NTNU
Innovation and Creativity

Cognitive video surveillance: an ANN/ CBR hybrid approach

Anders Oredsson

Master of Science in Informatics
Submission date: June 2007
Supervisor: Agnar Aamodt, IDI
Co-supervisor: Sule Yildirim, Høgskolen i Hedmark

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

Abstract

Cognitive vision is an interesting field of research that tries to create vision systems with cognitive abilities. Automated video surveillance is increasingly needed to watch our public transport systems, either as a totally automated system or as an operator assistant. In this thesis a design for a cognitive automated video surveillance system is proposed. A hybrid ANN/CBR behavior analysis system is proposed as a cognitive extension of existing video tracking system, and a prototype is developed. By investigating the current state of cognitive vision and automated video surveillance and by looking at existing hybrid approaches to combine them, it is argued that this approach is an area of research where few designs are implemented and tested. Several frameworks for both ANN/CBR hybrids are proposed in literature, and so called emergent cognitive system frameworks are also presented, but few implementations have been done. As far as our literature review has spanned, no automated video surveillance system is attempted with the use of an ANN/CBR hybrid.

Acknowledgements

I would like to thank both my supervisors *Dr. Sule Yildirim* and *Dr. Agnar Aamodt* for assistance through the process of writing this thesis. Special thanks to *Dr. Yildirim* for introducing me to the project, and helping me with the development of this Master's thesis proposed approach.

I would also like to thank *Dr. G. A. Jones*, at the *University of Kingston (UK)*, for letting me visit *Kingston University* and for introducing me to their video analysis research. Also thanks to all the people at Kingston University who helped me by showing and explaining their research.

Finally I would like to thank *K. Berg-Eriksen* for general support.

Table of Contents

Chapter 1

Introduction

1.1 Master Thesis

This Master's Thesis was written at the department of computer and information science (IDI) at the Norwegian University of Science and Technology (NTNU). The thesis was started january 2006, and finished june 2007.

1.2 Motivation

ACE (Autonomous Artificial Cognition in Complex Environments) is an EU project proposal created by a network of universities and stakeholders from all of Europe (Trollhetta AK, Norway. Norwegian University of Science and Technology, Norway. Kingston University, UK. University of Plymouth, UK. University of Skvde, Sweden. Institute of Computer Vision and Applied Computer Sciences, IBAI, Germany. Metropolitano Di Roma SpA.). This Master's thesis is an attempt to explore the ideas of this project proposal. The Master's thesis is independent of the EU project proposal, but is at the same time heavily inspired by it. Both my supervisors Dr. Sule Yildirim and Dr. Agnar Aamodt participated in creating this project proposal, and the master's assignment was proposed by Dr. Yildirim as a direct result of her involvement in the project proposal.

Crime, anti-social behaviour and terrorism represent an increasing concern for most European countries and their citizens. The primary goal of the project proposal as an EU proposal for a cognitive systems call is to significantly advance the current state-of-the-art in automating the surveillance of scenes by building an autonomous cognitive system with human-like learning, reasoning and ”active perception” capabilities. On the other hand, the project proposal has two specific objectives as an EU proposal:

- To get an understanding of how human cognitive capabilities can improve the way we build situated systems that continuously interpret their environments through visual and auditory input channels.
- To develop a technology for autonomous surveillance and threat detection systems, by improving their situation assessment ability through a novel approach that combines lower-level perceptive with higher-level cognitive capabilities.

In order to build an autonomous cognitive system with human-like learning, reasoning and ”active perception” capabilities, we employ an approach which takes advantage of the strengths of symbol-processing AI methods, in particular case-based reasoning, having good reasoning and planning capabilities, and combine them with subsymbolic methods, in particular artificial neural networks, which has good coupling with the physical environment. To this we add the sensing, tracking, and preprocessing capabilities from camera systems. Our approach will allow a system, through its integration of these method types, to continually adapt to a changing environment by updating its explicit network of cases and other domain knowledge as well as its neural model connections. A particular focus will be on integrating categories emerging from the neural layer with the symbolic reasoning based on an explicit knowledge model.

The master thesis stays within the same larger and specific goals but has an ambition level that supports a focus on designing the framework that combines a symbol-processing AI method (CBR) with a sub-symbolic AI method (ANN). This way it forms the basis for an autonomous, human-like learning and reasoning cognitive video surveillance system. The framework couples the physical environment with neural networks, emerges categories from this coupling and forms the cases based on these categories. The framework is then implemented in the form of a prototype to validate the usefulness of the design. The physical environment is captured by a camera in the form of video frames. These captured video frames are used as data for training and testing the prototype. A survey on the literature about the state of the art of on cognitive surveillance systems is also given in the thesis.

1.3 Readers guide

This thesis is divided into the following parts.

Introduction

Background and introduction for the thesis.

Problem description

Description of problem, short introduction to automated video surveillance, the connection with cognitive vision, and an overview of the task.

Literature review

Literature review of Video Surveillance, Cognitive Vision, Artificial Neural Networks, Case Based Reasoning, and ANN/CBR hybrids. We will also look at some existing automated surveillance systems.

Design

Design, concepts and description of solutions to our thesis problem.

Process and Prototype

The master thesis process covering design development, how data was collected, prototype design and implementation, process of running and analyzing experiments.

Results

The experiments and results of our thesis.

Conclusion

The conclusion of the thesis, a discussion regarding results, and a summary of scientific findings.

Reference

References.

1.4 The fight example

Through this master thesis we will use one example through all chapters. This example is called *The fight example*, and concerns how our system will try to detect if somebody is fighting in any of the videos analyzed by our system. This example can be thought of as the ”red tread” the reader can follow from start to end. By using the same example through the whole text, we hope to give one detailed case which will take the reader all the way from the video containing the fight, to the final detection of the fight event.

Chapter 2

Problem description

2.1 Automated video surveillance

The project description that was originally agreed upon between me and my supervisor was the following:

Both symbolic approaches of AI and neural networks have a number of desirable properties which when combined would help building truly intelligent agents. The advantages of the symbol systems are that humans also use symbols to communicate. In critical situations, knowledge of rules can help save lives as well as knowledge communicated by rules can speed up learning. On the other hand, neural networks are useful in situations where knowledge cannot be easily prescribed by rules. They also exhibit graded response and can perform in the situation of partially missing knowledge. In this project, a hybrid system is proposed which uses neural networks for converting sensory information from the environment into categories (i.e. objects, actions, and actors, and other spatial and temporal structure of the environment), and which enables CBR for reasoning with these categories. The categories are effectively the scene descriptors embedded in the case structure.

The project composes of the following tasks:

- *Design of the case stucture and deciding on scene descriptor types for example scenarios.*
- *Review of research relevant to categorical formations*
- *Design of a neural network that maps environmental information onto categories.*
- *Application of the neural network for several scenarios.*

The mapping of environmental information onto categories can be in two ways. The first one is to feed environmental information into the network directly. The second one is to feed some features of the environment into the neural network. A collaboration with Kington University, UK is foreseen to decide on which way to proceed the work.

After exploring the problem, our focus shifted to cognition and cognitive vision. Instead of simply measuring the performance of the system in success-rate, we decided to create a measurement-matrix regarding the cognitive levels of an automated video surveillance system. This decision was based on the fact that an automated video surveillance system does more then detecting events. For example, learning events, re-learning events, explaining detected events, running in real-time and running in a real-world environment are all important features of our goal design. The chapter *Cognitive Video Surveillance* introduces cognition to this thesis, our measurement matrix for measuring the level of cognition, and looks at existing comparable systems from this perspective.

2.2 Problem description

In this thesis we will work on adding cognition to an automated video surveillance system. This will be done with a symbolic / sub-symbolic hybrid system that aims at benefiting from both the symbolic and sub-symbolic methods. A measurement of the level of cognition in an automated video surveillance system will be proposed, and we will use this measurement both on existing systems and the hybrid prototype, to propose that this hybrid system increases levels of cognition.

2.3 Project goals

We present the following goals as the goals we want to achieve through this project:

- Specify a set of requirements for an automated video surveillance system to be cognitive
- Develop a design of a cognitive automated video surveillance system based on an ANN/CBR hybrid
- Implement a prototype of the proposed design for experimentation
- Measure the prototype's ability to perform the task of automated video surveillance, based on the set of requirements concerning how a cognitive automated video surveillance system should perform

By fulfilling these goals, we hope to answer the following set of questions:

How can we define cognition in the task of automated video surveillance?

What is a cognitive automated video surveillance system? How does it relate to the fields of cognitive vision and automated video surveillance? What requirements must an automated video surveillance system fulfill to classify as a cognitive automated video surveillance system?

Can cognition be achieved through an ANN/CBR hybrid in the task of automated video surveillance?

Will we be able to fulfill the requirements for a cognitive automated video surveillance system by creating an ANN/CBR hybrid system? What other approaches are investigated in literature, and are they cognitive?

How does an ANN/CBR hybrid system's performance compare to the performance of a pure ANN system or a pure CBR system in an automated video surveillance setting?

What advantages can we achieve by combining ANN and CBR into a hybrid?

Is an ANN/CBR hybrid useful for behavior analysis when placed on top of a video tracking perception system?

In literature, video tracking has been used for video surveillance, but is it a useful perception system to build a cognitive automated video surveillance system upon? Is it useful as the underlying perception system of an ANN/CBR hybrid behavior analysis system as part of a cognitive automated video surveillance system?

How do we represent the real-time real-world scene under surveillance internally in an ANN/CBR hybrid connected to an underlying video tracking perception system?

What information should be processed, maintained and stored? What part of the real-world does this information represent?

How can we handle temporal data in an ANN/CBR hybrid system?

If the system is analyzing what is happening now, how can it represent and reason upon what happened in the past second or the past hour?

How will an ANN/CBR hybrid placed on top of a video tracking perception system handle noise and false detections from a real-world input?

Is our approach a robust approach? Will it be able to handle the noise of the real-world? What happens if noise increases? Will the system's performance collapse?

2.4 Research methodology

To fulfill the goals of this project, the following research methodology will be used:

Literature review

We will review the field of automated video surveillance as well as the field of cognitive vision.

Specification of requirements for a cognitive automated video surveillance system

We will define what a cognitive automated video surveillance system is, and what requirements an automated video surveillance system must fulfill to be cognitive.

Design of a system that fulfills the specified cognitive requirements

After we have developed a set of requirements for a cognitive automated video surveillance system, we will develop a design that fulfill these requirements.

Implementation of an experimental prototype of the proposed design

The proposed design will be implemented as a prototype for measurements through experimentation.

Measurements of the prototypes performance regarding the specified cognitive requirements

By finally measuring our prototype through experiments, we will test our approach and see if the predicted behavior matches the observed behavior, and if this observed behavior fulfill our requirements for an automated video surveillance system to be cognitive.

2.5 The gap between cognitive vision and automated video surveillance

There are currently two fields of research that is related to our thesis: Cognitive vision and automated video surveillance. Cognitive vision tries to understand and model human vision and is therefore a field of research that tries to understand how humans solve these tasks. Solving the task itself is a subproblem that will be easily solved once the modeling of the human solver is complete. Automated video surveillance focuses more on the task itself. When you enter London in a car, an automated video surveillance system ”sees” your cars registration number and puts your car in a register. The system can’t do anything else and optimized algorithms designed to only find the registration number are used. In the chapter *Cognitive Video Surveillance*, we try to look at these two fields and find the gap between them. By using cognitive vision in a realized automated video surveillance task, we hope to show that the two could and should be combined.

2.6 Task issues

During our design we faced multiple issues that we had to solve in order to make the prototype recognize events. Instead of limiting our scope, we made these issues our main problems to solve. This means that our design was created with no compromises on the input video data. Such compromises could for example be a scene with a white floor and only dark clothed people, a room lightened in a way that minimized shadows casted by the people moving around, fights that always ended with one actor lying down on the floor, only pink bags etc.

Instead we have targeted these as our main issues, and tried to make these issues the premisses for our experiments.

2.6.1 Real world data

Our system is supposed to use real world video-feeds as input. In the real world, no simplifications are made. As we recall the box world often used in AI experiments, the box world is a simplification of the real world and will only tell us what is possible in a symbolic world and not what is possible in the real world. The box world contains no noise, and the worlds elements are already symbolized since we already know what they are, where they are, and their properties. This box world can be modeled directly in a symbolic way with predicates, and then reasoned upon using first order logic. Our domain is automated video surveillance and a box world would only tell us how we could reason upon the collected data, and not how to collect the data. By running our system on real world data, we show that the features we collect from our video data-set are actually detectable. Instead of 'cheating' by labeling the entities in our video as "a person" or "a box", and then state that the perception would be future work, we hope to show that this actually can be done, using the main ideas of our design. It has been suggested that the non-perfection of real world data is the biggest problem for symbolic systems to work in the real world (Brooks, 1991). The final "if fighting then sound alarm" reasoning should therefore be looked at as the small problem while detecting "if fighting" should be considered the big one.

2.6.2 Temporal data

A problem we faced during the design of our system was temporal data. If we look at video, it has a dimension of time. An event our system should detect could consist of 3 or 4 different parts stretched over time. If a person leaves his or her bag, it could be just to rest a tired arm, but if the person later leaves the scene without taking the bag with him/her, it should be considered as a left-luggage event. Because events can be stretched over time, "now" isn't enough to detect an event, we also need to remember what happened previously. The parts of the events could also be stretched over time in different ways, so, simple time-limiting isn't good enough (keep all features for 2 seconds). Our video gives us a continuos flow of features, and we need a system that can learn events stretched over time, without the need of ad-hock algorithms that decides on what data to store.

2.6.3 Noise and missing data

Because our real world data already contains a lot of noise, and since our feature-detection algorithms isn't perfect, we expect a lot of noise and missing data. This is a parameter that could be reduced by better cameras or better algorithms, but our approach is to create a system that is designed with these problems in mind. To investigate how our system handles these data quality issues, we will try to add noise and missing data to our system to see how it handles under even worse circumstances. Better cameras and better algorithms should not be required for our automated video surveillance system to work, but should instead improve our system's performance. If our basic design can detect desired events without being specialized for a specific set of events, and can operate using available video surveillance equipment, the design's robustness is shown, and future work could focus on improving the design or implementation of the design.

2.6.4 Representation

Our system's use of both symbolic and sub-symbolic methods required representation of data both at a symbolic and sub-symbolic level. We have to find a way to represent our data that can be handled in both paradigms. This means that our representation can't be all symbolic, but symbolic structure is needed to organize and reason on our data, so the hybrid approach required a hybrid representation. This is an important part of our design and solution, and a compromise between the two paradigms was created; not all sub-symbolic and not all symbolic.

2.6.5 Unified architecture

By combining symbolic and sub-symbolic methods we hope to explore the possibilities of using a hybrid approach in the video surveillance domain. We try to use the well-researched behavior from both symbolic and sub-symbolic methods, and try to measure the difference between a hybrid and a non-hybrid approach. With our representation in place, how the methods would interact was our next problem.

Chapter 3

Literature review

3.1 Introduction

This chapter covers a literature review concerning the initial motives for this project, the research it builds on, and other related research. We will look at video surveillance assistance, cognitive vision, artificial neural networks, case based reasoning, and we will briefly discuss three existing automated video surveillance systems.

3.2 Video surveillance assistance

The level of assistance computer systems offer in video surveillance systems is currently primitive. The activity monitored through video is typically classified as typical or atypical by the assistant system. The classification is based on pre-built signatures or learned activity models (PRISMATICA; ADVISOR). The operator is alerted when an atypical activity is detected, and then has to interpret the situation manually. Alerting a security guard is also done by the operator (Troscianko, 2005). The problem with this approach is that the computer assistance detects atypicality but not crime or emergencies. This mapping between atypical and dangerous is not correct and results in a large number of false alarms. As with any ”wolf wolf” situation, every false alarm undermines the system's credibility and the operators confidence in the system could decrease to a level where the system becomes a distraction instead of an assistant. By attacking the problem at a higher level through the use of cognitive methods like reasoning and learning, the gain in using computer assistance in video surveillance could be drastically increased.

3.3 Cognitive vision

By adding artificial intelligence and cognitive abilities to computer vision a paradigm emerges that is referred to as cognitive vision (Melzer et al, 2003). This term generally concerns autonomous systems that learn to recognize and interpret their environment. This paradigm evolves from the previous paradigm of purposive vision (Aloimonos, 1994) which tries to extract task-orientated information. The field is covered by ECVision's roadmap (P Auer et al, 2005) which divides the field into three *dimensions: The scientific foundations of cognitive vision*, *the functional capabilities of cognitive vision* and *the realized task-specific competence of cognitive vision*. These three dimensions spans from pure research theory to practical application, and the idea is that progress in theoretical research will propagate down to practical applications. By documenting and developing an understanding of what practical application maps to what functional capabilities, and what functional capabilities map to what parts of the scientific foundations, a future application could pick and combine well documented ”methodical parts” from the research field, and more rapid and standardized application of cognitive vision systems could emerge.

The scientific foundations of cognitive vision covers:

- *Visual Sensing* (The vision system's input).
- *Architecture* (The environment in which the vision system needs to work, including training and setup).
- *Representation* (How the system represents its visual input and prior knowledge).

- *Memory* (How the prior knowledge is stored, organized and retrieved).
- *Learning* (How the system learns knowledge).
- *Recognition* (The system's way of detecting specific events from visual input).
- *Deliberation & Reasoning* (How the system reasons).
- *Planning* (How the system plans its actions).
- *Communication* (How the system communicates with other systems or agents).
- *Action* (What an action is, and how it is performed by the system).

The functional capabilities of cognitive vision covers:

- *Detection and Localization*
- *Tracking*
- *Classification and categorization*
- *Prediction*
- *Concept formation and visualization*
- *Inter-agent communication and expression*
- *Visuo-motor coordination*
- *Embodied exploration*

The realized task-specific competence of cognitive vision is the final application of these functional capabilities. Two main issues are addressed: the development of a cognitive vision system, and the requirements for a task-specific cognitive vision system. Concerning the development of a cognitive vision system, the roadmap predicts that a fully capable surveillance system will emerge in stages, and an example of these stages is given. Concerning the task-specific cognitive vision system examples, the requirements that these systems should fulfill are described. What is important is that the system should use functional capabilities from cognitive vision research, but the task dictates what capabilities are required. A cognitive home assistant has other requirements then an advanced driver assistance system. Cognitive science is traditionally dominated by symbolic representations and reasoning, named the *information processing* approach (Kelso, 1995), but the field is slowly shifting away from this (Clark, 2001). An alternative model is called Emergent Systems (P Auer et al, 2005). Emergent Systems cover connectionist systems, dynamical systems and enactive systems. Researchers have also explored the possibilities of hybrid systems, combining aspects of symbolic systems and emergent systems (Granlund, 1999). These systems still use symbolic representation, but these representations should be created by the system itself, not be pre-programmed. The symbols should be learned from the real world. Today's most advanced cognitive vision systems are symbolic systems, but they have achieved little concerning creating general robust systems. Enactive and dynamical systems are argued to be more general and robust, but little progress has been done concerning their cognitive capabilities. Hybrid systems' ability to use the best from both worlds is a young but evolving part of cognitive science, and opinion is divided with both arguments for (Granlund, 2005) and against (Christiansen & Hooker, 2000) hybrid systems. As summarized in (P Auer et al, 2005), the main questions that cognitive vision research has to answer are the following:

- What should the balance between Hard-Wired Functionality vs. Learned Capabilities be?
- Do cognitive systems need to be embodied?

A list of priority challenges that need to be solved is also proposed:

- Methods for continuous learning.
- Minimal architecture (Environment for the system to work within).
- Goal identification and achievement.

- Generalization, creating general solutions.
- Utilization and advancement of system engineering methodologies.
- Development of complete systems with well-defined competence.
- Research tools.
- Time horizons and resource implications.

3.4 Artificial neural networks (ANN)

ANNs is an ideal candidate for our task classification and has been applied successfully in other task classification systems. Architectures for integrating categorical and symbolic representations have been studied, as in (Cangelosi et al. 2000; Cangelosi and Parisi 2001) where it was demonstrated that internal categorical representations are enhanced when the networks are asked to categorize and label the input stimuli. This model also permitted the construction of hierarchies of categorical representations applicable to our top level CBR reasoning system. ANNs have shown success in solving problems such as incomplete/partial data, generalization to unseen situations, predictions and continually changing environments (Rumelhart et al., 1986). These are all problems we expect to encounter when connecting our top level symbolic reasoning system to real world data.

3.5 Case based reasoning (CBR)

CBR joins the set of symbolic methods used in high level AI. These systems have shown cognitive capabilities concerning complex reasoning, interpretations, elaboration of data, and decision making (Aamodt and Plaza, 1994). The CBR reasoning compares current situation with previous situation-specific experiences in form of cases stored in the CBR-system's case-base/memory. Learning/Adaption consists of maintaining the case-base's completeness and accuracy. New cases are remembered while false cases are removed. A large set of tactics and strategies concerning maintaining and updating CBR case-bases have been studied, and CBR systems have shown success in domains involving classification, but also in more general problem solving domains (Watson, 1997). Even if our goal as AI researchers is to imitate and understand cognitive behavior, and not to copy nature directly, a hypothesis in neuroscience says that humans collect context and all parts of a situation and stores them as a single event (Rolls and Treves, 1998).

