaked in order to acquire the expected results. The probabilities are shown in table 9.4. For an explanation of the node names see section 6.2.1.

Table 9.4: Results of experiment 1: calibration.

Node	Inhibit probability
Suspicious rate	.01
Entering forbidden area	.93
Stopping on non-stopping zone	.96
Speeding	.951
Parking on non-parking zone	.96

Since the calibration only contains experiments with cars, no probabilities for people are shown. However, the nodes *Suspicious rate*, *Enter forbidden area* and *Stopping on non-stopping zone* are also defined for a person, so the same probabilities can be used. Then there is one person-specific node left, called *Running*. This probability is set to .96 because it is similar to speeding. Note however that a security guard might disagree. A running person might be considered more suspicious than a driver that exceeds the speed limit, since running is an active act and exceeding the speed limit can be the result of a small mistake.

The purpose of test 3 was to calibrate the system such that objects do not lose the label 'suspicious' very fast. In other words, if objects keep acting suspicious, the system should take it all into account and give an alert at a certain time. After performing test 3 for the first time, the suspicious rate dropped considerately during the first ten seconds. Therefore, the suspicious rate probability was adjusted to the current value of .01. Now the car is still somewhat suspicious after five minutes. When test 3 was done, test 2 needed to be redone in order to determine the right probability for the *Speeding* and *Running* nodes.

9.2.2 Experiment 2: Validation

Setup Using the configuration from the calibration, one experiment is conducted and it is explained how the information of four observations run through the system. The following test is defined:

> Person runs for 20 seconds in a forbidden area. Expected result: at least one alert within these 20 seconds.

Results The test succeeded: an alert was given after 14 seconds. Figure 9.2 shows a small part of the process of this test. On the left side of the figure, the object and its immediate environment is shown. The upper two observations at t_{160} and t_{161} are observed by the camera north of the object, while the lower two observations at t_{162} and t_{163} are done by the camera west of the object. The vectors below the title *Observation Agent* represents the information that is acquired by the (imaginary) Observation Agents in the NetLogo environment. This information is sent through JADE to the Reasoning Agent.

In the Reasoning Agent, the ObservationWriter bundles the observations (as Observation objects) in TimeWindows. When a TimeWindow is created, it automatically determines the speed and curvature. This TimeWindow is 'written' to the Whiteboard and the ActionWriter picks it up for inspection. In this case, the person runs (since the speed is 10.0 km/h, which is greater that 9 km/h) and it moves through a forbidden area. So for each TimeWindow, two actions are attached. When the ReasoningEngine reads these TimeWindows, it determines the evidence for all actions. Since the object is not stopping at a non-stopping area, the second value of the evidence vector is false ('F'). This evidence is inserted into the Dynamic Bayesian Network (DBN) at the right time slice number. After that, the marginal probability is requested from the DBN. That probability is sent to the Alerting Agent for evaluation. Depending on the current policy of this agent, an alert is given or not.

Note that this table is simplified: the current frame rate in NetLogo is set to 30 frames per second, which means that an average TimeWindow consists of $2 * 30 = 60$ observations. However, since the object is observed by two different cameras, observations t_{160} and t_{161} must reside in another TimeWindow than t_{162} and t_{163}.

In order to illustrate the system even more detailed, figure 9.3 is created. It shows one time slice of the DBN in this experiment. De red numbers are the marginal probabilities and the red characters represent the evidence for a particular node. The blue numbers are the probabilities that the evidence is not inhibited. In other words: it is equal to $(1 -$ inhibit probability$)$.

9.2.3 Conclusion

The calibration experiment of the Reasoning Agent showed that the agent is flexible in the sense that the weights can be adjusted in an intuitive way. Furthermore, the agent acts as we expected: the suspicious rate increases if violations are detected and it decreases slowly if no violations are detected any more.

In the validation experiment we observed that the suspicious rate increases significantly faster if multiple violations are committed at the same time, which is a desired result. Furthermore, the process of the whole system is made clear.

More scenarios need to be tested in order to give firm conclusions. Nevertheless, the first experiences with the Reasoning Agent suggest that it can effectively reason about suspicious behavior.