3.6 ANN and CBR hybrids

Neither the lair of symbolic nor sub-symbolic AI research has been fully successful in developing truly intelligent systems. Hybrid approaches combining the powers of both sides are proposed and studied (Lees and Corchado, 1999). ANNs' machine learning capabilities have been used to enhance the reuse of past experience in the CBR reasoning cycle (Lees and Corchado, 1999). Different approaches for combining ANNs and symbolic knowledge processing (Expert systems) have been studied and realized (Ultsch, 1995), and different levels of integration, from connecting modules, to a fine-grained integration of the two sides have been investigated (McGarry, Wermter and MacIntyre, 1999). These studies show that a CBR-ANN hybrid system has a lower error in weather forecasting experiments compared to solely ANN, statistical and CBR methods. ANNs have shown to be possible to integrate with symbolic systems in a wide range of problems, and different types of integrations are identified and related to different types of problems.

3.7 Existing video surveillance systems

Grasping the whole field of existing video surveillance systems is a task too large for this Master's thesis. Instead, we will look at three chosen systems.

3.7.1 AVITRACK

The EU project AVITRACK aims to automate the supervision of commercial aircraft servicing operations on the ground at airports (M Borg et al, 2005). The system consists of a scene tracking module and a scene understanding module. The system is based on video from multiple cameras, and a unified representation of the scene is created and reasoned upon. Although if the task of supervising aircraft servicing operations on the ground at airports is different from video surveillance in subway stations, both systems are supposed to detect events of moving objects. What is interesting is that the system uses predefined symbolic video events. A pre-defined event consists of a name, a set of physical objects, a set of composite sub-events and a set of constrains. The constrains can for example limit the event to certain types of physical objects, and can also decide on the order which the composite sub-events have to unfold. One important reason why this system can be modeled with pre-defined events is that the model is known. All operations follow a set of procedures and these procedures are known in advance, so that the procedures can be precisely modeled, and if the procedure isn't followed, an alarm can be fired. In the subway station, this is totally opposite, and no formal descriptions of how people should behave exist. The difficulty of modeling operations using expert knowledge is also discussed, and the recognition of more complex operations is described as future work.

3.7.2 ADVISOR

The ADVISOR (Annotated Digital Video For Intelligent Surveillance And Optimized Retrieval) project is the first to demonstrate the integration of all the capabilities of *Image Capture, Behavior Recognition, Motion Detection, Crowd Monitoring, People Tracking, Archiving, Search and Retrieval, Human Computer Interface* and *Communications over an IP infrastructure*. The project's goal is to improve the management of public transport networks through better exploitation of data from CCTV cameras. The project aimed to use the latest developments in computer vision technologies. For behavior detection, ADVISOR uses Recurrent Bayesian Networks (Bremond, Moenne-Loccoz and Thonnat, 2003). The Bayesian Networks allow the system to model a causality between A and B. The conditional probabilities are learned from a set of examples of the domain. Violent behavior is recognized as a high level of global agitation. The global agitations probability is based on features from current and previous frames (A frame is all the aggregated features of a specific time-interval in the video feed).

3.7.3 Traffic Surveillance

Nagel (Nagel, 2004) discusses a traffic surveillance system which generates a natural language description of the scene it is surveilling. A powerful underlying analysis system that can *track road vehicles under realistic boundary conditions* processes the raw video feed. The next processing step is to recognize movement primitives from trajectories. A Situation Graph Tree is developed from the underlying movement primitives. The movement primitives are compared with a set of pre-programmed ”higher level” events. These events are combined to test for even higher level events, and on the top level of this event hierarchy, events like *agent A is crossing B* are found. The system’s internal model of the world, its events and the relations between different events are pre-programmed. No learning is needed since the model is already known and understood.

Chapter 4

The Design

4.1 Introduction

In this chapter we will go in detail on how we designed our solution. We will look at the overall design, the concepts of the design, the design's connection with cognitive vision, video analysis, behavior recognition and finally our design's use of ANNs and CBR.

4.2 Cognitive video surveillance

Introducing cognition to video surveillance requires several steps. By defining our use of cognition, and an analysis of what cognition means for automated video surveillance, we should show the reader in what way the cognition is used and understood. We also need a way to measure the cognition of our system.

4.2.1 Defining cognition

There are several definitions on what cognition is. Some are field dependent, and they span from psychology to artificial intelligence. Since our assignment concerns video surveillance and not general human cognitive behavior, we are going to limit our view on cognition to the video surveillance context, but before we connect cognition to context, we must first state what cognition is.

> *Cognition is: the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses. (Oxford American Dictionaries,2007)*

Our system's input would be the senses, where experiences are made. The system's output would be its actions or its process of understanding through thought. By also letting the system explain this understanding to some degree, the results and the explanations become the output. The level of cognition required to fulfill a task is defined by the task and the task's context. It's not required that the context contains a concrete task, but some abstract task like ”being an ant” is always present, and is the system's goal in that context. For example ”being a mouse in a basement” could be our cognitive context. Since we really don't know what ”being a mouse” requires cognitively, we could refine our task and context to ”behave like a mouse in a basement”. Since few practical tasks consist of just copying a natural being's behavior (creating an artificial mouse, just for the sake of reproducing a mouse's behavior), we should refine the task one more time to make the task more concrete and measurable. Our final task could be stated as ”finding food, don't get eaten, raise children as a mouse in an basement”. ”Finding food” would require subtasks as explore area, and remember places where food often arrives, ”don't get eaten” would require subtasks as find shortest way to a hole, hide, stay quiet, don't go too close to predators. This way we can try to find the cognition requirements for the given context, and then measure how well the system fulfills these requirements. If a real mouse fulfills the requirements perfectly we can consider it as a cognitive being in the ”being a mouse” context. However, in another context, if it tries to play chess, it would fail miserably, and would not show any cognitive behavior related to the ”chess playing” context (it would however still fulfill the task of being a mouse). At the same time a chess player would probably starve to death as a mouse in a basement, and would struggle to fulfill the contextual task of being a mouse. We can therefore say that we measure the level of cognition based on how well the system performs given a contextual task. Another important factor is how much cognition does the task require? The task of being a rock could be described as ”if on the ground; stay, and if in the air; fall”. The inputs would be ”on the ground” and ”in the air”, and the outputs would be ”stay” or ”fall”. Even if our computer system can model this behavior through the use of newton laws, we would not consider it to have cognitive abilities. However if we look at ”being a cat”, the cognitive contextual requirements are much more complicated, and by fulfilling them, one can say that the system has cognitive behavior. Today a lot of doors open automatically. The simple sensors and motors have replaced the need for a door opener person. The task of opening a door for someone when he or she arrives at the door is so simple that we don’t think of it as cognitive value, but regarding the cognitive contextual task, the automated door recognizes and understands that there is someone in front of the door, and that it needs to be opened. What is important to remember here is that even if a human door opener probably could talk and have a sense of humor, our automated door opener doesn’t need these abilities since the task of opening a door doesn’t require them. To summarize, we first have to identify the requirements of a given task in a given context, and then measure a system’s abilities to fulfill these requirements. If the requirements are fulfilled, the level of cognition required to fulfill the task can be measured and quantified.

4.2.2 Cognitive vision and automated video surveillance

As described in the previous section, we should now try to identify the task of video surveillance and the cognitive behavior required to fulfill this task. The video surveillance system (normally consisting of video cameras and human operators) is required to detect a certain set of events and then report these to guards that can handle the given case at the given location. An event is something that happens in our environment, and every describable thing that happens could be called an event. For example ”a man standing still” is an event, ”a man lifting his hat” is an event and ”nobody touches the wall” is an event. The endless set of possible events contain a subset of events that we are interested in. Since there is no reason to alert the guards if a man lifts his hat, the system should not attempt to detect this event, however a fight event would be interesting to the guards and the system should attempt to detect it. The system’s output is the alerts given to the guards, and the system’s inputs consist of multiple video feeds and possible audio-feeds, but also feedback from guards related to false alarms and new types of events that the system needs to detect. A simple short time measure of the system's success could be the number of false, missed and correctly alerted events. In a long term perspective, the system would also require a level of adaption. New types of events should be learned based on feedback from guards, and false alarms should be reduced through experience. To prevent certain types of events, new rules could be introduced and rated as very important, for example; firing a gun could be the actual event we want to prevent, but forbidding guns could be the method. One could assume that this kind of policies would change over time, and the system should be able to adapt to these. Video-feeds as input could require that existing camera quality is sufficient and environmental variables like light, shadows, people in scene, architecture and camera placement can't always change to fit the surveillance system's best. The system should also be robust and tolerate changes in these variables. Some configuration from an operator is acceptable in these adaptations, but by automating video surveillance, the system should not need to be watched. The alarms sent to the guards should also contain more information then just being an alarm. What event is detected should be reported, where the event happened should be reported, the appearance of the actors should be described, and where they went after the event should be reported. This kind of explanations require that the system to some degree understands what is happening. If the description of the actors is sent as pictures, it still fulfills its task, so there is no reason to copy the human operator exactly. A live map with all the actors visible could actually be an improvement of the human guards' received descriptions by speech over radio. The guards should also be able to query for additional information, either over radio, or maybe through their PDAs. What is important is that the guards receive enough information to handle an event, but also to prepare and organize as they approach the event's location. If the system could identify events which would require an ambulance, it could either call the ambulance directly, or hold a telephone line ready for the guards to call over radio. The final important requirement is the real-time detection of events. The guards needs to be alerted as soon as possible, and no processing-delay is acceptable. A non-adaptive system that only detects left luggage fulfills only a very small part of these requirements, and should not be considered having cognitive behavior

Cognitive behavior measurement matrix for automated video surveillance	
Nr.	**Description**
1	Detect specified events from video-feeds
2	Alert guards when events are detected
3	Learn new types of events, and improve detection of events over time
4	Adapt known events to new conditions and adjust performance to improve levels of false and missed events
5	React to feedback from guards and supervisors concerning policies and results
6	Work robustly in a real-world environment with noisy and missing data
7	Run in real-time, and detect events with no or little processing delay
8	Distinguish between different types of events, and act properly to the event detected
9	Deliver a detailed explanation of what is detected to the guards
10	Aid the guards in handling the alerted event

Table 4.1: Cognitive behavior measurement matrix for automated video surveillance

in our video-surveillance context. However, as an aid to a guard operator it could still be useful, but not regarding the cognitive contextual task of being an cognitive automated video surveillance system.

To measure these requirements we will now create a matrix of requirements and fulfillments that we will use while designing my system, running experiments, interpreting results and comparing with other automated video surveillance systems (Table 4.1).

I will now discuss each of these points regarding our previous definition of what cognition is. If we look at the idea of our system's abilities to understand, one could say that if the system can detect different events with a high rate of success, it understands these events. The word Understanding, from the definition of cognition, is describing the system's awareness of what is happening. For example, if the system can detect an event, the system understands that the event is occurring, and therefore understands the event. The term is vague, but since it is used in the definition of cognition, we need to establish an idea of what it means that our system understands something. What is going on under the hood is not important to our cognition measurement, so from the observers point of view, if the system classifies the events correctly, it must have some understanding of what these events are. By also providing an detailed explanation of why an event was detected, the system understands, in some form, why the events happening. The system's understanding of what is happening is not deep or complete compared to a human's understanding of the same situation, but we could still say that the system to some degree understands what is happening: ”The system understands that there is a fight”. This meaning of understanding is tightly related to interpretation and perception as in the meaning ”I understand the Italian language”. Understanding Italian basically means that you can detect and translate the different words of Italian into your own language, just as our cognitive video surveillance system perceives its environment, and translates what is happening there into a set of detected events. According to the definition of cognition, the system's understanding and knowledge should be acquired through thought, experience and senses. Our video-feed from which we detect the events and the camera which records this video-feed would be considered as our senses (or our sensor), but one must also remember that the feedback given by the system's operator while the system is learning, is delivered through the system's ”senses”. By learning and adaptation we can show how experience is used to gain accuracy. Our definition of cognition says nothing about the real-time aspect of video-survaillence, but we also included the ability to run in real-time as a requirement, because an automated-video-survaillence not running in real-time is probably too slow to alert any guards inside a practical time frame.

4.2.3 Solving our cognitive goals

By reviewing our measurement matrix of cognition in video-surveillance, we can identify how our chosen strategies should fulfill the measurement matrix requirements (Table 4.1).

From these requirements, we can identify a few overall concepts. The system must handle real world data, the system must learn, and the system must ”think” in a way that can be explained to a human. The overall task is of course to detect a set of selected events.

Requirements 1 and 2 formd the outlines of our cognitive system's inputs and outputs. Our system will aim at detecting events, but when the event is detected, no action will be made. This is because the task of actually sending an alarm on the detected event to some guards is outside the scope of this thesis. Even if the main task of our system is to detect events, the learning and adaption is as important (not being able to detect new types of events actually means that these events not are detected).

Requirements 3, 4 and 5 are about learning and improving performance with experience, but also the possibility to re-learn and adjust. We have chosen two methods to attack these requirements on two different levels. At the lower level, we use ANNs to transform features detected by video-analysis systems into describing-sentence-categories understandable to humans. The categories are specified by the system's operator, and they are learned through training sessions. After our describing-sentence-categories are formed, we use CBR to reason on these categories at a symbolic level. Our implementation of CBR will use experience only gained from a training session, and then when actually running, all situations will be compared to previously experienced situations. Learning new events, re-learning old events, and adjusting to false/missed events would be done by known CBR methods.

Requirements 6 and 7 are about the environment where our system is supposed to run. The system must run on real video and in real-time. By accepting the fact that the input video is full of noise and that our detection-algorithms are far from perfect, we try to create a system that relies on a large amount of input, and that remembers so that the system is less reliant on a single feature in a single video frame. By feeding our input data to an ANN, we hope to handle the issues about noise and missing data at an early level of our analysis, so when we start reasoning in the symbolic CBR level, the data is information rich and strong in the sense that small errors in the lower levels of the event detection process are handled, and are not amplified. It should also be mentioned that except from when training, both ANNs and CBR are fast and uses no complicated calculations during regular runtime. The CBR search-space is finite and can be minimized through known CBR techniques that reduces the number of stored cases without decreasing performance. The ANN doesn't have a search-space, and runs with a speed only proportional to the number of nodes and connections in the net.

Requirements 8 and 9 are about the system's internal representation and its ability to express this representation. By introducing the category stage of our processing, we try to first generate categories that are understandable by humans, and then we reason on these categories by comparing with earlier experiences (a process also similar to how humans often reason, and because of this also understandable, in contrast to for example a statistical explanation).

Requirement 10 is about what happens after the event is detected and the alarm is fired and will not be explored in this thesis. How the system communicate with the guards is considered to be outside the scope of our experiments.

4.3 The overall Design

The overall design of our system can shortly be described as a five step process.

1. Analyze real world video and find objects.
2. Track objects over time.
3. Mine object-features for each object for each video frame.
4. Use ANN to map objects' features into objects' categories.
5. Use CBR to reason on objects' categories to detect scenarios.

The objects' features will from now on simply be called features and the objects' categories will from now on simply be called categories.
The Functional Capabilities of Cognitive Vision in (P Auer et al, 2005) states the following functional requirements:

- Detection and localization
- Tracking
- Classification and categorization
- Prediction
- Concept formation and visualization

- Inter-agent communication and expression
- Visuo-motor coordination (concerns embodied cognitive systems only)
- Embodied exploration (concerns embodied cognitive systems only)

Detection and Localization is described as *"the system's need to be able to discriminate between regions in the visual field on the basis of color, texture, motion, or depth"*. The ability to detect and locate areas of interest is the first pass of our process. By analyzing the real world video, we filter the image and find *objects* that are interesting for further investigation. Detecting unexpected visual entities is also described as an important part of this capability. Next, "a system should be capable of detecting hard-wired features.". While analyzing objects, a set of features are extracted by hard-coded algorithms. These features are basically shape, speed, size and location and could be described as numerical data. *Tracking Capabilities* is described as *"A cognitive vision system will have a hard-wired tracking capability that can track pre-determined regions or features in continuous 4-D spatio-temporat paths (or lower dimensions, depending on the particular approach to cognition vision being pursued)"*. Our step of tracking objects over time fulfills this requirement, and *connects* object A found in frame X with object A found in frame Y. *Classification and Categorization* is described as *"Classification of visual entities ... [and] ... the capability to classify behavior."*. It is also pointed out that classification should be able to adapt and change, this refers to learning new classes and behavior types to detect. Our final two process stages try to fulfill these requirements through a hybrid between symbolic and sub-symbolic approaches. *Prediction* is described as *"being able to interpret the intent underlying behavior specifically to predict future spatio-temporal configurations of the visual environment"*. Prediction is not prioritized in our system due to the fact that if the system can predict an unwanted event, it should alert the guards immediately. Because of this there is no practical difference between a detected and a predicted event. At a lower level in the tracking stage of our process, predictions are made on the future location of objects, but this is only done to make the tracking better and the predictions are not passed on to higher levels of the system. *Concept Formation and Visualization* is described as *"A cognitive vision system should be able to* *create some form of conceptualization of its perceived environment and it should be able to visualize those concepts"*. An internal representation of the current perceived environment with visual entities and relations between these entities are formed and reasoned upon by our system and they are *visualized* by the system. *Inter-agent Communication and Expression* enables the communication with other *agents*, in our case the guards.

4.4 Concepts

4.4.1 Scenarios

Let us start from the end of our process-chain and with Scenarios. A Scenario is a situation/event that we want to detect. In our fight example, the scenario would simply be "a fight". The system's goal is to detect scenarios in videos, and the scenarios we wish to detect will span a range: walking on rails, leaving luggage, people fighting, people falling over and tagging on walls. In the final video surveillance system, these events would fire the alarms that are passed to the guard when the events are detected. The scenarios we want to use are (Inspired by Fuentes and Velastin (Fuentes and Velastin, 2003)):

- Fighting.
- Left luggage.
- Walking on track.
- Vandalism through spray painting walls.
- Lying down on floor.

4.4.2 Objects

Objects are the fundamental structural component for our system. A person walking around in the video being analyzed is an object, a bag that is left in the cameras view is an object, and a passing

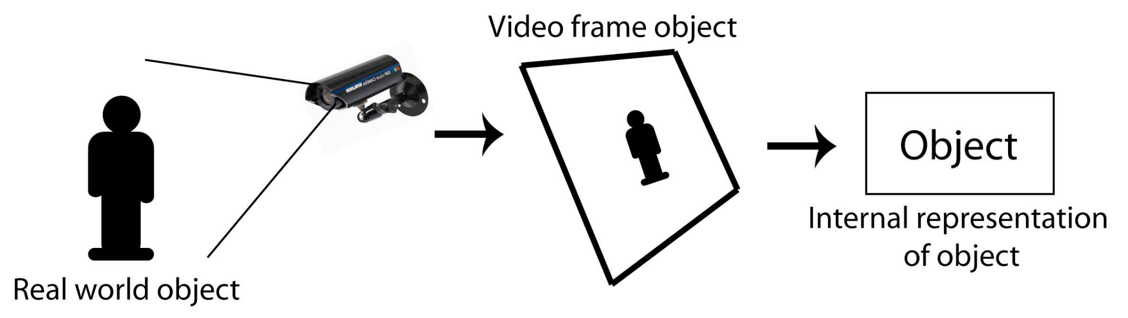


Figure 4.1: Relationship between a real world object, a video frame object, and the internal representation of an object

train would be an object. A poster on the wall or a bench would not be considered as an object, since these are static environmental entities. All static parts of the scenery will be ignored while finding objects since the underlying perception system starts by segmenting out areas in the scene, different from the static background. This description of objects relates to real world objects, but internally the system also represents these real world objects with an internal representation we call object. To distinguish the two meanings of the word object, we will use the term real world object when referring to objects of the real world. The system's perception of the real world object happens through the video input. A real world object will be captured in a video frame and then this video frame object will be used to update the system's internal representation of the real world object.

An important aspect of internally represented objects is that they are maintained over time. Person A from video-frame X is internally the same object as Person A in video-frame X+1. This means that an object also has a history, and will be the structural element that binds the actions of Person A at time X with the actions of Person A at time Y. In our fight example, the two (or more) persons fighting are the objects of the fight event. The objects are traced from the moment they enter the video frame until they leave the scene. For this whole period, the internal representations for both the real world objects are maintained and updated by analyzing video frame objects. The fight could be considered as a special relation between these objects, but these details will be described later.

4.4.3 Features

So what do we know about our objects? We know their features. Features are the basic data that we know about each object for each video frame. Features concern an object's size, shape and movement, its arrival and exit time, and its relations to other objects and scene elements. Let us think of features as the most basic object data. Initially for each video frame, features like shape and size are measured with numerical data, for example the object could be 183 cm tall, and could move at 5 m/s. These numerical data features are then converted to a common feature representation, and these commonly formated features are what will be called features from now on. The initial set of numerical data features will be referred to as numerical data. All features are represented in a common way, with a probability. That means that the speed will not be given in meters per second, but in the probability of being fast, slow or static. By probability we mean the system's certainty and not a statistical calculation. If the system is 95% sure that the object is *wider then tall*, the probability of the feature called *Shape Wider then tall* is 95%. All objects will for each video frame receive a full set of features and their probabilities. These features are then used to find the objects' categories, which finally are used to recognize scenarios.

The features we want to use are (Inspired by Fuentes and Velastin (Fuentes and Velastin, 2003)):

- Shape wider then tall
- Shape taller then wide
- Shape human (width vs. height ratio of a upraised person)
- Size small
- Size big
- Size human (ca. 2 meters on the diagonal)
- Speed slow

- Speed fast
- Speed relation (between last frame and last 50 frames)
- Position at track
- Position at gate
- Position on wall (not floor)
- Position at bench
- Arrive (Have just arrived)
- Exit (Have just exited)
- Join (The object is a joint of two objects: a group)
- Part (The object just parted with another object)

4.4.4 Categories

Categories can be described as advanced features. A category is constructed by several features, and should be considered more high level than features. In the same ways as features, a category describes an object, and is a part of the objects description. One example of a category is an object's probability of "being a person". This category can not be inferred directly from one single feature, but should use several hints from several features. Size, shape and movement all give hints about an object being a person and not a bag. By combining information from all different feature-types, the probability of the object's category "being a person" should be decided. So why do we need categories when we already have the underlying features? It is important here to point out that the categories don't contain all the information of the underlying features, but are limited to the information we are "mining" for, what we want to know. For example we are not directly interested in a person's height, but its height can help us determine if the object is a person, or if we know that an object is a bag, we are not interested in what kind of bag it is (big/small/long/short). An object's set of categories contains the information of the object's features, but in a cleaner and more information rich form. For our fight example, knowing that it is two persons interacting, and not a train and a bag can be considered as the first step towards detection of the fight scenario. Our first category is called PERSON, and should have a high probability on objects that are human.

The categories we want to use are:

- PERSON (Is a person)
- WALKING (Is walking)
- STATIC (Is a static object: not moving)
- GROUP (Is a group)
- MOVESIRREGULARLY (Moves in irregular ways: jumps around)
- LEFTBYOTHER (Is left by another object)

4.4.5 Cases

Since we are using Case Based Reasoning we are going to use the concept of a case through the text. It is therefore important to clarify what a case is, independently of CBR. In its simplest form, a case is a set of objects with their categories and features. The current case is the current visible objects and their current features and categories. In CBR cases are matched with previous experienced cases, to see if we earlier have seen similar events, and if these events should be reported. What is interesting to note here is that in our fight example there are two or more fighting persons is the detected case. These cases should be independent of if another person waits in another part of the camera's view. This means that each video-frame contains a lot of different cases (dependent on the number of objects in the scene) but most of these are not interesting, and will not fire an alarm. Let us think about a case as something that could be a scenario that we want to detect, but most of the time it isn't.

4.4.6 The truth

In some way, the system need to know what it is supposed to detect. Symbolic systems, including rule systems, where the operator models the truth, will be discussed, but also learning systems that learn from example will be discussed. In both approaches, the operator need to tell the system: ”This is a fight, and this is not”. In a rule system example, where the rules are modeled manually by operator, the rules are the truth: ”If somebody leave his/her luggage, sound the left luggage alarm”. In our approach we want the system to learn from example, so the examples becomes the truth: ”This example is a fight, this example is not”. We will refer to this data as truth data. When dealing with truth data features, we call them truth features. When we are dealing with truth data categories we call them truth categories. In our learning approach we which to train the system by showing it examples: ”This is a fight, this is not a fight”. The example event, and the teacher's comments form what is often referred to as training data. By truth data, we are talking about the teacher's comments, and not the example itself.

4.5 Video Analysis

The first step of our event detection is to analyze the video. There are several approaches on how to analyze video, and we have chosen to use object tracking as our method. Instead of analyzing all image frames of the video independently, we try to collect information about an object in the scene over time. Because of noise and faulty analyses, a single frame could easily consist of a lot of errors that we want to be robust to, and a more detailed description would require more hard-wired algorithms. Examples of more advanced low level algorithms could be facial expression detection, skeleton detection, body part segmentation and texture descriptions. By focusing on tracking an object over time, we collect a few attributes that over time give a solid mean value as result. We can also focus on speed and movement behavior, and we know that it is the same object that does something even if the actions are stretched over time. With this approach, the system will know that a person is walking on the track, but not what clothes he or she is wearing. Object tracking for video surveillance is both described in (Fuentes and Velastin, 2003) and (Greenhill, Renno, Orwell and Jones, 2005).

Initially we wanted to extend an existing video tracking system, but we later discovered that it was impossible to access the desired existing video tracking system, and the data sets used by this system. Because of this we had to create our own video tracking system to work as our underlying perception system.

4.5.1 Blobs and trajectories

The results of video object-tracking analysis are blobs and trajectories. Blobs are areas in the video image frames that contain video frame objects. If a video frame is analyzed well, it will result in one blob for each detected real world object in the video frame. Predictions on previously detected objects' next positions are made and stored as hypotheses in the *tracker hypothesis engine*. By feeding the new blobs to the *tracker hypothesis engine*, matches are found between blobs and hypotheses, and a blob that matches an object's prediction hypothesis is used to update this object. The object is updated and maintained over time. By remembering previous states of the object we create a trajectory. With this trajectory, we determine where the object has been, we predict where it is heading, we can find its average size and shape, and we can use it as a structure to connect parts of events that occur at different times.

Object tracking and trajectories are explained in (Fuentes and Velastin, 2003).

4.5.2 From Image-plane to Ground-plane

The blobs initially exist only on the image-plane: that is, the plane of the 2D video-frame. A blob has a width and a height, but no depth. This representation gives us some problems, for example the same object viewed close and far away has different sizes so there is no way to know the object's

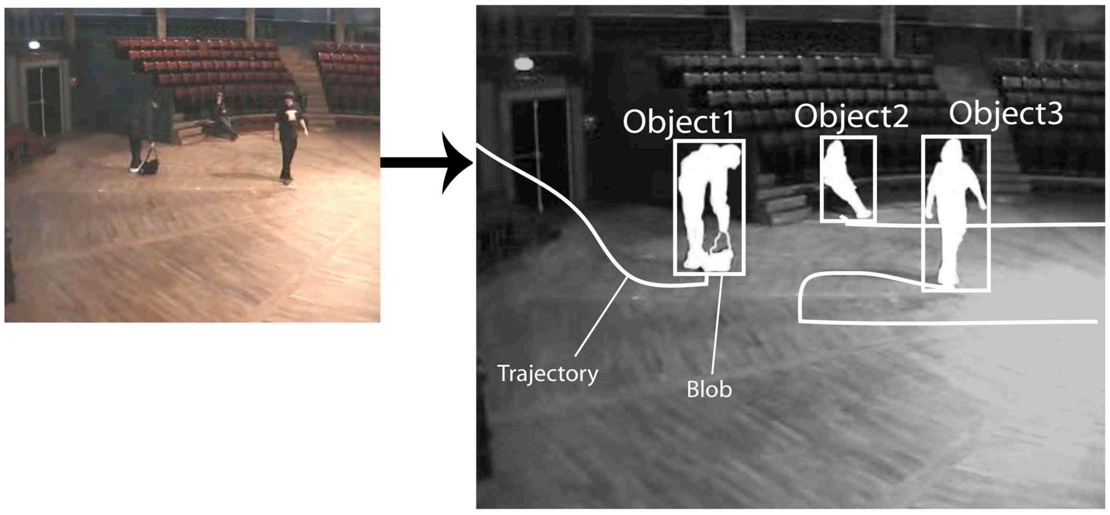


Figure 4.2: Example of blobs and trajectories, segmented and tracked from a actual video feed in this study.

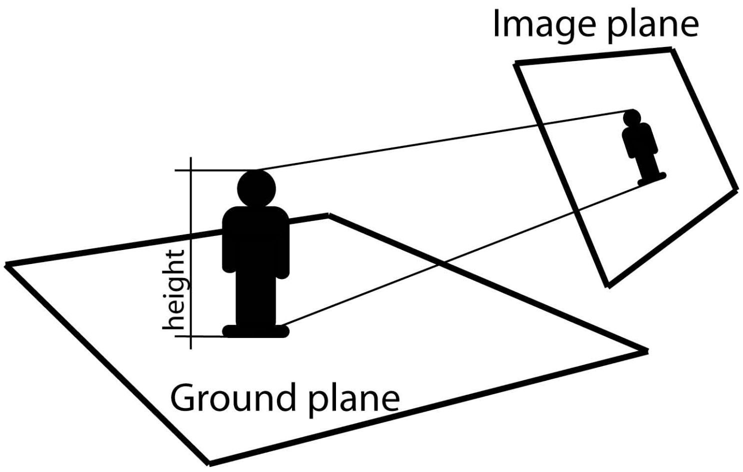


Figure 4.3: Relationship between image plane and ground plane

real world size based on this plane only. Another problem is the distance between two objects; in the image-plane, two objects standing far away from each other could appear really close. By modeling the cameras perspective view of the world, a transformation from image-plane to a ground-plane is created (Greenhill, Renno, Orwell & Jones, 2005). This transformation is calibrated when the camera is installed, but other research tries to create self-calibrating cameras (Renno, Remagnino & Jones, 2003). The simplest and used implementation of this transformation is to find out what corner on an image-blob that is closest to the real world ground-plane, and then transform this point from image plane coordinates to ground plane coordinates. Once we know the scale of this point (how long a centimeter would be at this point), we translate the blob's width and height from image-plane sizes to ground-plane sizes. To get the height, the camera must look at the scene from the side, but this is also true for a real guard. Another problem with this simple method is that if the person jumps, it would be translated as movement on the ground-plane away from the camera, and not up in the air. We know that this method isn't perfect, but with ground-plane positions in the trajectory and real world dimensions describing the object, our knowledge about the object is increased. Another good side on having ground-plane positions is that we can derive it to get ground-plane speed. This might be noisy, but by looking at averages over N frames, we get a good idea on the object's actual movement.

4.5.3 Overlapping in the image-plane

One problem with blob detection from video frames is that with simple filtering techniques two real world objects overlapping each other in the video frame only can be detected as one. We call this blob, which contains multiple real world objects, a blob group. By analyzing internally maintained objects and their trajectories, we detect when two objects form a group and when they part from the group. It is important to remember that this group is in the image plane and that the included objects could possibly be far away from each other in the ground plane. This could possibly be solved by looking at object height, ground-plane position plus other visual features, and then make predictions on where objects will be when they exit the group and where they are when inside the group (Corvee, Velastin and Jones, 2003). A simpler solution could however be less complicated and better for our use. An often used movie-effect is that a person hits another person in the face, but if you look at it from another angle, you would see that he actually misses. By accepting these kinds of miss-readings of the scene, we create a much simpler group-detection system that takes advantages of the overlapping-problem instead of being compromised by it. Also by knowing that this grouping isn't totally accurate, we know we have to use other factors then simply grouping to sound our alarm. This is also realistic to how a human guard would look at the scene: there is a higher probability that fighting occurs in a group then between people standing away from each other, but grouping alone isn't enough to sound the alarm. The group of real world objects would be perceived as a single object by the *video analyzer* in the input video frame. Instead of trying to represent this group as two objects, we simply store it as one single new object. The two real world people in the group is now a single object internally to the system. By mining the group object's features, a category named GROUP is inferred. A group of two people is probably wider then a single person, so it is possible to determine that an object is a group based on its size and shape features. If a person lifts a bag from the ground, the bag object and the person object will form a new object (just like a group object). This time we will not be able to tell that the new object is a group, since a bag doesn't significantly affect a person's shape or size. In this situation, the new object would have similar features to any person with or without a bag, so the system would think of this new object as a person, and not a "person and bag" group. When two objects join into a group, the new object's join feature is activated, and when an object split into two smaller objects, both these objects' part features are activated. A case where these join and part features would be very informative is in the case of left luggage. A static, non-human and small object parts from another human object. The human object then leaves the scene. We know that the static non-human object couldn't get there on its own, and since it spliced from a human object, we can assume that the human left the static object. Leaving your luggage for a second or two isn't illegal, but leaving your luggage, and then leaving the scene should fire an alarm for unattended luggage. Even if our group-detection system was limited to the image-plane, there are very few other possibilities for how this scenario could unfold.

4.5.4 Crowds

A problem with our blob-detection system is that it is currently unable to handle crowds. If one or two objects overlap in the image-plane, we can use this to form groups, but if the scene is filled with a crowd, the whole scene would be recognized as one big group, and little information about the individual objects or persons in the crowd would be known. Research is done in this field, and an attempt to match the data detected from single objects are investigated. By tracking faces and not full bodies, one could build a prediction model on how the blob covering the body connected to the face would look, and this prediction could be analyzed as a full-body blob. This would result in inaccuracies on height, shape and position, and some left luggage would not be visible to the camera, but these problems are similar to a human guard's problems. No attempts are made in our design to handle crowds.

4.5.5 Feature extraction and abstraction

So what kind of data are we going to extract from our video-feed? The data extracted from our video is divided into two parts: We call the first part structural scene data; consisting of objects, their appearance and disappearance, and objects joining groups and parting from groups. The objects are maintained over time, and their relations could be understood as a graph. The second part would be features inherited by these objects. The features detected would be size, shape, position and speed, and for each video frame, all objects are updated with new feature probabilities. The newly extracted features first consist of numerical data like height in centimeters and speed in meters per second.

An important part of generating a scene representation consisting of objects and features is that we are leaving the video-frame. From this point on, there is no relation between the data and where the data is collected. Multi-camera systems are researched upon (Remagnino, Shihab & Jones, 2004), and our representation allows that the data is collected from multiple video sources.

Research is done on calibrating multiple cameras onto the same ground-plane and then synchronize data so that we end up with one big scene (covering a bigger area the a single camera can record). It can also be presumed that multiple viewing angles of an object would increase the accuracy of the object and the object's features. One can imagine a big virtual surveillance 3D space, where a large number of cameras place objects on a shared ground plane space. Our design is limited to one camera only, but the data delivered from the single camera video analysis could in an extension of our design be delivered from multiple cameras. This is however not tested.

Finally our object's features are converted into a set of probabilities. Instead of storing that the object is two meter tall, we store a high probability of the object being human shaped. Instead of storing the meters per second a object moves, we store the feature "moves fast", the feature "moves slowly", and their probabilities. The same goes for small / big, wider then tall / taller then wide, fast / slow, etc. The final step is to compare the object's position to known scene objects like gates, train-track, benches or walls. Instead of storing meters from closest gate, we store the feature "is close to a gate" and its probability. These translations can be described as fuzzy membership functions, as part of the methods of fuzzy logic (Nagel, 2004).

4.6 Behavior recognition

By analyzing video, we extract a set of objects and their features from the video stream. Or goal is to detect a finite set of unwanted events from this input data (regardless of how low-level or high-level the input data is). These events are described as behavior, and the task is described as behavior recognition. There are many different ways to recognize behavior, and many different types of behavior to recognize. Behavior recognition is basically pattern recognition. An event is a behavioral pattern that is different from other events (everything is some type of an event). Our goal is to recognize this pattern. To recognize a pattern, the system needs to know the pattern. This could either be programmed by an operator through algorithms, rules or examples. The system's job would then be to translate its world input into the same *language* as the programmed events.

Declarative logic is a possibility where rules like: ”IF a AND b AND NOT c THEN d”, would define the premiss conditions of a, b and c required for d. The problem with this approach is that the hard part of the task is still not solved. Translating noisy real world input into clean and simple predicates is the hard part of the task. If we already recognize the pattern of a, b and c, the pattern recognition is already done, and the last step of reasoning is trivial compared to the underlying pattern recognition. The next step could be to write our rules to work directly on the numerical data features extracted our video like height in centimeters, speed in meters per second. The rules would suddenly explode into huge complicated beasts: IF (a is above 5.1 AND a is less then 7.2 AND b is more then 4.4) OR (c is more then 6.6) THEN d. To capture the pattern of an event like ”close to bench” would be possible with this approach, but the pattern of ”fighting” would require a large ruleset where the operator had to go through the data over and over to find all kind of threshold values. This process would basically be pattern recognition done manually by an operator, where the pattern is a fight. There could of course be low level features like ”hit in the face” or ”fist punch”, but that kind of features require a much more advanced video analyzer. The alternative approach, to manually model events we want to detect, is to learn the pattern through machine learning. Several approaches are possible. In this design we will use an ANN/CBR hybrid for event detection. In our design, the CBR is responsible for recognizing behavior that is similar to previously learned examples of this behavior / scenarios. The ANN recognizes categories, and these categories could also be understood as behavior or parts of behaviour. For example the category WALKING is a behavior recognized by the ANN, but the difference between a category (recognized by ANN) and a scenario (recognized by the CBR) is that the scenario is sent as the system’s final alerting output, while the category is an internal immediate recognition. The CBR with input from the ANN are what we can point out as the parts of our system that performs the behavior recognition.

4.7 Why learning?

The question should be: where and how should the system get its prior knowledge. If the system is supposed to detect event A, it has to have some prior knowledge of what event A is. The system could learn the event by looking at similar events, a programmer could feed the system with rules that describe the event, or the system itself could be programmed as an *event-A detector*. An important element here is flexibility and general purpose. An *event-A-detector* would have problems with detecting event B, and would have no possibility to do this. A learning system is much more general, and can learn both event A and event B. Now if our task was to only detect event A, a simple event-A-detector would be sufficient, but as stated in the Cognitive Surveillance chapter, this is not the case of our automated video surveillance task. If we look at the different events we want to detect, we can see that their complexity or simplicity to describe varies. For example detecting that a person is walking on the rails in a subway station would be possible with a simple and fast algorithm that tested if the person's position was *on* the track. However, writing an algorithm that detects fighting would be much more complicated. The same is true for a rule system where the operator manually enters rules. A rule describing falling on rails would be simple to model, but a ruleset defining a fight would be hard to model correctly. No claim is made that a rule set can't distinguish between a fight and a non-fight, but modeling this rule set based on numerical data is hard to do manually. By using learning, we bypass these modeling problems.

By training the system with videos of a real fight, and then telling the system that this is a fight, we let the system itself figure out what the difference between fight and non-fight is. This way the system might find small differences that humans don't know about. This small difference could be something that isn't an explicit part of a fight scenario (for example the relationship between speed and distance to closest person), but that gives good results when the system analyses real world data. If the system's assumptions are wrong, more training data is needed, so by keeping all the wrong classifications of events, the system can re-train itself with old and new training data. We don't need to know what our system *sees* when it detects a fight as long as it can detect it. Our design is supposed to learn what to detect and needs to be able to detect a large set of different events, also events that the system isn't explicitly designed for. To accomplish our goals of cognitive video surveillance, we need learning, and besides the fact that simpler solutions are possible for simple events, our system is supposed to be general purpose. A hard-wired non-learning system would not fulfill many of the requirements that we have decided for our cognitive system. The parts of our system that learns are the CBR and the ANN. The ANN learns categories as patterns in the input features, and the CBR learns by storing example cases of both events and non-events (There are always an events happening, but the system don't try to detect them).

4.8 Alternative approaches to behavior recognition

Our design uses an ANN/CBR hybrid to recognize behavioral patterns, but several different approaches have also been explored in literature. Hidden Markov Models (HMM), Bayesian Networks (BN), Pure Artificial Neural Networks (ANN), and Pure Case Based Reasoning (CBR) are all possible approaches. Implementation of a BN system with a discussion concerning HMM, BN and ANN can be found in (Bremond, Moenne-Loccoz and Thonnat, 2003). More symbolic and rule based systems can be found in (Nagel, 2004) or (M Borg et al, 2005), and CBR can be found in (Aamodt & Plaza, 1994). The span of different approaches to the problem is so large that arguing against all approaches which are not chosen would require a thesis of its own. Good results have also been shown by these other methods, so instead of arguing against them, we will try to argue why we chose the ANN/CBR approach. Using ANN/CBR hybrids in the task of video surveillance is not researched upon at a large scale. Many frameworks for how ANNs and CBR can be combined in a hybrid is discussed for example in (Lees and Corchado, 1999), but they are not used in the field of video surveillance. This is also true regarding what is called emergent systems (P Auer et al, 2005) in cognitive vision, so exploring these fields is in itself worth research, even if other methods have been shown to give good results. The next two sections (5.8 & 5.9) will focus on advantages of using ANNs and CBR, and will form the argument why these methods was chosen.