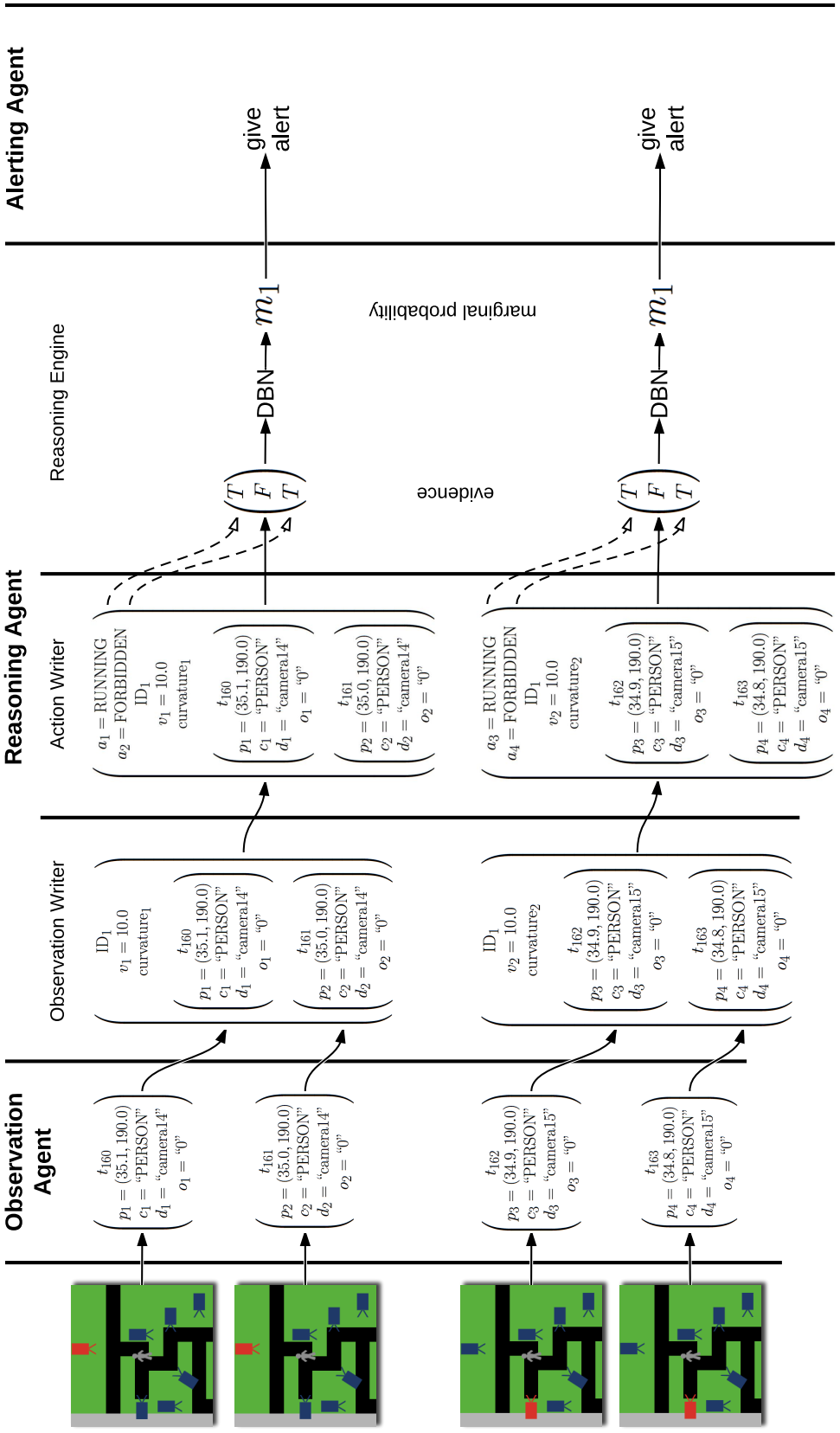

Figure 9.2: The processing of four observations from experiment 2: validation.

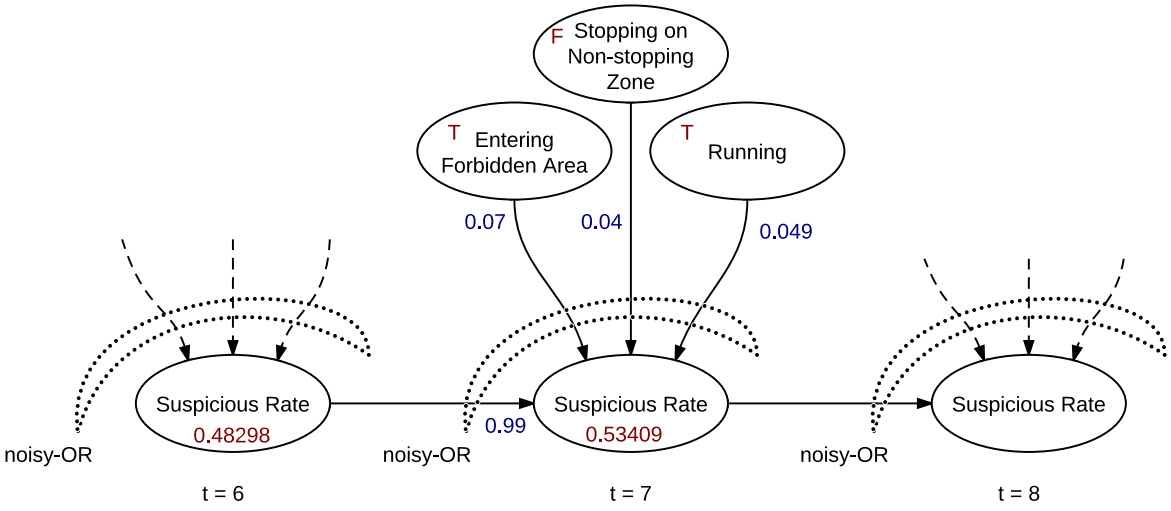

Figure 9.3: Slice 7 of the DBN from experiment 2: validation.

10 Summary and Conclusion

During this graduation project a so called proof-of-concept is created for a video surveillance system. Parts of that concept are implemented and tested. Section 10.1 describes the proof-of-concept that is designed. Section 10.2 shows what parts are implemented and its performance is summarized in section 10.3. Section 10.4 describes the project results and section 10.5 closes with recommendations for future work.

10.1 Proof-of-concept

A proof-of-concept is designed for a surveillance system that detects suspicious behavior. The system is designed as a distributed system so that it is suitable for both centralized and distributed video surveillance setups. Two alternative designs are described and arguments are given why the current design is most preferable.

The system is organized as an agent network. Each agent has its own responsibilities. Three agents are defined: the *Observation Agent*, the *Reasoning Agent* and the *Alerting Agent*. The Observation Agent is the only agent that can have multiple instances. In other words: multiple Observation Agents are present in the agent network, one Reasoning Agent and one Alerting Agent. The responsibility of the Observation Agents is to observe the world, extract information and send it to the Reasoning Agent and each other. The Reasoning Agent has an overview of the whole surveillance area, reasons about it and sends alerts to the Alerting Agent. This agent decides which alerts are passed on to the operators in the Surveillance Base Station and how many.

10.2 Implementation

Two prototype applications are implemented, namely:

- The Observation Agent which analyses a video stream by detecting, tracking and classifying objects.
- The Reasoning Agent which reasons about the data received from the Observation Agents and reports how suspicious each object is.

State-of-the-art techniques are used for realizing the prototypes. The tracker from the Observation Agent uses the *P-N Learning* technique to learn from its mistakes. *Dynamic Bayesian Networks* are created to reason about the data that the Reasoning Agent receives. Furthermore, the agent is able to communicate in the *FIPA Agent Communication Language* using *JADE* which makes it easy to communicate with other agents.

One of the advantages of the system is that it can work independently once it is calibrated and configured. It is feasible to replace tools if necessary without braking the system. Automated unit tests are constructed to help the developer keeping the software reliable.

10.3 Experiment Results

Experiments are conducted on the three different tasks of the Observation Agent mentioned above. When detecting objects, the object was on average being detected 5.25 frames too early or too late with a frame rate of 25 frames per second. The tracker performed perfectly for three out of five datasets and eventually failed for the other two datasets because it started to track the background. The average error rate of all five datasets is .1150. Finally, the classification of the objects resulted in an error rate of .6113. The results of the tracker and detector are considered good, while the classification still needs improvement.