4.9 Artificial neural networks

4.9.1 Goals with using ANNs?

Our main goals of using ANNs are the following:

- Convert real world features into human sentence categories.
- Learn and ground categories from real world features
- Generalize to unseen situations
- Handle noise and missing data
- Handle temporal features

Our first goal is to lift the objects' feature data into human understandable sentences. The categories are learned from features that are collected from the real world, together with truth data entered by the learner. This connection between real world data input, and human understandable symbolic level of output is the main task of the ANN. Since real world data contains noise, and it might be incomplete, we have to be able to handle data with low quality, and we must be able to use experience from training data on testing data.

4.9.2 Inputs and outputs

As described earlier the features are stored as probabilities, and so are the categories and scenarios. One great aspect of this is that we represent a feature even if its probability is zero. This matches good with the static input and output structure of an ANN. By mapping each type of feature to a single input node in our ANN, the ANN monitors all features all the time (event if their probabilities are low). A low probability of the feature "is close to a gate", is understood as "not close to a gate". Because of this probabilistic representation, all features automatically have their opposites represented. On scales where there are more then two poles such as size, our probability-representation can be looked upon as a sort of fuzzy logic representation, where the probability of feature "small"

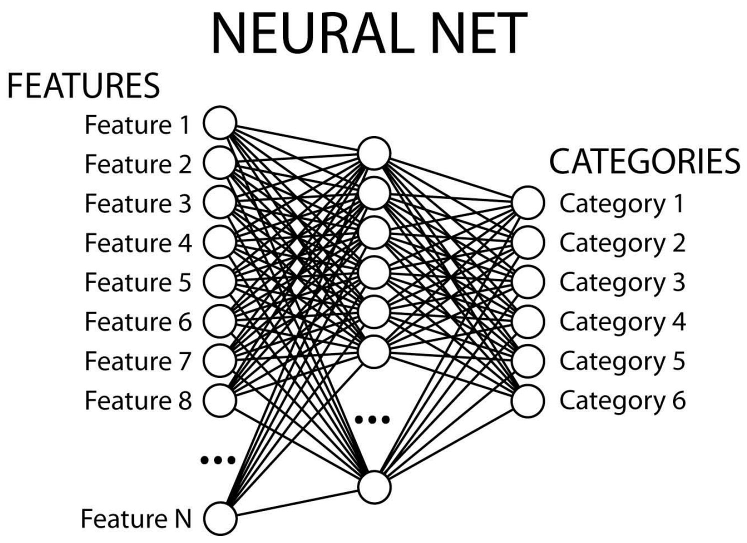


Figure 4.4: Example of an ANN, where features are fed as the input, and categories are created as the output

is strong on small sizes, the probability of feature ”human-sized” is strong around human size, and the probability of feature ”big” is strong on bigger sizes. For each video frame, the *video analyzer* updates the maintained object set with the current features found in the current video frame. All objects now have updated features, and we analyze each object with the ANN stage. For each object, the object’s features are fed into the ANN, and we get a set of categories as output. These categories are then stored in the object structure, and we can now forget the object’s features. After all the objects have been processed, we have a set of objects with categories, that are ready to be processed by the CBR.

4.9.3 Feed-Forward vs. Recurrent network

With a simple FFN (Feed-Forward Network), the object’s current features are mapped to the object’s current categories. This is simple and fast but can’t represent temporal data. The FFN will contain no information about what happened 2 seconds ago, and the same features will result in the same categories every time, independent of temporal context. This can be solved with Recurrent networks where some ”artificial” output-nodes of the net is fed as input. This recurrence gives the ANN the possibility to hold some memory of what has happened. For example if the net outputs a high probability for the category PERSON, this can be fed back, and then later we can reinsure that this is a person even if the features that provided this category's high probability is gone (this depends on the fact that a person can't change into a non-person in the real world). What is fed back is unknown to us since the net is self organized, and since the backward feeding happens inside the ANN. This new memory lets us learn new types of categories like ”have been running”. The self-organizing ANN has to figure out through training what categories are worth to remember (or maybe create an internal abstract concept that is worth remembering). The disadvantage of recurrent nets is that they require more training then with a simple FFN. In our final prototype we don't implement recurrent networks, but this is the first improvement of the prototype we would like to do, if the span of this master's thesis was expanded. Because of this, we leave the recurrent networks as a part of the design that was never tested or implemented.

4.10 Case Based Reasoning

4.10.1 Goals with using CBR

Our main goals of using CBR is the following:

- Reason on human understandable data in a human understandable way.
- Learn by example.

4.10.2 Inputs and outputs

The stored cases are objects with its categories. The stored case also contains truth data about the scenarios we want to detect. The truth data is what the system is told by the operator while learning; ”This case is a fight”, ”This case is not interesting”, ”This case is a bag left behind”. When we train the CBR, the cases are bundled with the given truth data, and then stored in the case base. When not training we query the case-base with the current cases, and we find the closest match based on the query cases' categories. This process is similar to a human thinking; ”Have I ever experienced a similar situation?”. For each querying case, we find the closest matching case in the case base, and since all stored cases are bundled with truth data we can now use the truth data of the matching case. This process is similar to a human thinking; ”Last time I experienced a similar situation, my teacher told me that there was a fight going on”. The final outputs of the CBR are a set of scenarios and their probabilities. If the scenario ”fight” has a high probability, we have detected a fight.

4.10.3 Simplifications in our design

In our implementation we are using is a very simple form of CBR that mainly consists of a K-nearest system, where K is 1. No knowledge-base will be used, and the Retrieve/Reuse/Revise/Retain circle will simply consist of storing cases while training, and finding a case's closest match when querying. More advanced CBR often use algorithms to remove error-cases from the case-base and to keep the size of the case-base at a minimum. Extending this design could easily be done by setting K to 5 and then calculating a probability of the right result based on the difference in the K matching stored cases, or by not storing cases if a query of that case gives the same result as the case we are going to store. The prototype implements the simple CBR with K as 1. Originally our goal was to use TrollCreek as our CBR system, but the simplifications in the CBR made TrollCreek a much to complicated system for our use.

4.10.4 Explanation of results

In a final working system, the system is required to deliver an explanation on why a scenario was detected. This step would not require a lot of change in our system since the categories already consist of human understandable concepts. By simply formating the categories' and scenarios' probabilities into text, the system could easily deliver an explanation in the form of: ”This object is probably fighting because, it is interacting with another person, it is a person, the other person is lying on the ground, this person is moving around a lot”. By pre-processing the case-base, a model of what categories matters could also be created by looking at a case and then vary one variable at

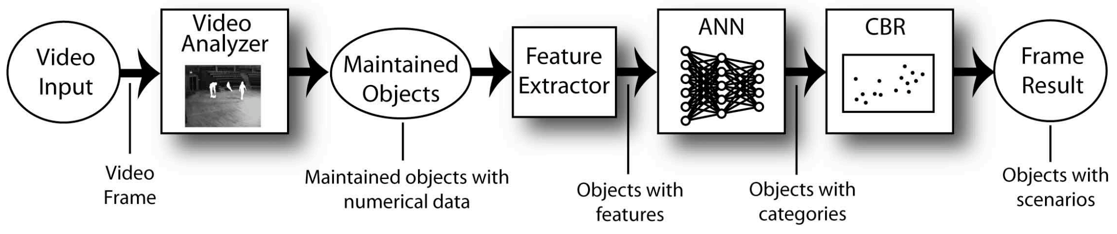


Figure 4.5: Process diagram of the running system.

a time to see if the result has changed (which would make this variable important to this detection). In the prototype, no attempts are made to explain the detected results.

4.11 The fight example in a running system

We have until now described the different concepts and parts of the design. In this section we will put them all together. Let us imagine an implementation of the system that is running in a real subway station. The system has just finished its installation, and the operators have trained both the ANN and the CBR with hours of recorded surveillance video. The subway station is empty when the system is finally started.

4.11.1 Process review.

Before we look at the fight example, let us review the system process (Figure 4.5). For every video frame, the whole illustrated process is executed for every detected object in the scene. Initially a *video frame* is captured from the *video input*, and then fed to the *video analyzer*. The *video analyzer* detects blobs in the video frame, and compare these with the currently maintained objects. If the scene is empty, no objects are maintained, and no blobs should be detected. If no blobs are detected and no objects are maintained, the process stops after the *video analyzer* is done, and the system waits for the next *video frame*. Let us imagine that a person enter the scene. The person is detected as a blob by the *video analyzer*, and the *video analyzer* looks for a matching object in the set of maintained objects. Since this is the first *video frame* where the person is detected, a new object is created that represents the real world person. This object is maintained until the person leaves the scene, and is currently only containing numerical data (a blob and a trajectory). The object would look something like this:

```
OBJECT [
    Identifier [Object1] (The object's internal name identifier)
    Numerical data [
        Current shape [BLOB] (Current shape from video frame)
        Trajectory [TRAJECTORY] (The object's shape history)
    ]
];
```

The set of maintained objects now contain one *object with numerical data*, and the object is ready for processing. Next time (next video frame) the *video analyzer* detects the person, a match will be found in the maintained objects, and the matching object will be updated with new numerical data (no new object will be created). After several frames, the trajectory will contain a history of the object's previous positions and we can visualize this as a trail left by the object while moving around in the scene. Now that we have an object with *numerical data* in our set of maintained objects, the object is ready to be processed. This is done from the time of the very first video frame, in which the object is detected. The object is first fed to the *feature extractor*. The *feature extractor* extract features from the object numerical data. Each feature's probability is calculated based on the numerical data, with the use of fuzzy logic. For example if the object is 50 cm tall, the feature ”Size small” will be given a probability of 90%, the feature ”Size human” will be given a probability of 40%, and the feature ”Size big” will be given a probability of 0%. The newly extracted features are stored as part of the object, and the object now looks something like this:

```
OBJECT [
    Identifier [Object1] (The object's internal name identifier)
    Numerical data [ ... ] (The object's numerical data)
    Features 1-N [ (The object's set of features and their probabilities)
        Size small [90%]
        Size human [40%]
        Size big [0%]
        ...
        Feature N [ ... ]
    ]
];
```

The next processing stage is the *ANN*. The object's features are fed to the *ANN*, and we get a set of categories as output. An example category is the PERSON category which is active if the ANN find the learned pattern of a person in the input features. The categories are then stored as part of the object and will be used as input in the next processing stage, the *CBR*. All the object's features' probabilities are fed simultaneously as input to the *ANN*, and all the output category probabilities are outputted simultaneously. Since the *ANN* is self-organized, we don't know the mapping between the input feature set and the output category set, but we try to give an example of how they could map. Let us look at the category ”PERSON”. A person's shape would probably be taller then wide, and about 1.8 meters tall. If the shape is taller then wide like a human would be, both the features ”Shape human” and ”Shape taller then wide” would have high probabilities. If the object is about 1.8 meter tall, the feature ”Size human” would have a high probability, and the features ”Size small” and ”Size big” would have lower probabilities since they describe sizes relative to a human size. A person's height, smaller then 1.8 meter, would give the feature ”Size human” a low probability , and the feature ”Size small” a high probability. The category ”PERSON”'s probability could be based on the feature ”Size human”'s and the feature ”Shape human”'s high input probabilities, but could also be influenced by other features. We have now added categories to our object, and the object would now look something like this:

```
OBJECT [
    Identifier [Object1] (The object's internal name identifier)
    Numerical data [ ... ] (The object's numerical data)
    Features 1-N [ ... ] (The object's set of features and their probabilities)
    Categories 1-N [ (The object's set of categories and their probabilities)
        PERSON [80%]
        WALKING [10%]
        STATIC [0%]
        ...
        Category N [ ... ]
    ]
];
```

The next processing stage is the *CBR*. The object's categories are fed to the *CBR*, and we get a set of scenarios as output. These scenarios are then stored as part of the object and will be used as part of the *frame result* (Figure 4.5). The object's categories are used to create a CBR case. The case could be looked at as an object with categories, but the object is only a container for the categories. The CBR finds the case in its case base that best matches the current case, that is; the case that has the most similar category probabilities as the current case. The matching case from the case base is bundled with a set of scenarios with probabilities, and these are now stored as a part of the object we are processing. For example the probability of the scenario ”FIGHT” is 5%, and the probability of the scenario ”LEFT BAG” is 59 %. This process is similar to a human thinking: ”Have I experienced a similar situation to the current situation?” and ”What was happening last time I experienced a similar situation?”. The truth data collected from the matching case is stored as part of the object, and the object would now look something like this:

OBJECT [

  Identifier [*Object1*] (The object's internal name identifier)

  Numerical data [...] (The object's numerical data)

  Features 1-N [...] (The object's set of features and their probabilities)

  Categories 1-N [(The object's set of categories and their probabilities)

  Scenarios 1-N [(The object's set of scenarios and their probabilities)

    FIGHT [*5%*]

    LEFT BAG [*59%*]

  ]

];

The last processing stage is to create the *frame result*. The frame result is a collection of all objects collected scenarios and would look something like this:

FRAME RESULTS [

  Object1 [FIGHT [*5%*] , LEFT BAG [*89%*]]

  Object2 [FIGHT [*15%*] , LEFT BAG [*0%*]]

  Object3 [FIGHT [*25%*] , LEFT BAG [*11%*]]

  Object4 ...

];

A threshold function is finally used to determine if the guard should be alerted. If the threshold level lies at 80%, the guard would now be alerted that ”Object1 is a left bag”, and should be investigated. After the *final result* is created, all objects are stripped of features categories and scenarios, and are now ready to be processed again, when the next video frame arrives.

4.11.2 The fight example

Let us return to the fight example. Two persons enter the scene and are detected by the *video analyzer*. The *video analyzer* creates and maintains two objects representing the two persons. For each frame the objects' *numerical data* is updated, and both objects are processed through the *feature extractor*, the *ANN* and the *CBR*. The two persons are just walking slowly around, so no events are detected. They are both standing up, so their ”PERSON” categories' probabilities are high, and

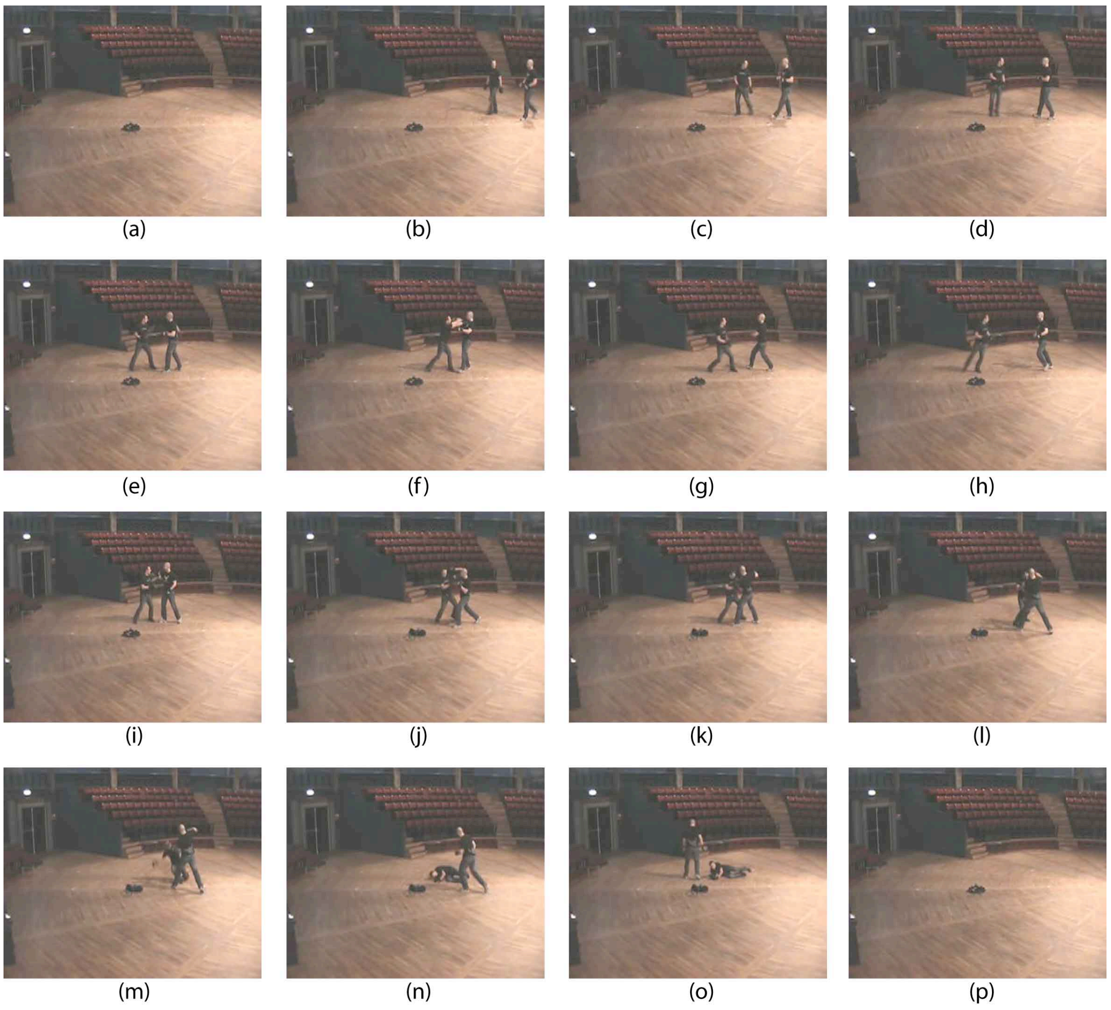


Figure 4.6: 16 Sample frames (a to p) from a data-set video containing a fight.

when they move, their ”WALKING” categories’ probabilities rise. For every frame the CBR compares the objects’ categories with previously experienced situations, but the previously experienced situations that only contain persons walking around doesn’t contain any high probabilities for either the ”FIGHT” scenario or the ”LEFT BAG” scenario. The objects are continually updated and processed, and for every video frame the system rechecks if any wanted scenarios are happening.

Let us look at some sample frames from a data-set video containing a fight (Figure 4.6). Frame *(a)* shows the empty scene, and frame *(b)* shows the two persons entering the scene. Frame *(c)* and *(d)* shows the two persons not yet fighting, and the system is still running normally, detecting and processing both objects every frame. This state continues until frame *(e)*. At frame *(e)*, the two objects are for the first time interacting. The moment the two objects overlap in the *video frame*, the *video analyzer* can no longer distinguish them, and creates a new object representing the two of them. This ”group” object is maintained as any object, until the ”group” splits into two individual objects or until the ”group” leaves the scene. We now have a group object, and the category GROUP's probability is probably high. The GROUP category could for example be activated by the group having the height of a human (feature: Size human), but at the same time being much wider then a human (feature: Wider then tall). In video frames (f), (g) and (h) the two persons get close and then they push themselves away from each other. The group splits back to being two individual objects. These frames are interesting, since we now have a relationship between the two objects, and they speed and direction changes rapidly. The features ”Speed ratio”'s probability is now hight. This feature measures the difference between the last 5 frames' and the last 20 frames' speed and direction. If you walk across a room and then suddenly turn and walk the other way, this feature's probability will be high. The category MOVES-IRREGULARLY's probability could be mapped from this feature, and maybe also the current speed (so that stopping isn't considered as moving irregularly). When the persons again attack each other, the categories GROUP and MOVES-IRREGULARLY should both have high probabilities, and these could be unique for a fight. When the group object's categories are matched with case-objects' categories in the case base, all cases found that have high probabilities of GROUP and MOVES-IRREGULARLY categories, have high probabilities of being a FIGHT scenario. When the FIGHT scenario's probability rises above a threshold the system detects the fight, and the final result created from these objects and scenario probabilities result in an FIGHT alarm pointing to the group object. The persons continue to be a group that moves irregularly until frame (o), where the one person walks away, and the other is lying on the ground. Until this frame the fight would have been detected (for each frame the system would output that the group object X is fighting). Now that they finally leave each other, and stop their rapid movement, both the GROUP category's probability and the MOVES-IRREGULARLY

category's probability becomes low, and the system stops to report the fight. The person lying down could also be reported as a LYING-DOWN scenario, this scenario could for example be based on the categories PERSON and LYING-DOWN. The category LYING-DOWN will not be tested in this thesis, but could be inferred from features like "Shape wider then tall", and that its size is somehow human (180 cm wide instead of tall). Finally in frame (p), both persons have left the scene, and the system is back in its initial state with no maintained objects.

4.12 Architecture

As described in (P Auer et al, 2005), identifying the architecture of the system is important. Architecture is described as the external systems, processes and steps surrounding the core cognitive vision system. The architecture is what the system requires to run, the minimal architecture is the minimal requirements for the system to run. For example, an automated video surveillance system requires cameras. It also requires a computer to execute, and the cameras must be connected to this computer. Some systems require that the cameras are calibrated, and if the system is a learning system, an operator or a teacher is required to train the system before the system can run properly. Communicating with guards over radio would also require an external radio system, and a radio interface for the system to use.

4.12.1 Cameras

For any video surveillance system, a set of cameras is required. The system could run on a single camera, but as described earlier, more cameras would give better coverage, and a better view of the scene. The cameras would require calibration for mapping between the image plane and the ground plane. This calibration could either be done manually, or learned automatically by the system (Renno,Remagnino & Jones, 2003). Calibrating for light levels and color difference is required, but this process can also be automated (Orwell, Remagnino and Jones, 2001). In our design we use a single camera, that we calibrate manually.

4.12.2 Truth data

Before training the system, truth data must be made for the system to train on. This require us to go through the data-set of surveillance video, marking the video with truth (Categories and Scenarios).

4.12.3 Training ANN

The next step is training the ANN with truth-categories. This requires a lot of video processing. Once this step is done, there is no need to repeat it unless new categories are required, or the learned categories are wrong. The ANN's topology and weights are saved to disk.

4.12.4 Training CBR

The final step in the training process is to train the CBR. This step requires one run through the training data.

4.12.5 Alerting the guards

When the system is up and running, and events are detected, the system needs to communicate its findings to the guards. We have not explored this final step in this thesis, but in a fully operational system this would be required as a part of the architecture.

Chapter 5

The Process and The Prototype

5.1 Introduction

In this chapter we hope to give the reader an overview of my master thesis process. As the rest of the thesis handles the final product, We hope to include the reader in how my thesis evolved into the final product. I will first look at the development of our design. We will include the first draft, and how this has changed over time. We will also argue for our final choices and why they were chosen over the old ones. In the next part we will tell the story about data collection, not only the final data-sets will be discussed, but also other previously explored data-sets not used later.The third part will cover the development of my experiment prototype. We will mainly focus on implementation details regarding the video analyzer, the ANN, the CBR, and the data-flow that connects these components.

5.2 Design development

5.2.1 Initial thoughts

In our first approach on combining ANN and CBR, we decided that the ANN's output should be the CBR's input. Our first idea was to encode each feature into zeros and ones, and then feed each feature as one input iteration to the ANN. The output would then be encoded categories where a combination of zeros and ones would represent a category. All scene objects and their features would be combined into a graph, so that a node representing feature A would be connected to a node representing object B. This graph could be described as a simple semantic network, with a single relation called ”have-feature”. Later connections between different objects could be created when these objects interacted. A ”is close to” relation between two objects was imagined. This representation was similar to TrollCreaks representation of a case. TrollCreak is a CBR platform developed at NTNU, Norway which is designed to possibly be used as an underlying general CBR framework in any CBR application, and contains a lot of general CBR functionality that we wanted to use.

A problem with the binary solution was the need of absolutes. Either a category was detected or not. Either a feature was present or not. These absolutes (only true or false) were not suitable in a domain with a lot of uncertainty. With absolutes, a person is either walking or not walking, but what if the person is walking really slow? What if the person is walking really fast? By feeding and encoding each feature and then output and decode each category we would only get a list of absolutes. We wanted to feed the net probabilities like the object is 50 percentage running or 50 percentage small (a little bigger then small). By allowing probabilities of each feature, the ANN would work as a set of soft rules instead of a set of hard rules. We were concerned that no hard rule (true or false) was able to describe our categories, and since we would know the features and categories in advance, no need to handle new types of features was required (since this already required extending the analyzer). We realized that the data fed to the ANN as input was not suitable for grounding. The reason to ground the net in real world data is for example for classification, where for example the net has learned horse and donkey, and then when learning mule, the net would handle this input as a mix between mule and donkey, since it had similarities to both. Our system would never experience ”a mule”, since no new feature types would be received. When the whole input-space is known, no grounding is needed since no unknown categories would be encountered. The graph representation also gave us some problems. If the whole current scenario was stored as one case, how would the system know what parts of the graph were interesting? If two people in the scene were fighting, while another was reading a poster, the system needed a way to know that the poster-reader wasn't a part of the fighting scenario. This was also true when we later wanted to find matching cases from the CBR's case-base. What part of the graph should we try to match against cases in the existing case base?

5.2.2 First design

To solve the problems found in our initial thought process, some changes were done before we decided on our first design. To keep the design and the system manageable, we decided to simplify rather then adding more complexity when seeking solutions. Each problem that arose could be solved with a new ad-hock solution. This approach of ad-hock solutions would lead us to a very complex and confusing system. Our first simplification concerned the ANN's inputs and outputs. We decided that each feature would be fed to the ANN as one input node. The feature's probability would decide the input node's intensity. This way, all features could be fed at once, and the ANN would at all time know all features (no internal storing of data inside the net). The same was done to the outputs. Each output node in the ANN represented a category, and its intensity decided the category's probability. The ANN now had a finite set of features as inputs and categories as outputs. We had to format each feature into a probability, but this was easy and resembled fuzzy logics in many way. Since all nodes' intensities now represented a probability of a feature or a category, our net was now well designed to handle non-absolutes like: ”The object is a little tall”. The net could for example map a desired level over multiple features into a category. This way our ANN would work as a set of soft rules, as desired. Our categories are no longer grounded in real world data, but since no unknown real world data would be fed to the network, no generalization at the input-level was needed. To solve the problems with what part of the graph is important, we decided to divide the graph of scene objects into multiple cases. One case would contain one object, its categories, and its closest related neighbor objects. By having one CBR-case for each object in the scene, each object had to be marked with its own truth-data, so that one object's part of a scenario was detected, not a whole-scene scenario. Since each case now consisted of a single object, its features and its closest neighbors, the whole case could be looked at as a point in an N-dimensional space where each dimension represented a category or a neighbor object's category. Because of this, the CBRs matching algorithm could simply consist of a "find closest point in case-space" algorithm. When we decided on this, we also decided to drop the complexity of TrollCreak, and simply implement CBR as a simple K-nearest system (with K as 1). We also allowed ourself to store *every* case from training data in our case-base. Research on how to maintain a small case-base without reducing the CBRs performance is already conducted, so we decided to ignore these issues in our prototype implementation and instead refer to research conducted by others (Zehraoui, Kanawati & Salotti, 2003).

5.2.3 Revisions

A problem we encountered after our first design was completed was how to handle temporal data. If a bag was left for a short time by its owner, no alarm should ring, but if the owner then left the scene, a "left-luggage" event should be detected. These kinds of temporal data required our system to have some sort of memory. One way to keep this information was to establish a relation between two objects that parted. In the "left-luggage" scenario, a relation would be kept between the bag and the owner, and when the owner left the scene, the bag object-case should be recognized as "left-luggage". Another example on how temporal data could be stored is that if an object is a person, the object will not stop to be a person. If the object is confirmed with 99 percentage probability to be a person in frame A, this should somehow be stored and be usable in future frames. The relations between objects was implemented in a simple way. Each object's case contained a set of categories and the categories of the object it last interacted with. This could be extended to the last 5 objects it interacted with, but for simplicity we only used one in our prototype. Related objects that disappeared was also kept for some time with a ”left” feature, so that the left-bag-case would contain information about its owner leaving. This simplification limits the system in many ways, but for our test-data it was sufficient. With a more complex CBR implementation these problems can be solved, but because of our prototypes size we decided to keep it simple. To store categories like ”is a person” over time, we introduced recurrent networks. By feeding the network’s output as input, the ANN can remember certain things, and can learn to remember specific things. What is important with this approach is that the network decides what to remember itself. By training the network for a long time, optimal internal representation of ”memory” would develop, and the outside of the network would only know that the network gave better results, not how it did it. By choosing what to remember, we could possibly bias the network in a non optimal way. These two changes in our design made it possible to handle temporal data, but these were in the end left as future work.

5.3 Collecting data

When we started to look for proper data-sets we ran into problems. Getting the right data ended up being a big problem; most actual subway surveillance data was restricted in use, and most open and free-to-use data-sets focused on simpler events then we wanted to detect (crossing room, walking, stopping etc.). People hanging around and walking across the scene was in no interest to us, since these actions aren’t the kind of events that we wanted to detect. The CAVIAR set is a good example (CAVIAR, 2004). These sets contains some fighting, and some left luggage, but no situations similar to fighting but without fighting. We were initially interested in a data-set with already marked truth data, so that we didn’t need to do the video analysis ourself. Kingston university’s papers told us that they had the data and the video-analysis system we could use. The rest of our system was formed so that we would only use features confirmed as detectable by Kingston University’s system. A trip to visit Kingston University confirmed our beliefs, but they could not give us their video surveillance data-sets, since these were of real people on real subway stations and hence were confidential. The data was restricted, and could not be shared. Our next step was then to create our own video data-sets with all the events that we wanted to detect. After creating our own video data, we implemented a ”simplified” video analysis system that gave the same output and was almost equivalent in functionality to the Kingston University’s system. We did not need to make this analyzer robust to environmental changes like light-conditions since Kingston University’s system is robust regarding these concerns; replacing our ”simplified” system with Kingston University’s system would simply be a programming task of integrating two systems (Both systems use XML as intermediate storage format, so translating one XML file to another would be enough). Because of this, all features we can detect are referenced to papers from Kingston University stating that these features can be detected. The final result concerning our data ended with ourself doing everything; Creating video data-sets, analyzing these sets with our own analyzer, and finally tagging everything with truth-metadata. This whole process ended up being a fair amount of the total workload, which limited other parts, for example a wider review of other existing systems, where more work could have been done. These possible improvements that we did not explore are described in *future work*.

5.4 The Prototype

To test our thesis, we decided that we needed to implement a working version of our design. Our theory was that an implementation of our design would have certain behavior and certain properties, and our prototype was the experiment where we tested our theory. As described in (Simon, 1995), AI is special in the way that the model and the subject that we model is the same thing. Our theory is that our model can behave in a special way, and we use our model to prove it. By introducing cognition, we also try to model the behavior of a video surveillance operator, but it is important to remember that we don’t model the operator itself. No claim is made that a video surveillance operator use neural nets and CBR, but a claim that similar behavior can be achieved with these to.

If we wanted to understand how the operator works, we should try to model the operator, but we are only interested in achieving the same behavior. The system is interesting in an AI perspective, where we try to model systems that can fulfill certain tasks, but is not interesting in understanding how the operator operates.

The prototype required the following functionality:

- A video player, capable of playing, showing and reading video files.
- A video analyzer, capable of segmenting objects and extracting features from video streams.
- A camera calibrator, capable of calculating and calibrating the relationship between image plane and ground plane.
- A scene setup system, where the user could label scene objects such as gates and track.
- A truth data editor, where the user could enter truth data into the video stream meta data, while looking at the video.
- A object tracking system, consisting of a hypothesis engine that calculated probability for two objects in different video frames to be the same object.
- An Artificial Neural Network Editor, for setting up the ANN's topology.
- An Artificial Neural Network system with error back-propagation.
- A simple CBR system with case storing and case querying.
- A trainer framework, capable of training the system with thousands of repetitions of videos and frames.
- A testing framework, capable of testing the system with 3 configurations, ANN, CBR or ANN/CBR hybrid.
- A logging system capable of displaying graphs.

The initial idea was to build our system on top of an existing video analysis system (already fulfilling much of the required functionality), but because this collaboration was troublesome, we ended up implementing all of the functionality myself. This resulted in a undesirable distribution of work between implementing and testing, just to get the system to work required much more work then that we wished to dedicate to the prototype.

5.5 Implementation details

In this section we will look at implementation details for three main components of our prototype: The video analyzer, the ANN and the CBR. The video analyzer will be described in three different subsections: Filtering the video frame, Hypothesis engine, and feature extraction. The prototype also contain a set of editors and visualization tools, but these will not be described in detail since their function is to simplify the user interaction with the prototype, and is not part of the video surveillance process. Details regarding how we obtain the video frame from the video file will also be left out since the Quicktime media library handle the major part of this functionality.

5.5.1 Data-flow

Our prototype processes a video file in tree passes: Video Analysis, ANN analysis and CBR analysis. These three passes were initially separated, but could easily be integrated into one. The reason we kept the passes separated was to simplify the conduction of experiments. While training the three passes are totally divided, so that intermediate data, for example the trained ANN's weights, can be stored for later reuse. While analyzing testing-data the ANN and the CBR run simultaneously on preprocessed data from the video analyzer. This separation between video analyzer and ANN/CBR was needed since the data delivered from the video analyzer had to be labeled with truth data. This truth data was needed so that we could measure the difference between the output from the prototype, and the wanted output. If the system was running in a real subway environment, the entering of truth data for error measurement had to be done *after* the real-time video feed was fully analyzed, but since we were going to test each video multiple times, we added the truth-data that described the wanted output *before* the ANN/CBR processing. Each passes' algorithms will be described in sections 5.5.2 to section 5.5.6.

Pass 1 : Video Analysis

The first pass consists of analyzing video and tracking objects over time. First the video frame was filtered and blobs were detected. Next, we inserted these blobs into the hypothesis engine which updated existing maintained objects. Finally features were extracted from the maintained objects' blobs and trajectories. Each video frame's maintained objects with features were stored to disk, so that we could add truth data, and reuse them later without processing the video all over again. The stored data passed on to Pass 2 was structured like this:

```
OBJECTS 1-N [
    OBJECT 1 [
        Identifier [Object1] (The object's internal name identifier)
        Numerical data [ ... ] (The objects blobs and trajectory)
        Features 1-N [
            Feature 1 [90%] (Feature type 1 and its probability)
            ...
            Feature N [30%]
        ]
    ]
    ...
];
```

Pass 2 : ANN

When the ANN pass was executed, objects with features were loaded from disk. For every object in every video frame, the features probabilities were fed into the ANN, either as training-data together with truth-data or as testing-data. Each object had a set of features, and after this step the object also had a set of categories. The categories were collected from the ANN's output, and stored as part of the object. The data passed on to Pass 3 was structured like this:

```
OBJECTS 1-N [
    OBJECT 1 [
        Identifier [Object1] (The object's internal name identifier)
        Numerical data [ ... ] (The objects blobs and trajectory)
        Features 1-N [ ... ]
        Categories 1-N [
            Category 1 [90%] (Category type 1 and its probability)
            ...
            Category N [30%]
        ]
    ]
    ...
];
```

Pass 3 : CBR

The last step of the process is to train and test the CBR. For every object in every video frame, the CBR is fed with categories detected by the ANN. The learning cases are stored together with the scenario truth-data in the case base. Later when the CBR is tested, the current case is compared with the learned cases from the case base, and the truth-data from the closest matching case is delivered as the final result. The final output data for every video frame was structured like this:

```
FINAL OUTPUT [
    OBJECT RESULT 1 [
        Identifier [Object1] (The object's internal name identifier)
        Scenarios 1-N [
            Scenario 1 [90%] (Scenario type 1 and its probability)
            ...
            Scenario N [30%]
        ]
    ]
    ...
];
```

5.5.2 Filtering the video frame

The video analysis was done by filtering the each video frame with multiple filters before using a hypothesis engine to find connections between detected blobs from different frames. The video filtering process consisted of the following filters:

- Background subtraction algorithm (subtracted the average video frame from the current video frame)
- Average noise subtraction algorithm (subtracted the average noise of each pixel from the video frame's pixels)
- A grayscale filter (reduced color image to grayscale image)
- A threshold filter (reduced the image to black and white)
- A erosion/dilatation filter (removed noise and filled small holes)
- A shadow filter (used original image to detect possible shadows, and subtracted these from filtered image)
- A flood-fill filter (detected segmented areas and created blobs by measuring bounds of these)

Background subtraction algorithm

The idea behind the background subtraction algorithm is that if we have a perfect image of the background of the scene (the part of the scene that we don't want to detect), and we subtract this image from a captured video frame, only pixels with a different color or intensity then the background will not become zero. The only reason for a pixel to change to be different from the background image is that an object that is not part of the background covers it. Objects not part of the background are what we are looking for, so by subtracting the background image, we find the pixels that are covered by these objects. To create the background image the following algorithm was used:

```
FOR EVERY COLOR CHANNEL c:
    FOR EVERY PIXEL(x,y,c) IN BACKGROUND IMAGE:
        BACKGROUND IMAGE PIXEL(x,y,c) =
            SUM OF EVERY CAPTURED VIDEO FRAME PIXEL(x,y,c)'S VALUE
            DIVIDED BY
            NUMBER OF CAPTURED VIDEO FRAMES
```

The *c* describes the three color channels: red, blue and green, which are filtered independently. The *x* and *y* describes the pixel position in the video frames and background image. The number of pixels are constant and the same for background image and all captured video frames. To minimize processing power a cumulating version of the algorithm was developed, where the captured video frame were added to a cumulated background image containing the sum of all captured video frames. To generate the background image we only needed to divide the cumulated background image frame by the number of video frames captured. The process was also stopped 200 captured video frames, and the background image was frozen. In changing light conditions, the background image had to be updated once in a while, but for our prototype this was unnecessary. To subtract the background image from the captured video frame the following algorithm was used:

```
FOR EVERY COLOR CHANNEL c:
    FOR EVERY PIXEL(x,y,c) IN CURRENT CAPTURED VIDEO FRAME:
        VIDEO FRAME PIXEL(x,y,c) =
            ABS[ VIDEO FRAME PIXEL(x,y,c) - BACKGROUND IMAGE PIXEL(x,y,c) ]
```

Average noise subtraction algorithm

After finishing the background subtraction filtering, there was still a lot of noise in the video frame. There was a lot of noise in the bright areas of the video frame, and little noise in the dark areas of the video frame. Patterns left by the video compression process were also appearing in regular places. To cope with this noise, an average noise subtraction algorithm was used, that was similar to the background image subtraction algorithm but processed the result of the background subtraction algorithm and not the original image itself. An average noise image was used to threshold the video frame individually for each pixel. To create the average noise image the following algorithm was used:

```
FOR EVERY COLOR CHANNEL c:
    FOR EVERY PIXEL(x,y,c) IN AVERAGE NOISE IMAGE:
        AVERAGE NOISE PIXEL(x,y,c) =
            SUM OF EVERY FILTERED VIDEO FRAME PIXEL(x,y,c)'S VALUE
            DIVIDED BY
            NUMBER OF CAPTURED VIDEO FRAMES
```

It is important to remember that this algorithm uses the filtered output video frame from the *background subtraction filter* and not the originally captured video frame. The chain of filters continues through the whole filtering process, where the output of filter X is the input of filter X+1. To subtract the average noise image from the filtered video frame the following algorithm was used:

```
FOR EVERY COLOR CHANNEL c:
    FOR EVERY PIXEL(x,y,c) IN CURRENT FILTERED VIDEO FRAME:
        IF VIDEO FRAME PIXEL(x,y,c) IS LESS THEN AVERAGE NOISE IMAGE PIXEL(x,y,c)
            VIDEO FRAME PIXEL(x,y,c) = 0
```

This filter was also stopped after 200 frames, and the average noise image was frozen.

Grayscale filter algorithm

Before we could use our threshold filter, we needed to convert the color video frame into a grayscale video frame where each pixel's light intensity ranged from value 0 (black) to 255 (white). To generate our grayscale video frame, the following algorithm was used:

```
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    GRAY(x,y) = ( RED(x,y) + GREEN(x,y) + BLUE(x,y) ) / 3
```

Threshold filter algorithm

Our next goal was to segment the image into *detected object pixels*, and *background pixels*. This segmentation turned the video frame into a black and white image, where white pixels represented the *detected object pixels* and black pixels represented the *background pixels*. This segmentation was done by filtering the grayscale video frame with threshold T. If a pixel's intensity was below T it turned black, and if its intensity was above or equal to T it turned white. Since most background image color values and average noise levels now were removed from the video frame, a threshold of T = 8 gave good results. To segment our video frame, the following threshold algorithm was used:

```
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    IF PIXEL(x,y) IS LESS THEN T:
        PIXEL(x,y) = 0
    ELSE:
        PIXEL(x,y) = 255
```

Erosion/Dilatation filter algorithms

An erosion filter ”eats” of white objects' edges, and thereby make them shrink. If the object is small enough (like a single noisy pixel) it disappear completely. The following erosion algorithm was used:

```
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    IF PIXEL(x,y)'S NEIGHBORHOOD CONTAIN A BLACK PIXEL:
        PIXEL(x,y) = BLACK
    ELSE:
        PIXEL(x,y) = WHITE
```

A pixel's neighborhood consist of itself and all eight surrounding neighbor pixels. White pixels have an intensity value of 255 and black pixels have an intensity value of 0. An dilatation filter ”adds” to white objects' edges, and thereby make them grow. If two objects are close, they ”grow” together, and small holes in the white objects are filled. The following dilatation algorithm was used:

```
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    IF PIXEL(x,y)'S NEIGHBORHOOD CONTAIN A WHITE PIXEL
        PIXEL(x,y) = WHITE
    ELSE:
        PIXEL(x,y) = BLACK
```

The erosion and dilatation filters were used by first filter with the erosion filter once, and then filter with the dilatation filter twice. After these filter were applied, most ”salt and pepper” noise was removed, and small image features close to each other had grown together.