Also the second prototype is tested: the Reasoning Agent. Experiment 1 calibrated the internal weights of the agent by performing different suspicious acts using a simulator. This calibration experiment successfully calibrated the agent to our predefined needs, but it also shows that the agent is flexible. Experiment 2 validated the Reasoning Agent by performing yet another suspicious act using the simulator, but it also illustrated the operation of the complete surveillance system.

Because of the limitation in time, the experiments are not extensive enough to make firm conclusions. However, the results suggest that this prototype potentially can be the basis of a successful automatic video surveillance system that detects suspicious behavior and notifies the operators.

10.4 Project Results

In section section 1.5 project goals were defined. Below, we explained what our opinion is about the results of those goals.

- *Perform a short literature study on automatic video surveillance to find out what the state-of-the-art is.* This short literature study gave a good overview of the field of video surveillance. During the project, lots of other papers are found to help with specific problems and the survey papers helped to get grip on the basics and possibilities of automatic video surveillance.

- *Design a model of a system that is able to automatically detect suspicious behavior.* The strength of the current model lies in the clear responsibilities for each agent. This makes designing and implementing the agent details a lot easier. The scalability of the central Reasoning Agent is yet to be tested, but the experience with the implementation suggests that it is an elegant model for this purpose.

- *Implement the video processing of a single video stream.* The results of the video processing are good, keeping in mind that a basic object detection technique is chosen, the tracker is not yet tweaked in an attempt to improve results and the object classifier is expected to improve significantly with better data or another tool.

- *Implement combining the information of multiple cameras to detect suspicious behavior.* The Reasoning Agent prototype is able to have an good overview of what the current state of the surveillance area is. Information from multiple cameras are combined such that the

Reasoning Agent can reason about it. Furthermore, the agent is designed such that tools can be easily replaced.

- *Perform experiments on both implementations and analyze the results.* The analysis shows the strengths and weaknesses of the prototypes and suggests that they have a great potential.
- *Give recommendations for future research.* When more research is conducted on this model and these prototypes, it is expected to evaluate into a very useful automatic video surveillance system. Therefore, section 10.5 gives suggestions about topics that need attention.

Based on this information about the project goals, we can proceed with answering the research questions. Section 1.5 formulated the following two research questions:

> Question 1: How can we use a video surveillance camera to replace the human eye?
>
> Question 2: If reasoning is applied to the data from multiple cameras, to what extend can we automatically detect suspicious behavior?

Part of the answer to question 1 lies in the Observation Agent. This agent is able to detect an object and observe its trajectory. The experiments show that the agent is not able to classify the object reliably yet. It does a good job at mimicking what a human being would do, but the agent shows situations where a person would have no problem in outperforming the agent. However, it is assumed that lots of improvements are possible. Apart from detection, tracking and recognition, humans also incorporates context and knowledge. This is captured by the Reasoning Agent. This smart agent can successfully incorporate context and knowledge in the judgment of how suspicious an object is. To conclude, we are able to emulate operators with a system which can potentially outperform a human being.

Question 2 targets the performance of the detection when data from different cameras is combined. This combining process is done in the Reasoning Agent. It has an overview of the situation and is able to reason about how suspicious an object is. One limitation is that the suspicious behavior must be predefined. If the agent does not know about a certain suspicious or dangerous act, it cannot detect it. But if the Reasoning Agent has been told what is suspicious and how it can be measured, it is able to detect it although it is observed by multiple cameras. To conclude, the answer to question 2 is: if we define what is considered suspicious behavior, we can automatically detect it.

The prototype applications have shown that an automatic video surveillance system can be built that detects suspicious behavior. The distributed nature of the system makes it easy to develop and maintain. The combination of rules based reasoning and probabilistic reasoning has proven a very viable and flexible solution.