Shadow filter algorithm

The most complex filter used was the shadow filter. Most objects in the scene threw shadows on floors and on walls, and these shadows were detected as not distinguished from the objects that threw the shadows. By investigating video frames we discovered that the shadows had much softer edges then the objects in the scene, so we wanted to take advantage of this. We also recognized that no shadows were thrown above any objects, and no shadows should be detected in really dark areas. The first step of the shadow filter algorithm was to create a shadow mask for the current video frame. The shadow mask was white where the algorithm thought there was no shadow, and black where there could be shadow. The following algorithm was used to create the shadow mask:

```
ALL PIXELS ARE INITIALLY BLACK
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    IF PIXEL(x,y) INTENSITY IS LESS THEN 10
        PIXEL(x,y) = WHITE
    IF ABS[ PIXEL(x,y) - PIXEL(x,y+1) ] IS MORE THEN 20:
        ALL PIXELS ABOVE PIXEL(x,y) = WHITE
```

By ”all pixels above” we mean all pixels in the current pixel’s column that are above (over) the current pixel. By combining the black and white shadow mask with the black and white video frame through a logical AND function, only white pixels in the video frame were kept white if the shadow mask marked that there was no shadow at that pixel. The following algorithm was used to filter the video frame with the shadow mask:

```
FOR EVERY PIXEL(x,y) IN VIDEO FRAME:
    IF PIXEL(x,y) IS WHITE AND SHADOW MASK(x,y) IS WHITE:
        PIXEL(x,y) = WHITE
    ELSE
        PIXEL(x,y) = BLACK
```

White pixels has an intensity value of 255 and black pixels has an intensity value of 0. After the video frame was filtered with the shadow filter, the white pixels in the video frame covered the objects in the scene not part of the background. There was still some small noise pixel groups, but these was rejected in the next *Flood-fill* filter.

Flood-fill filter algorithm

Finally we used a Flood-fill filter to identify connected regions in the black and white image. All pixels in a connected region was labeled with the same region-identity value, and all regions had a different region-identity value. The following algorithm was used to flood filter the video frame:

```
STEP 1: ALL PIXELS ARE INITIALLY MARKED AS UNFILTERED
STEP 2: FIND A WHITE AND UNFILTERED PIXEL(x,y)
STEP 3: FIND A NEW AND UNIQUE REGION-IDENTITY VALUE
STEP 4: MARK THE PIXEL AS FILTERED, AND SET ITS REGION-IDENTITY VALUE
STEP 5: FIND ALL WHITE AND UNFILTERED NEIGHBOR PIXELS(x2,y2):
            FOR EACH FOUND PIXEL(x2,y2) CONTINUE RECURSIVELY FROM STEP 4
STEP 6: CONTINUE FROM STEP 2 UNTIL NO WHITE AND UNFILTERED PIXELS ARE FOUND
```

By neighbor pixels we mean all eight pixels surrounding the current pixel. Each white pixel were now labeled with a region-identity value. To finally extract blobs from the video frame we needed to find the boundaries of each region. The blob is initially a rectangle covering its region pixels. The following algorithm was used to extract blobs from the filtered video frame:

```
FOR EACH UNIQUE REGION-IDENTITY VALUE V:
    CREATE BLOB B:
        B.ID = V (identity)
        B.XMIN = MIN [ x FROM PIXEL(x,y) WITH REGION-IDENTITY V ]
        B.XMAX = MAX [ x FROM PIXEL(x,y) WITH REGION-IDENTITY V ]
        B.YMIN = MIN [ y FROM PIXEL(x,y) WITH REGION-IDENTITY V ]
        B.YMAX = MAX [ y FROM PIXEL(x,y) WITH REGION-IDENTITY V ]
```

Finally all blobs covering less then 10 pixels were rejected, since these often were caused by noise.

5.5.3 Hypothesis engine

The detected blobs were fed to the *tracker hypothesis engine* that connected them to previously maintained objects. This was done through hypotheses. A hypothesis was suggested saying that new blob X was connected to maintained object Y. The probability of this hypothesis was calculated, and finally all hypotheses were sorted by probability: the best ones were accepted first, and hypotheses with a probability that was too low was rejected. If blob X matched maintained object Y, any other hypotheses containing blob X or object Y were rejected. While accepting the sorted hypotheses, the blob was labeled as *accepted*, and the object was labeled as *updated*. After all hypotheses were accepted or rejected, we were often left with some un-accepted blobs and some objects that were not updated. If a blob was not accepted by any hypothesis it could mean one of two things, either the blob was a new object that just entered the scene, or the blob was an object left by another object, for example a bag left by a person. If the blob was close to a gate in the scene, the blob was considered to be a new object entering the scene. If the blob was close to another object, the blob was considered to be splitting from the close by object, and a new object was created. This new object kept a relation to the object it divided from. A similar algorithm was used on objects that was not updated by any blob. If the object was close to a gate in the scene, the object was considered to be leaving the scene, and if the object was close to another object, it was considered to join a group with that close by object. The gates of the scene were labeled as gate-areas through an area-labeling tool in the prototype. When calculating the probability of a hypothesis, several variables were used:

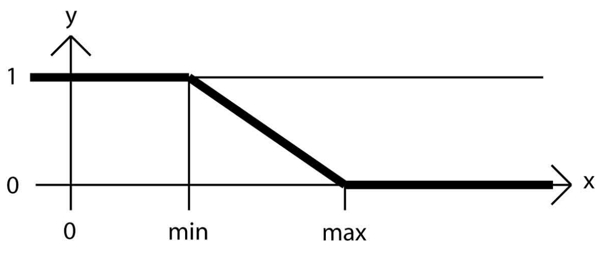
- The blob's percentage coverage of the object's last shape and position
- The distance from the blob's position to the object's last position
- The blob's width/height ratio compared to the width/height ratio of the object's last shape
- The blob's size compared to the size of the object's last shape (width*height)

When the hypothesis engine was finished, some objects were updated, some new objects were created, and some object were removed. Relationships between objects previously connected were also created and maintained.

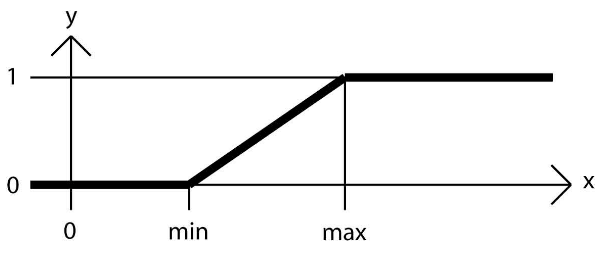
5.5.4 Feature extraction

After the maintained objects were updated, features were extracted from the object's current state and history. The object's current speed could be compared to the object's average speed over the last 10 frames, and the object's current shape could be compared to the object's average shape over the last 10 frames. Each object's feature probabilities were calculated, and if the object had a relationship to another object, that object's features were also included as special "relationship" features for the current object. Before extracting the features, the objects' shapes needed to be transformed to 3D space so that a person standing far away from the camera has the same height as the same person standing close to the camera. For each object, its two bottom corners (the corners closest to the ground) were translated from image plane to ground plane. By measuring the distance of these two points on the ground plane we get the objects width in ground plane space. Since the ground plane space is independent to distance from camera (one meter on the ground is one meter independent of its position relative the the camera). Once we know the ground plane width of the object we can calculate the object's 3D space height by multiplying the ground plane width with the ratio between image plane width and image plane height. This requires that the camera observes the scene from the side, but good approximations are made with cameras looking down at the scene at a angle up to 45 degrees. We now have both the image plane width and height, and the approximated 3D space width and height. We can also find the ground plane speed by looking at difference in ground plane positions from frame to frame. The features were calculated through the use of fuzzy logics. I will use three fuzzy logic functions to extract features from the numerical data (blobs and trajectories) and these three functions will be illustrated by the following graphs:

y = FUZZY-LOWPASS(x,min,max):



y = FUZZY-HIGHPASS(x,min,max):



y = FUZZY-BANDPASS(x,lowMin,lowMax,highMin,highMax):

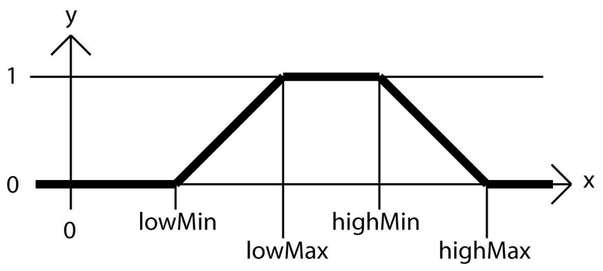


Image plane is in the next examples called IP, and ground plane is in the next examples callsd GP The following pseudo code describes how some of the features probabilities were calculated:

```
SHAPE HUMAN = FUZZY-BANDPASS( ( IP-WIDTH x 3.0 / IP-HEIGHT ) , -3.0 , 1.0 , 1.0 , 5.0 );

SHAPE WIDER THEN TALL = FUZZY-HIGHPASS( ( IP-WIDTH / IP-HEIGHT ) , 1.0 , 5.0 );

SHAPE TALLER THEN WIDE = FUZZY-HIGHPASS( ( IP-HEIGHT / IP-WIDTH ) , 1.0 , 5.0 );

SIZE HUMAN = FUZZY-BANDPASS( SQRT( IP-WIDTH$^2$ + GP-HEIGHT$^2$ ) , 0.0m , 2.0m , 2.0m , 3.0m );

SIZE SMALL = FUZZY-LOWPASS( SQRT( IP-WIDTH$^2$ + GP-HEIGHT$^2$ ) , 1.0m , 2.0m );

SIZE BIG = FUZZY-HIGHPASS( SQRT( IP-WIDTH$^2$ + GP-HEIGHT$^2$ ) , 2.0m , 3.0m );

SPEED RELATION = FUZZY-LOWPASS( GP-SPEED / AVERAGE GS-SPEED (10 FRAMES) , 0.0m/s , 2.0m/s );

SPEED FAST = FUZZY-HIGHPASS( GP-SPEED , 1.0m/s , 3.0m/s );

SPEED SLOW = FUZZY-LOWPASS( GP-SPEED , 0.0m/s , 1.0m/s );
```

The next features handles objects' interaction with scene elements like gates and benches:

```
IF THE OBJECTS IP-SHAPE OVERLAPS A GATE SCENE ELEMENT:
    FEATURE AT GATE = 1.0
ELSE:
    FEATURE AT GATE = 0.0


IF THE OBJECTS IP-SHAPE OVERLAPS A BENCH SCENE ELEMENT:
    FEATURE AT BENCH = 1.0
ELSE:
    FEATURE AT BENCH = 0.0


IF THE OBJECTS IP-SHAPE OVERLAPS A TRACK SCENE ELEMENT:
    FEATURE AT TRACK = 1.0
ELSE:
    FEATURE AT TRACK = 0.0
```

The next set of features handles handles the existence of an object, like arrive and exit:

```
IF THE OBJECT ENTERED THE SCENE LESS THEN ONE SECOND AGO
  FEATURE ENTER SCENE = 1.0
ELSE:
  FEATURE ENTER SCENE = 0.0
```

```
IF THE OBJECT LEFT THE SCENE LESS THEN ONE SECOND AGO
  FEATURE LEFT SCENE = 1.0
ELSE:
  FEATURE LEFT SCENE = 0.0
```

```
IF THE OBJECT PARTED FROM ANOTHER OBJECT LESS THEN ONE SECOND AGO
  FEATURE PART = 1.0
ELSE:
  FEATURE PART = 0.0
```

```
IF THE OBJECT WAS COMBINED BY TWO SUB-OBJECTS LESS THEN ONE SECOND AGO
  FEATURE JOIN = 1.0
ELSE:
  FEATURE JOIN = 0.0
```

Finally if an object has a relation to another object, the object inherits the other object's features as ”other” features:

```
”OTHER SHAPE HUMAN” FEATURE = RELATION OBJECT'S ”SHAPE HUMAN” FEATURE
”OTHER SPEED FAST” FEATURE = RELATION OBJECT'S ”SPEED FAST” FEATURE
```

If no relation to another object exist, all ”other” features' probabilities are set to 0. Each object now has a set of features with probabilities ranging from 0.0 to 1.0. These features' probabilities are now ready to be fed as input to the ANN.

5.5.5 ANN

The ANN is a feed-forward three layered ANN with one input node for each feature type, and one output node for each category type. Error-backpropagation is used to train the ANN, where errors are calculated by finding the difference between an output nodes output value, and the wanted output value acquired from the truth data. The activation function used by the ANN was a sigmoid function with output ranging from -1.0 to 1.0. Each node in the ANN was also biased by a weighted bias input link from a bias node with a constant output value at -1.0. The algorithms used to run the ANN will not be given in detail since they not are unique for this prototype.

5.5.6 CBR

The CBR run in two different modes: training and testing. While training, the STORE algorithm is used, and while testing, the QUERY algorithm is used. Both algorithm uses an object's categories as input, but only the STORE algorithm also uses truth data as input, and only the QUERY algorithm have scenarios as output.

STORE algorithm

The objects fed to the CBR were labeled with truth data, and were structured like this:

```
OBJECTS 1-N [
    OBJECT 1 [
        Identifier [Object1] (The object's internal name identifier)
        Numerical data [ ... ] (The objects blobs and trajectory)
        Features 1-N [ ... ]
        Categories 1-N [
            Category 1 [90%] (Category type 1 and its probability)
            ...
            Category N [30%]
        ]
        Truth scenarios 1-N [
            Scenario 1 [100%] (Scenario type 1 and its probability)
            ...
            Scenario N [0%]
        ]
    ]
    OBJECT 2 [ ... ]
    ...
];
```

These objects were converted to cases that were simply added to the case base. The cases in the case base were structured like this:

```
CASES 1-N [
    CASE 1 [
        Categories 1-N [ ... ]
        Truth scenarios 1-N [ ... ]
    ]
    CASE 2 [
        Categories 1-N [ ... ]
        Truth scenarios 1-N [ ... ]
    ]
    CASE 3 [ ... ]
    ...
];
```

When training of the CBR was done, the case base contained one case for every object in every video frame. The case base grew big, but with our limited data-sets it stayed within reasonable size, so no methods designed to reduce the case base's size were applied. No output was generated from the STORE algorithm.

QUERY algorithm

The QUERY algorithm works differently then the STORE algorithm in several ways. First, the input object's categories' probabilities were treated as a coordinate in a N dimensional space. The number of dimensions in the space was decided by the number of categories used, and each axis (dimension) in the N dimensional space was related to one categories probability. The reason that we want to think of this as a N dimensional space is that *distance* between two coordinates in this space are measured with an N dimensional *Pythagorus* function and not by a *Manhattan* function made by the sums of the distance measured along each axis. If there was a 3 dimensional space (based on three categories), the distance between two coordinates would be measured along the straight line between the two coordinates. The distance between coordinates is important since it is used to measure how well an input object matches a case in the case base. The input objects categories' probabilities were used as a coordinate in the N dimensional case base space, and the best matching case was the one with the lowest distance to the input object's coordinate. Once the best matching case was found, we could create our final output. The categories and the identity of the input object was combined with the scenario truth data from the case to create an final result output object. The final output for all objects in the current video frame was structured like this:

```
FINAL OUTPUT [
   OBJECT RESULT 1 [
        Identifier [Object1] (The object's internal name identifier)
        Scenarios 1-N [ ... ]
   ]
   OBJECT RESULT 2 [ ... ]
   ...
];
```

5.6 Example process from video input to detected events

In this section we will go through the whole process from video input to detected events, based on how we implemented the proposed design in our prototype. By going through every single step, we hope to give the reader a complete picture of how our prototype actually works. We will start with the video input and the tracker, next we will look at the ANN, and finally we will look at the CBR. What is important to remember is that the prototype runs in two different states: training and testing. Because of this, both ANN and CBR have two different ways to execute: one for training and one for testing. All elements explained through this example are implemented and are working in our prototype experiment and this section example could be understood as an introduction to our experiments.

Note: The example is divided into three subsections, and each subsection is illustrated by a figure which should help the reader to visualize the steps explained.

5.6.1 From video input to objects with extracted features

The first step of the process is to grab a frame from the *Video input*. In our prototype, this is from a video file, but when applying the system in a real subway station, a live video feed should be used. The data set we use have a frame rate at 25 frames per second. *Current video frame* is the last video frame grabbed from the *Video input*, and consists of a colored bitmap with a resolution of 352 x 288 pixels. The current video frame is first used to update the *Average video frame*. The *Average video frame* is the average of the 200 first frames, and is supposed to be a good image of the background. This background image is used to distinguish between the scene's static background and the elements (for example humans) we want to see. The current video frame is now ready to be segmented. The *Frame segmentation* stage tries to extract blobs from the current video frame. As described under *Implementation details*, the background image is subtracted from our current video frame, and with several filters and the use of flood-fill algorithm, different areas in the frame is detected as connected pixels (white pixels in the *Segmented video frame* illustration). These areas

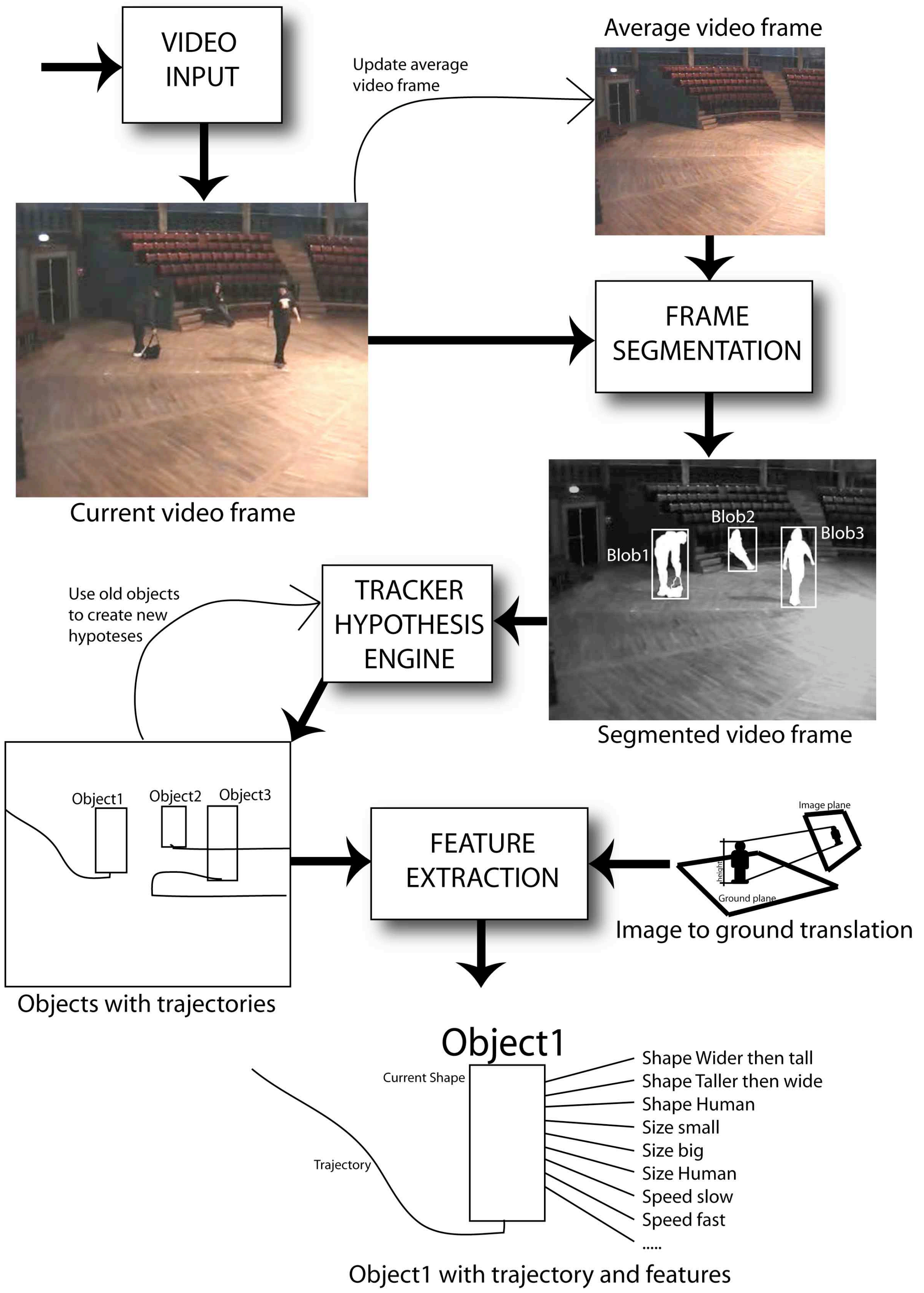


Figure 5.1: Illustration of example process described in ”From video input to objects with extracted features”