10.5 Future Work

Due to the limitation in time and resources, this research does not provide a read-to-use system. Future research can focus on developing the system and proving its performance. The following items are recommended:

- Multi-object support: since the system currently can track only one object, it would be a great addition to allow multi-object tracking. This primarily affects the Object Detector

module, the Object Tracker module and the IdentificationWriter class. The ReasoningAgent in general is prepared for handling multiple objects.

- Expert knowledge: experts are needed to give their opinion on what is suspicious and what is not.

- Solving single point of failure: currently, the system has two agents that are critical for the system to work: the Reasoning Agent and the Alerting Agent. It is recommended to consider system designs that do not have a single point of failure. However, this most probably will increase the complexity of the system and the Observation Agent specifically.

- Communication with the Observation Agent: currently, the simulator is able to communicate with the Reasoning Agent via JADE. However, the Observation Agent itself is not able to communicate yet.

- Pan-Tilt-Zoom camera support: to expand the area that is under video surveillance without installing significantly more cameras, Pan-Tilt-Zoom cameras can be a great solution. However, the background subtraction algorithm as well as the perspective transformation to determine the object's position assume a fixed camera. However, if these two Observation Agent issues are solved, the system is able to handle Pan-Title-Zoom cameras.

- Variable frame rate support: section 3.1 stated that we require the system to have a frame rate of at least one frame per second. However, when something suspicious happens it might be interesting to temporarily increase the frame rate in order to supply the Reasoning Agent with more detailed information.

- Smarter Reasoning: the Reasoning Agent has a basic reasoning network, but it can easily be extended to do more complex reasoning. For example: continuous nodes can be introduced and more actions can be invented. See section 3.3 for inspiration.

- An Alerting Agent needs to be implemented to present the alerts to the operator.

- Extensive tests and calibration is needed to make the system work reliable enough. Especially the Object Classification needs attention. Furthermore, the scalability should be tested: how many data can the Reasoning Agent handle?

- Object matching in the Reasoning Agent: currently the Reasoning Agent only can identify objects based on the naive assumptions that every observed object is the same. It would be very useful if it could receive the object information from the Object Tracker and still images of the object. In this way, the agent can also distinguish objects based on their visual properties. Furthermore, identifying non-physical features would improve the system even more. To give an example: imagine two marines walking down the street. They look very similar due to their uniforms. However, the style of walking might be a feature that distinguishes the two. One significant advantage is that this feature is not camera-specific. Finally, reasoning about place and time can improve this process. For example, a red car is seen at location A. Twenty seconds later, a red car is observed at location B which is 2.4 km away from location A. Whatever matching function matches these red cars, we can reject this matching based on the time and place of these (distinct) objects.

We showed the proof-of-concept of a system which is able to emulate operators and can potentially outperform a human being. Once the system knows what is considered suspicious behavior it can be automatically detected. This brings us one step closer to an automatic video surveillance system that can be used to prevent incidents and keeping us safe.

Bibliography

[1] S. Bandini and F. Sartori. Improving the Effectiveness of Monitoring and Control Systems Exploiting Knowledge-Based Approaches. *Personal and Ubiquitous Computing*, 9(5):301–311, June 2005. 23

[2] M. Beelen. Personal Intelligent Travel Assistant. Master's thesis, Delft University of Technology, 2004. 52

[3] G. Bradski. OpenCV Home Page. `http://opencv.willowgarage.com/`. Date accessed: March 11, 2011. 46

[4] C. Chen and G. Fan. What Can We Learn from Biological Vision Studies for Human Motion Segmentation? *Proc. International Symposium on Visual Computing*, 2006. 12

[5] F. G. Cozman. Axiomatizing Noisy-OR. pages 1–13, 2004. 51

[6] G. de Haan, J. Scheuer, R. de Vries, and F. H. Post. Egocentric navigation for video surveillance in 3D Virtual Environments. *2009 IEEE Symposium on 3D User Interfaces*, pages 103–110, 2009. 12

[7] C. Djeraba. MIAUCE Home Page. `http://www.miauce.org/`. Date accessed: April 14, 2011. 25

[8] M. W. Green. The appropriate and effective use of security technologies in U.S. schools: a guide for schools and law enforcement agencies. *National Institute of Justice*, 1999. 12, 13