are what our frame segmentation detects as non static scene elements, and each pixel ”island” is labeled as a *blob*. A bounding box surrounding the pixel island is calculated as the final *Frame segmentation* output. These blobs are everything that is passed to the next processing stage; the *Tracker hypothesis engine*. The *Tracker hypothesis engine* starts by creating hypotheses from previously detected objects. The difference between a blob and an object will soon be evident. An object have a trajectory (a history over previous positions). From the trajectory we can find the object’s speed, and by projecting the object’s current position and shape along the direction of speed, we create a hypothesis of where we predict the object to be in the next frame. By creating one prediction for each object, we end up with one hypothesis for each object. Once we have our hypotheses in place, we match them with the new blobs from the *Segmented video frame*. By comparing shape and position, a match score is calculated, and if a hypothesis matches a blob, the blob is considered the next position and shape of the object that predicted the matched hypothesis. If no match for a blob is found, a blob is considered a new scene element (just entered) and a new object is created. If no match for an object-hypothesis is found, it is kept for some frames (in case it’s behind something), and then considered gone. In addition to the object-hypotheses, a set of group- and part- hypotheses are built. If two objects are close, a hypothesis covering both of them is created in case they overlap in the same segmentation, and only one blob (covering both persons in video frame) is detected. Similar hypotheses are created to predict that an object splits into two. Next step is *Feature extraction*. We now have a set of objects with updated trajectories. The first we then do is to translate the trajectory from *image plane* to *ground plane*. Since we now have a trajectory and shape that is independent of distance to camera, we can start to extract features and add these to the object. Each feature is calculated from the shape and trajectory of the object, and then normalized through the use of fuzzy logic. For example if the object is 2 meters tall, the size would be reported as 0.9 human size, 0.5 small size and 0.5 big size. There are over 20 features, all listed in previous chapters. Finally a set of special features are added to each object. These special features consists of three types, *existence features*, *relation features* and *scene features*. The existence features concern the object’s existence and are kept for one second. When a new object enters the scene, it is labeled with an *arrive* feature;

the object has just arrived. When an object leaves the scene, it is labeled with an *exit* feature. If two objects overlap in the segmentation stage, they join into a single ”group” object. These group objects are labeled with a *join* feature, and if an object splits in two, they are both labeled with a *part* feature. It is important to remember that the segmentation can’t tell that two overlapping objects are two and not one object, so the *join* feature helps a little, but if two persons enter the scene while overlapping in the image frame, the system will have no idea that this is two objects and not one. The system is also unable to know if a ”group” object consists of two or five objects. The *relation features* are a little more complicated. When two objects part, a relation between the two objects is stored. If a person leaves his/hers bag, a relation is kept between the two. If a person hits another person, they overlap in the image plane, and a ”group” object is created. When the hit finishes and the overlap ends, the two objects are labeled with a *part* feature, and a relation between the two are kept. The relation could be described as a ”last interacted with” relation, and one is stored for each object. For each object, the features of the ”last interacted with” object is added as relation features. If person A leaves bag B, a relation is kept, and if person A then leaves the scene, bag B will get the relation feature *exit* from person A. This way, it is possible to see if the bag is left by its owner. But what if the owner leaves his bag, and then somebody else interacts with it before the owner has left? There are many ways to solve this, for example by having multiple ”last interacted with” relations or by having a special ”originally left by” relation. With our simple one-relation system, the other person interacting with the bag could now be considered its new owner, which would be true if person A gives bag B to person C. The last set of *scene features* describes the object’s relation to marked static scene elements. For example if a person is close to a bench, a ”close to bench” feature is given a high score, and if the person is close to the track, a ”close to track” feature is given a high score. ”close to bench” is interesting for example if a person is lying down (lying down on the floor should be reported, but lying down on a bench should not). It it important to remember that there is a finite set of features, and each object ”have” all features, but with different levels, so ”close to track” with a 0.01 level should be considered as ”not close to track” We now have a set of objects with features, and these objects are now passed to the next processing stage, the artificial neural net.

5.6.2 Artificial neural network

We now have an object with a set of features, actually we have multiple objects but let us look at one object's run through the ANN. Later the system iterates through rest of the object and does the same for these as for the first. What differs the ANN and the CBR from the initial video analysis is the two states; training and testing. The video analyzer works the same way both under training and testing, but the ANN and the CBR don't. Because of this we have to describe both the process of training the ANN and testing the ANN (The same will be done with the CBR). Our *Neural network* have one input node for each feature (remember that each object have a *level of* each feature). There is one output for each category and a hidden layer consisting of 20 nodes (20 was decided upon after testing different numbers). If we look at input node 1; ”Shape Wider then tall”, this represents the feature ”Shape Wider then tall”, and the object's *level of* feature ”Shape Wider then tall” is fed directly into the ”Shape Wider then tall ” *neural network* input node. Each feature's *level of* ranges from 0 to 1, and all of these are fed into the neural network's corresponding input nodes. The input layer of the *neural network* now holds all of the object's features and we perform a ”feed-forward” process on the *neural network*. In the same way the input nodes represent features, the output nodes represent categories, and the output value of each output node represents the *level of* the corresponding category. If the level of category PERSON is 1, the net thinks that the current feature set is a person, if the level of category PERSON is 0, the net doesn't think its a person. If *level of* is 0.5, the net is not sure. The next step depends on if we are training or testing, and we start by looking at the training state. The neural net is initialized with a set of random weights, and this random net would probably give us wrong results. The *level of* category PERSON is as random as the net's weights, and since the net isn't trained, it has no possible way to tell if the input features should be considered as a person or not. For each output node we now calculate an error based on truth data entered by the system operator (in this case me). Each object in each frame of the testing data videos are marked with a set of *Category truth data*. By calculating the difference between the net's output and the net's desired output, the error of each output node of the net is found and then propagated back through the weights of each net layer, and each weight

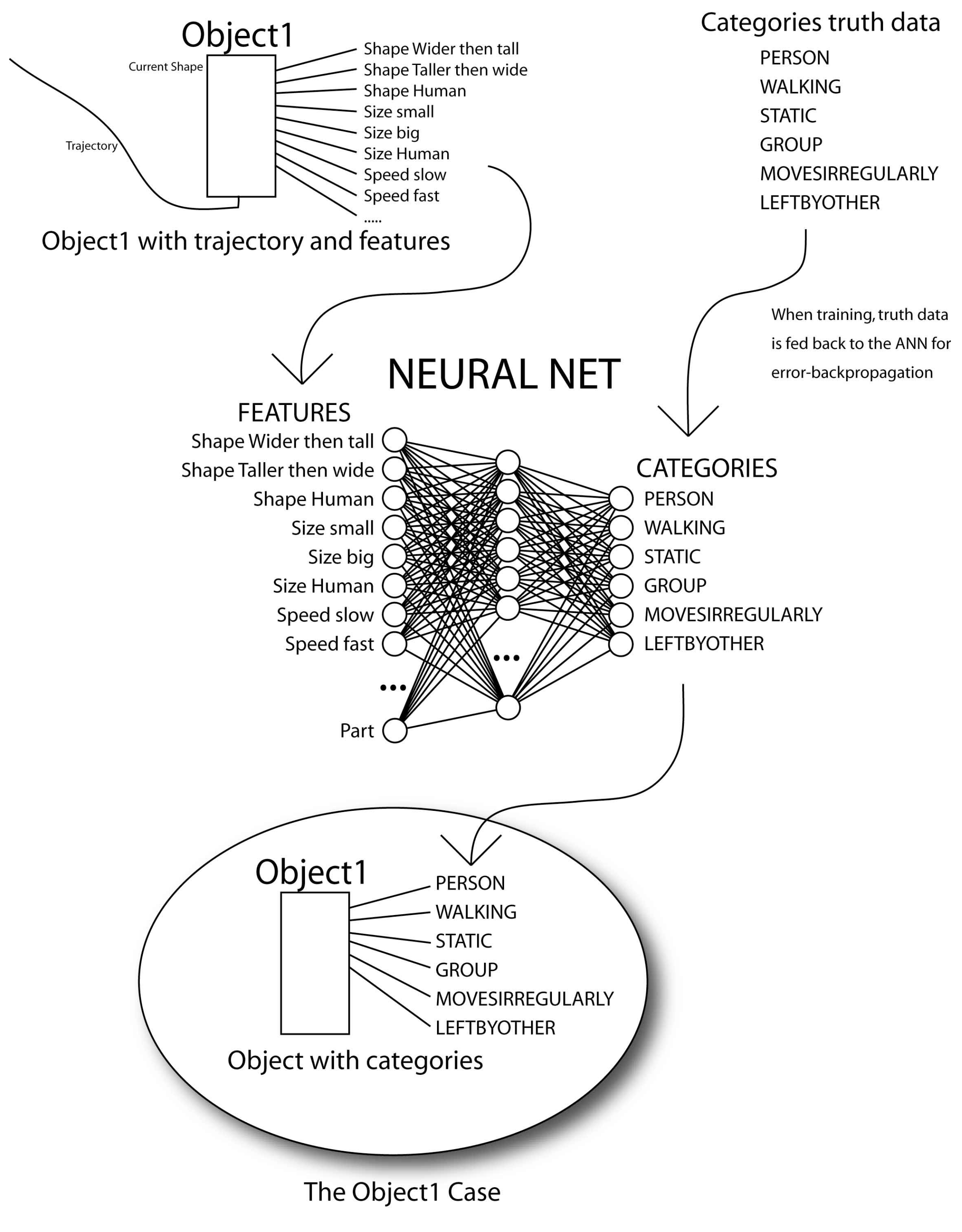


Figure 5.2: Illustration of example process described in ”Artificial neural network”

is corrected by a small ratio of the error. The process is known as ”error-backpropagation”, and by repeating this hundreds of times with hundreds of different frames, the neural net converges to a state with minimal output error. One can for example say that the weights of the edges leading from the input nodes to the PERSON output node is pushed up and down until a minimum error for PERSON is found. After training, the network’s output should be almost the same as the truth data, and generalization capabilities of the *neural network* should give us correct results on combinations of features not seen in the training stage. A short example of what features that could result in categories is now given. The PERSON category could for example be based solely on the object’s size features. In a subway setting, there are people, things that people carry and trains. The trains are much larger then the persons, and anything that a person can carry is much smaller then a person. No investigation in the neural net’s internal representation has been conducted, so we don’t know if this is the actual case, but in a subway station, size should be enough. The WALKING category and the STATIC category could be based on the speed features, and the GROUP category could be a combination of ”Shape Wider then tall” (two persons side by side is much wider then one), and the ”join” feature described earlier. MOVESIRREGULARLY could be based on difference between the object’s current speed, and the average speed over some time. For a person walking fast across the room, and a person standing still, there should be no difference, but for a person jumping around, a difference could be found. The final LEFTBYOTHER category could be a combination of being small (like a bag), and that the relation feature ”exit” is high, as described earlier. Since the net is self organizing, we don’t actually know the patterns learned by the ANN, but if there is a difference between a person and a non-person, the net should be able to find it. When not training, the object’s features are replaced with the new found categories and we now have an *Object with categories*. If the training of the ANN was successful, the new found categories should be consistent with objects in the training data, so a new object that looks (based on features) like a person in the training data is now detected to be a person. The Object with categories is what we call a case when entering the CBR stage.

5.6.3 Case based reasoning

We now have an object with a set of categories, actually we have multiple objects but let's look at one object's run through the CBR process. Later the system iterates through rest of the object and does the same for these as for the first. To the CBR, the object with a set of categories is a case, and will from now on be called a case. Similar to the ANN, the CBR has two states; training and testing. In more advanced CBR implementations, these are the same (the CBR automatically learns if the results from its testing is wrong, but we keep them separated). The *Case base* is the heart of our CBR, and contains a set of remembered cases (in our case it contains all training cases, about 1000). The *Case base* has two uses: *STORE* and *QUERY*. *STORE* is for training and *QUERY* is for testing. We will now look at training. When training, the CBR is fed with cases from the ANN. It is important to remember that we first train the ANN, and then train the CBR with the ANN in a testing state. The case fed from the ANN stage is first bundled with *Scenario truth data*. The *Scenario truth data* is entered by the system operator for each frame in each training video. It contains the truth about occurring events like *FIGHT* or *LEFT LUGGAGE*. The case is then stored in the *Case base*. Our *Case base* can be understood as a multi-dimensional space with one dimension for each category. A case's categories are therefore a position in the multi-dimensional space, and as illustrated, we hope that cases with a FIGHT scenario are grouped in one place, while LEFT LUGGAGE scenarios are grouped in another. In reality there are probably a large number of small groups mingled together in a fine-grained case space. After training the *Case base* is filled with cases bundled with *Scenario truth data*. When the CBR is in testing state, we use the *QUERY* entry to the *Case base*. While explaining the querying, we will try to draw a parallel to how humans reason with experience. The first step of *QUERY* is to find the case inside the *Case base* that's most similar to the new case fed from the ANN. In human reasoning, this step could be described as "Have we experienced a similar situation before?". Since our cases can be understood as a coordinate in a multi-dimensional space, the best match case is chosen as the closest multi-dimensional case coordinate. The output of *QUERY* is the best match from the *Case base* and let's remember that all cases in the *Case base* are bundled with truth data. In human reasoning, this

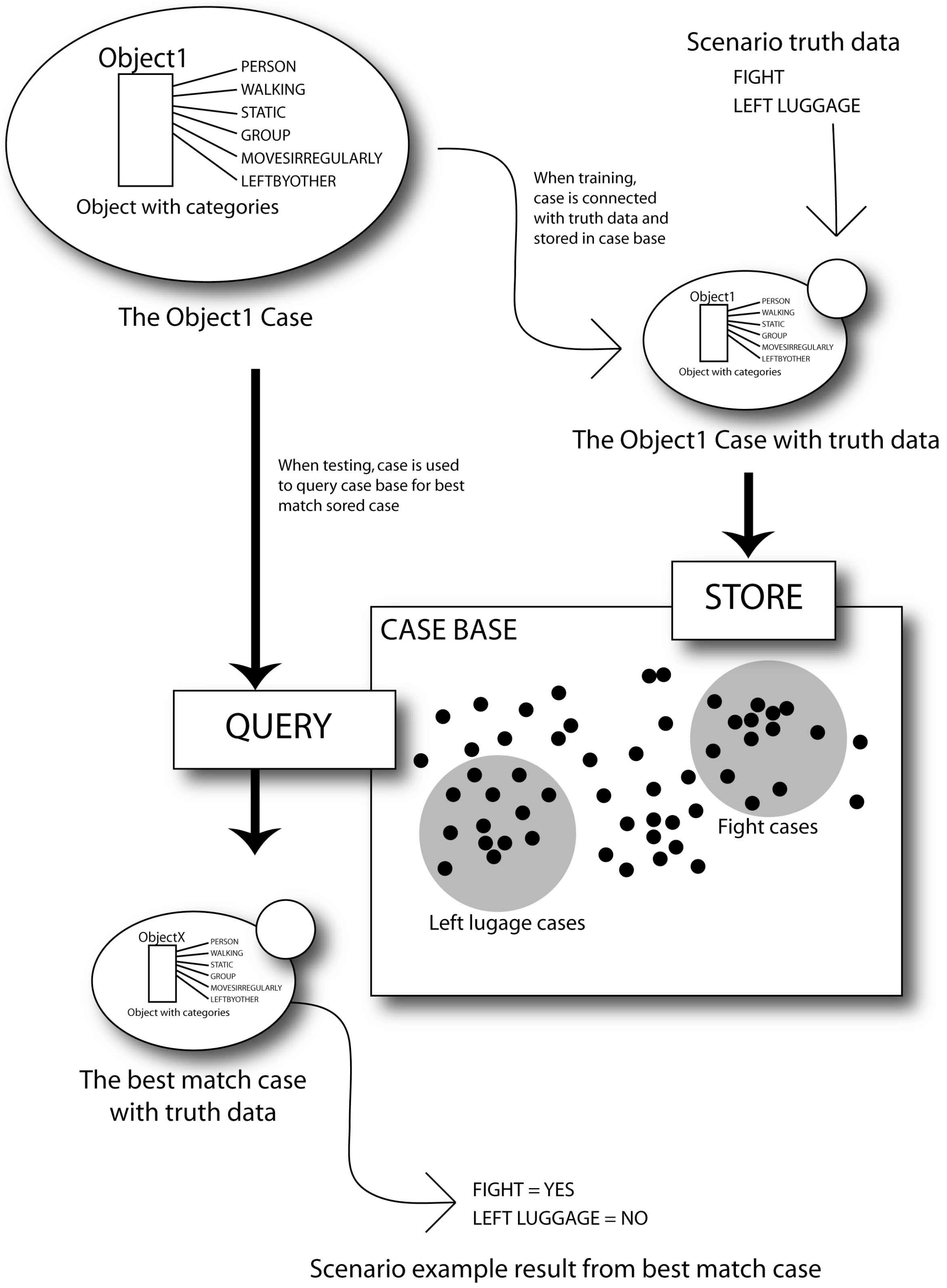

Object1
PERSON
WALKING
STATIC
GROUP
MOVESIRREGULARLY
LEFTBYOTHER
Object with categories
The Object1 Case
Scenario truth data
FIGHT
LEFT LUGGAGE
When training, case is connected with truth data and stored in case base
The Object1 Case with truth data
When testing, case is used to query case base for best match sored case
STORE
CASE BASE
QUERY
Fight cases
Left lugage cases
ObjectX
The best match case with truth data
FIGHT = YES
LEFT LUGGAGE = NO
Scenario example result from best match case

Figure 5.3: Illustration of example process described in "Case based reasoning"

step could be described as ”The most similar experience we have is this”. The next step in human reasoning would then be: ”What was right then?”. It is important to note that human reasoning is much more complicated then this, and the experience would be processes and thought about a lot, one can also imagine that a human would recall both experiences which went right, and experiences which went wrong, and then compare these to figure out what the best action was. New actions can also be tested. In our CBR, the reasoning is done, and the bundled truth data is given as final output. If the new case was similar to a trained case and that trained case was labeled with FIGHT truth data, the CBR will also think that the new case is a FIGHT. If the system is wrong, it would be a good idea to store the new case with FIGHT marked as false, but this is not done in our prototype. By stepping through the whole Video Analysis - ANN - CBR process we hope the reader have gained a better understanding for how the prototype works.

Chapter 6

Results

6.1 Experiments

To test our thesis we will conduct a series of experiments. Each experiment will be formed to confirm and shed light on our system's performance on each of the requirements in our *Cognitive behavior measurement matrix for automated video surveillance* (Table 6.1). Some requirements will be met with a qualitative answer, more then a quantitative answer. The experiments conducted were limited by our data-set and also by the scope of this Master's thesis, however: the experimental runs of our prototype consisted of 100 hours of pure processing time, and we think that, within the given limitations, we covered as much as we possibly could. Let us review the *Cognitive behavior measurement matrix for automated video surveillance*:

To explore our prototype's performance regarding the ten cognition requirements that we have proposed, we are going to conduct eight types of experiments (Table 6.2). Each experiment will handle one or two of the requirements, and some experiments will contain results obtained through test runs of the prototype.

Cognitive behavior measurement matrix for automated video surveillance	
Nr.	**Description**
1	Detect specified events from video-feeds
2	Alert guards when events are detected
3	Learn new types of events, and improve detection of events over time
4	Adapt known events to new conditions and adjust performance to improve levels of false and missed events
5	React to feedback from guards and supervisors concerning policies and results
6	Work robustly in a real-world environment with noisy and missing data
7	Run in real-time, and detect events with no or little processing delay
8	Distinguish between different types of events, and act properly to the event detected
9	Deliver a detailed explanation of what is detected to the guards
10	Aid the guards in handling the alerted event

Table 6.1: Cognitive behavior measurement matrix for automated video surveillance

Experiments	
Description	**Req nr:**
Detecting single type of event	1 & 2
Learning an event	3
Adaption and relearning	4 & 5
Performance in noise and missing data	6
Performance in time	7
Detecting multiple types of events	8
Explanation of detected events	9
Aiding guards	10

Table 6.2: List of experiments and which cognitive measurement matrix requirements they cover.

6.1.1 Detecting single type of event

Requirement 1 and 2 cover the automated video surveillance system's main task: to detect events. We have measured our prototype's performance on a data-set containing 6 videos. The videos have been used in different combinations of training-data and testing-data, and the videos contain one type of scenario that we want to detect: the *FIGHT* scenario.

Let us review the data-set table (Table 6.3). The *video name* holds the name of the video, and will be used as the video's reference name. The *frames* column holds the number of video frames in that video. The *event type* column tells us what kind of event we can find in the video, and the *e.*

Data-set			
Video name	**Frames**	**Event type**	**E. Frames**
F1	755	FIGHT	395
F2	513	FIGHT	193
F3	480	FIGHT	215
F4	659	FIGHT	536
C1	4675	N/A	-
C2	3054	N/A	-

Table 6.3: List of videos in data-set used for experiment: ”Detecting single type of evet”

frames column contains the number of frames that contains the event. For example, video F1 is 755 frames long, and 395 of those frames covers the interval of a FIGHT event.

The ANN was trained with 2000 epochs for each training set. The ANN had three layers, with features as input, 20 nodes in the hidden layer, and categories as output. The category output from the ANN was used as input for the CBR. The categories used can be found in (Table 6.4).

Categories	
PERSON	Is the object a person?
WALKING	Is the object walking?
STATIC	Is the object a static object?
GROUP	Is the object a group of objects?
MOVES IRREGULARLY	Is the object moving irregularly?
LEFTBYOTHER	Is the object left by another object?