[9] S. Hampapur, A.; Brown, L.; Connell, J.; Ekin, A.; Haas, N.; Lu, M.; Merkl, H.; Pankanti. Smart Video Surveillance: Exploring the Concept of Multiscale Spatiotemporal Tracking. *IEEE Signal Processing Magazine*, 22(2):38–51, 2005. 15, 23

[10] W. Hu, T. Tan, L. Wang, and S. Maybank. A Survey on Visual Surveillance of Object Motion and Behaviors. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 34(3):334–352, Aug. 2004. 18

[11] N. Ihaddadene and C. Djeraba. Real-time crowd motion analysis. *2008 19th International Conference on Pattern Recognition*, pages 1–4, Dec. 2008. 25

[12] I. N. Junejo, X. Cao, and H. Foroosh. Autoconfiguration of a Dynamic Nonoverlapping Camera Network. *IEEE transactions on systems, man, and cybernetics - Part B: Cybernetics: a publication of the IEEE Systems, Man, and Cybernetics Society*, 37(4):803–816, Aug. 2007. 21

[13] Z. Kalal. TLD Zdenek Kalal. `http://info.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html`. Date accessed: April 26, 2011. 45

[14] Z. Kalal, J. Matas, and K. Mikolajczyk. P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 49–56, June 2010. 45, 46, 86

[15] I. Lefter, L. Rothkrantz, P. Bouchner, G. Burghouts, and P. Wiggers. A Multimodal Car Driver Surveillance System in a Military Area. 2010. 16, 32

[16] T. Lethbridge and R. Laganiere. *Object-Oriented Software Engineering: Practical Software Development using UML and Java*. McGraw-Hill, 2002. 71

[17] Y. Li, C. Huang, and R. Nevatia. Learning to Associate: HybridBoosted Multi-Target Tracker for Crowded Scene. *IEEE Conference on Computer Vision and Pattern Recognition*, 0:2953–2960, 2009. 22

[18] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, 2002. 51

[19] P. Over, G. Awad, and J. Fiscus. TRECVID 2009 – Goals , Tasks , Data , Evaluation Mechanisms and Metrics. In *Proceedings of TRECVID 2009*. NIST, USA, 2010. 16

[20] H. Pasula, S. Russell, M. Ostland, and Y. Ritov. Tracking many objects with many sensors. *Proceedings of the International Joint Conferences on Artificial Intelligence 1999 (IJCAI-99)*, (1), 1999. 26

[21] J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers (San Mateo, Calif.), 1988. 51

[22] T. D. Räty. Survey on Contemporary Remote Surveillance Systems for Public Safety. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 40(5):493–515, Sept. 2010. 20, 26

[23] S. J. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010. 50, 52

[24] W. M. Thames. From Eye to Electron - Management Problems of the Combat Surveillance Research and Development Field. *IRE Transactions on Military Electronics*, MIL-4(4):548–551, Oct. 1960. 13

[25] M. Valera and S. A. Velastin. Intelligent distributed surveillance systems: a review. *Image (Rochester, N.Y.)*, (20041147), 2005. 13

[26] P. Wiggers. *Modelling Context in Automatic Speech Recognition*. PhD thesis, Delft University of Technology, 2008. 50, 51

[27] P. Wiggers, B. Mertens, and L. Rothkrantz. Dynamic Bayesian Networks for Situational Awareness in the Presence of Noisy Data. *International Conference on Computer Systems and Technologies - CompSysTech*, pages 411–416, 2011. 51

[28] U. Wilensky. NetLogo Home Page. `http://ccl.northwestern.edu/netlogo/`. Date accessed: January 5, 2012. 55

[29] U. Wilensky. NetLogo, 1999. 55

[30] X. Yang, Y. Xu, R. Zhang, E. Chen, Q. Yan, B. Xiao, Z. Yu, N. Li, Z. Huang, C. Zhang, X. Chen, A. Liu, Z. Chu, K. Guo, and J. Huang. Shanghai Jiao Tong University Participation in High-Level Feature Extraction and Surveillance Event Detection at TRECVID 2009, 2009. 17

[31] K. Yokoi, T. Watanabe, and S. Ito. Toshiba at TRECVID 2009 : Surveillance Event Detection Task, 2009. 17