Table 6.4: List of categories learned by the ANN in experiment: ”Detecting single type of evet”

Different combinations of videos were used as training-data and testing data, and the following tables shows the system’s performance on detecting the FIGHT events (Table 6.5, Table 6.6, Table 6.7, Table 6.8, Table 6.9, Table 6.10). Each table shows results of testing one video from the data-set. The *training-data* column shows which videos were used to train the system; the *event* column shows the percentage of frames where the fight event occurs, and where the fight event was detected; the *empty* column shows the percentage of frames where no fight event occurs, and that was classified correctly, and the *total* column shows the percentage of all frames in the testing video that was classified correctly.

So what do these results mean? Let's first look at the results from C1 and C2. The C1 and C2 videos are long videos (4675 & 3054 frames) with two or three persons are wandering around and waiting for a train. No fight events occur, and no false detections of fight events are done by the system. This means that all tests on both videos gives 100% correct results: no false detections. Tests of video C1 and C2 tells us that the system regularly doesn't give false alarms, but the videos containing fight events show us that some false events are detected. These false events happen right before and right after the fight, and consist mostly of single frame detections. The frames containing fight events are not always detected correctly, but while a fight is happening, the detected frames are often connected over longer intervals. If we plot the detection errors of the F2 test with F1 as training data, we see where the false detection occur:

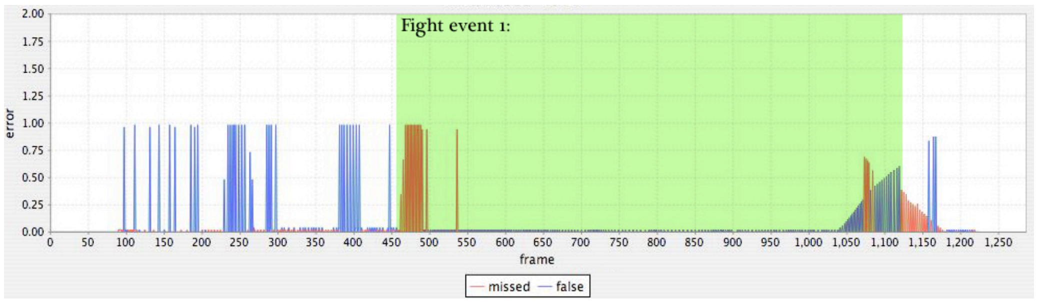


Figure 6.1: Errors in scenario classification while testing F2

In figure 6.1, the X-axis shows the frames of the video stretched out over time, and the Y-axis shows the error in detection. The green area shows where the fight is occurring *(frame 460 to 1120)*. Initially the false detected frames are single spikes *(until frame 220)*. As we get closer to when the fight begins, we see that the spikes occurs more regularly *(frame 220 to 460)*. In the beginning of the fight, the fast fluctuation between detection and non-detection continues *(frame 460 to 500)*, but then the detection stabilizes, and the fight is fully detected over a long interval *(frame 500 to 1050)*, almost until it is over. The system struggles with deciding when the fight is over (frame 1050 to 1170), but after some fluctuation, it stops detecting false events *(frame 1170 to the end)*.

Results for testing video F2			
Training-data	**Event**	**Empty**	**Total**
F1	92,75%	81,56%	85,77%
F1+F4	97,93%	72,81%	82,26%

Table 6.5: Results from testing video F2

Results for testing video F3			
Training-data	**Event**	**Empty**	**Total**
F1	63,72%	56,60%	59,79%
F1+F2	96,28%	56,98%	74,58%
F1+F4	100,00%	64,53%	80,42%
F1+F2+F4	75,35%	48,68%	60,63%
C1+F1+F2	10,23%	90,19%	54,38%

Table 6.6: Results from testing video F3

Results for testing video F4			
Training-data	**Event**	**Empty**	**Total**
F1	45,52%	57,72%	47,80%
F1+F2	63,25%	72,36%	64,95%
F1+F2+F3	58,58%	85,37%	63,58%
C1+F1+F2	10,07%	95,12%	25,95%

Table 6.7: Results from testing video F4

Results for testing video C1			
Training-data	**Event**	**Empty**	**Total**
F1	N/A	100,00%	100,00%
F1+F2	N/A	100,00%	100,00%
F1+F4	N/A	100,00%	100,00%
F1+F2+F3	N/A	100,00%	100,00%
F1+F2+F4	N/A	100,00%	100,00%

Table 6.8: Results from testing video C1

Results for testing video C2			
Training-data	**Event**	**Empty**	**Total**
F1	N/A	100,00%	100,00%
F1+F2	N/A	100,00%	100,00%
F1+F4	N/A	100,00%	100,00%
F1+F2+F3	N/A	100,00%	100,00%
F1+F2+F4	N/A	100,00%	100,00%
C1+F1+F2	N/A	100,00%	100,00%

Table 6.9: Results from testing video C2

Summary table of all detection percentage (Total) results:					
Training-data	**F2**	**F3**	**F4**	**C1**	**C2**
F1	85,77%	59,79%	47,80%	100,00%	100,00%
F1+F2	-	74,58%	64,95%	100,00%	100,00%
F1+F4	82,26%	80,42%	-	100,00%	100,00%
F1+F2+F3	-	-	63,58%	100,00%	100,00%
F1+F2+F4	-	60,63%	-	100,00%	100,00%
C1+F1+F2	-	54,38%	25,95%	-	100,00%

Table 6.10: All results from testing all videos

This test run got a total of 85,77% frames correctly classified. Let us look at a test run with a lower detection score. The next figure (Figure 6.2) shows F4 tested with F1, F2 and F3 as training data, and classified 63,58% of the frames correctly:

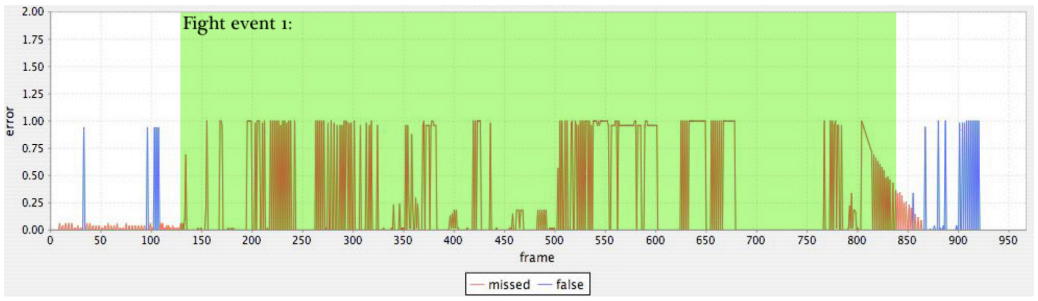


Figure 6.2: Errors in scenario classification while testing F4

In this run, we find the same pattern. Initially some spikes of falsely detected events, then fluctuation before the event starts, but this time the system doesn't go into a stable mode where the fight is detected continuously for the rest of the fight. Instead of being detected continuously, the fight event is full of holes (frames where no fight is detected). Some areas are pretty stable, but only 58,58% of the frames within the fight event is classified correctly.

The system gets its lowest scores after being trained with C1, F1 and F2. When testing with F4, only 25,95% of the frames are detected correctly (10,07% of the fight event is classified correctly, but 95,12% of the frames outside the fight event are classified correctly). If we look at the training data and the frame count, we see that the C1 video has about 5 times as many frames as the F1 and F2 videos. All of the frames of C1 are without fighting, and some of the F1 and F2 frames are also without fighting. This unbalance between fight frames and fight free frames could result in a system that most of the time classifies the testing data frames as not containing fighting (since this is right most of the time). The effect of the unbalanced training data should be investigated further, but we clearly see that more balanced training data (F1, F2 and F3 contains about a 50%/50% balance between fighting and non-fighting) gives better results. Using more epochs when training the net could also be investigated.

The results show us that the system *can* detect the FIGHT event, and the error free long videos without fighting tells us that it is also capable of not detecting false events. Finally we are going to compare our system's result with two modified versions of the system, one only using ANN, and one only using CBR (Table 6.11). The standalone ANN runs under the exact same conditions as the ANN in the ANN/CBR hybrid, and the standalone CBR runs under the exact same conditions as the CBR in the ANN/CBR hybrid. The only difference is that the standalone ANN has scenarios as output, and the standalone CBR has features as input.

The first thing we notice about this table is the ANN's inability to keep away from false events in the long eventless videos. When testing C2 after training with F1, only 22% is classified correctly, this means that 78% of the time, the ANN thinks there is a fight. Next we can also see that the ANN and CBR outperforms the ANN/CBR hybrid in row 2 and 3, but at row 4, both the ANN and the CBR falls in percentage of correctly classified frames, while the ANN/CBR hybrid classifies almost the double amount of frames correctly. These differences could be investigated even further, but we limit ourself to these numbers in this master's thesis. Later we will test the three versions of the system with noise in the training and testing data, so we might get some more insight in how the versions compare under those conditions.

Compared results between ANN, CBR and ANN/CBR				
Test-data	**Training-data**	**ANN**	**CBR**	**ANN/CBR**
F2	F1	84,99%	68,81%	85,77%
F3	F1	72,29%	60,83%	59,79%
F4	F1	49,62%	52,50%	47,80%
F4	F1+F2+F3	32,93%	36,87%	63,58%
C1	F1	52,45%	100,00%	100,00%
C1	F1+F2+F3	78,93%	100,00%	100,00%
C2	F1	22,04%	100,00%	100,00%
C2	F1+F2+F3	51,08%	100,00%	100,00%

Table 6.11: Compared results between ANN, CBR and ANN/CBR

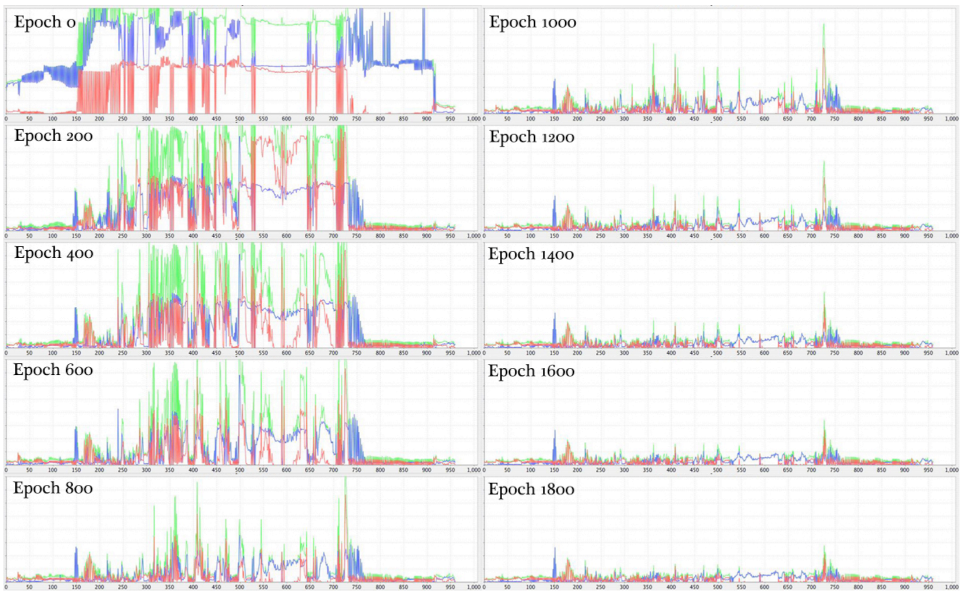


Figure 6.3: Error output from ANN after 0, 200, 400, 600, 800, 1000, 1200, 1400, 1600 and 1800 epochs.

6.1.2 Learning an event

Requirement 3 in the *cognitive measurement matrix* is about learning, and the next experiment is about learning to detect an event. We will now look into the ANN and the CBR while they are training. We hope to show how continuous exposure to the training data improves performance. First we look at the ANN while training to see how it approximates the training data. We train the ANN with 2000 epochs, and we look at the ANN's error output for every 200 epoch (Figure 6.3). The X-axis of figure 6.3 show the F1 training video stretched out over time, and the Y-axis show the normalized sum of errors in the ANN's category output. The point of the figure is to show that these errors are reduced for every training epoch. This process is described as the training of the ANN, and the error reduction show us that the ANN learns. We can also look at how the ANN output error reduces over time (Figure 6.4).

Next we are going to look inside the trained CBR's case base (Figure 6.5). We recall that the stored cases consists of category probabilities, and that they are labeled with scenarios from truth data. X-axis and Y-axis of figure 6.5 show different category probabilities, where black cases

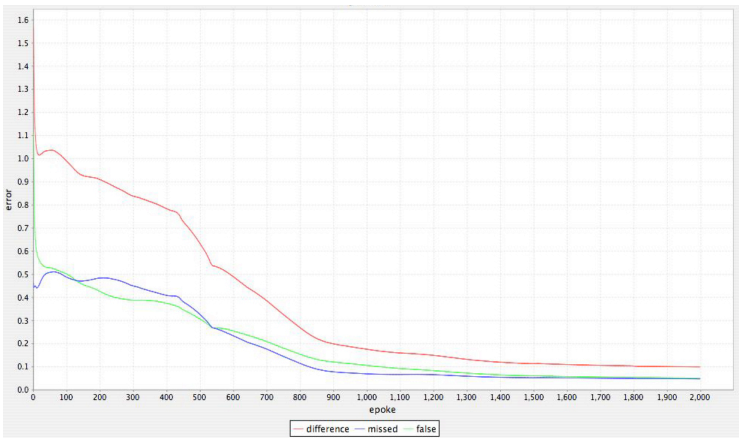


Figure 6.4: Decreasing ANN error output from epoch 0 to epoch 2000

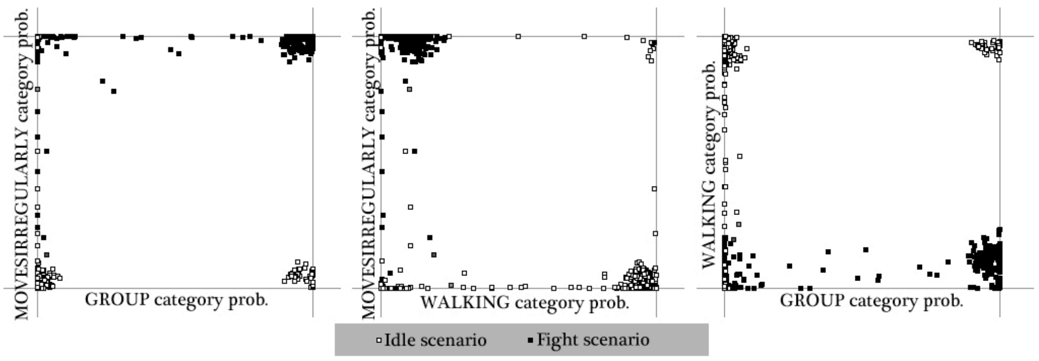


Figure 6.5: CBR cases, plotted on category axes

represent cases labeled as a FIGHT scenario, and white cases represents cases labeled as not being a FIGHT scenario. The case base we are going to look at is the case base after training with the F1 video. Some categories' probabilities are all 0% or all 100%, since no variations of the categories appear in the F1 video, for example: STATIC (no static objects), LEFT BEHIND (no objects are left behind), and PERSON (all objects in F1 are persons). This leaves us with 3 varying categories; GROUP, WALKING and MOVES IRREGULARLY.

From these cases we can see that after training with F1, the case base thinks that both WALKING axis and MOVES IRREGULARLY axis divides the case base in FIGHT and IDLE (no scenario detected) scenarios, while the scenarios are evenly distributed over the GROUP axis. This means that if an object is walking and also moves irregularly, there is a good chance that the matched case from the CBR will be labeled as a fight scenario. There are of course objects that are walking without fighting, which should be found in the CBR after training with a larger set of training videos than F1 only. It is also important to remember that the CBR learn from the ANN's output, so if for example all the ANN output categories' probabilities were inverted (or reordered), the CBR would still find similar patterns, and detect the scenarios correctly; the CBR doesn't know what these categories are, and doesn't need to, as long as there is a pattern it can find in the category cases.

6.1.3 Adaption and relearning

Requirements 4 and 5 in the *cognitive measurement matrix* is about adapting over time and relearning. In this experiment we are not going to make the system adapt over time, or relearn any scenarios, but we will explain how the process could be done. What is interesting here is the difference between the ANN and the CBR. Retraining the ANN requires that the ANN is exposed to training data thousands of times. This process can not be done in real-time, and requires the ANN to revisit all the training data, and we have to start from scratch. The CBR is different. The *revise* and *retain* steps of the CBR cycle are both supposed to improve the CBR's performance over time (Aamodt & Plaza, 1994). Each time a matched case gives scenario results, the results should be evaluated. For example if a single case in the case base over and over gives wrong results, it should be removed or its scenario data should be changed. This would be done by the *revise* step in the CBR cycle. In our prototype the CBR is simplified, so the case-base is never updated after training, but in a more complete CBR implementation, the CBR would continue to revise its cases. The *retain* step in the CBR cycle decides what cases to store (retain) in its case base. This process could run in real-time, influenced by for example the guards' feed back.

The difference between the CBR's and the ANN's ability to retrain itself in real-time since most of the time we only need to retrain the CBR (which we can do in real-time). If we revisit the category output from the ANN, we see that these categories are somehow independent of the finally detected scenario. The category PERSON should give the same results even if the object detected as a PERSON is fighting or not. The same is true for the categories GROUP, STATIC, WALKING, MOVES IRREGULARLY and LEFT BEHIND. If the ANN detects any of these categories falsely it has to be retrained, but if not, it doesn't need to. In a full implementation of the system, a larger set of categories would probably be needed to detect other scenarios then FIGHT and LEFT LUGGAGE, but once the categories are decided and trained correctly, the ANN could be kept as it is, while the CBR is updated in real-time.

If we compare this process with a human recalling experiences, the process could be described as: "Last time I experienced this situation, I thought it was a fight, but I was wrong, so now don't think it is a fight.". The CBR would in this example be the recalling and comparing of experiences, while the output categories from the ANN would be the system's perception of the experience. What is interesting in this example is that the experience (the ANN's output categories) stays the same, while the reasoning (the CBR matchmaking) and conclusion (the CBR case labeling) have changed. However, if the old error in classifying the case as a fight was because the person thought that the passing train was a fight, the perception of the situation had to be retrained (related to our system, this would be the ANN).

6.1.4 Performance in noise and missing data

Requirement 6 in the *cognitive measurement matrix* is about handling noisy real world inputs. In this experiment we will test the system while adding noise to the input data, both while training and testing. First 20% noise, and then 50% noise will be added to the input features, and all three versions of the system: the pure ANN, the pure CBR and the ANN/CBR hybrid, will be tested under these noisy conditions (Table 6.12, Table 6.13 and Table 6.14). 20% noise means that each input feature's probability is mixed 20% with a random value between 0 and 1, with this formula: **Feature = Feature*(100%-Noise) + Random(0 to 1)*Noise**. This means that with 50% noise , the feature is still in the right area, and not completely noised away. The system seems to operate properly also under noisy conditions. The exact effect of noise should be studied even further, but

Compared results between ANN, CBR and ANN/CBR under 0% noisy conditions					
Test-data	**Training-data**	**Noise**	**ANN**	**CBR**	**ANN/CBR**
F2	F1	0%	84,99%	68,81%	85,77%
F3	F1	0%	72,29%	60,83%	59,79%
F4	F1	0%	49,62%	52,50%	47,80%
C1	F1	0%	52,45%	100,00%	100,00%
C2	F1	0%	22,04%	100,00%	100,00%

Table 6.12: Compared results between ANN, CBR and ANN/CBR with 0% noise

Compared results between ANN, CBR and ANN/CBR under 20% noisy conditions					
Test-data	**Training-data**	**Noise**	**ANN**	**CBR**	**ANN/CBR**
F2	F1	20%	78,17%	69,79%	78,56%
F3	F1	20%	70,42%	60,63%	55,63%
F4	F1	20%	49,62%	50,53%	74,96%
C1	F1	20%	71,55%	100,00%	100,00%
C2	F1	20%	26,13%	100,00%	100,00%

Table 6.13: Compared results between ANN, CBR and ANN/CBR with 20% noise

Compared results between ANN,CBR and ANN/CBR under 50% noisy conditions					
Test-data	**Training-data**	**Noise**	**ANN**	**CBR**	**ANN/CBR**
F2	F1	50%	62,96%	66,08%	74,85%
F3	F1	50%	55,21%	62,08%	59,38%
F4	F1	50%	18,82%	59,03%	53,26%
C1	F1	50%	99,85%	100,00%	100,00%
C2	F1	50%	99,51%	100,00%	100,00%

Table 6.14: Compared results between ANN, CBR and ANN/CBR with 50% noise

no mediate collapse in the performance of any of the three methods are observed.

6.1.5 Performance in time

Requirement 7 in the *cognitive measurement matrix* is about the system's ability to run in real-time. Let us look at the different parts of the system, and their performance regarding speed. For our approach to be used in a realized automated video surveillance system, the system needs to run in real-time. The events and scenarios need to be detected as they happen, not later. In 3 years of operation, the system need to process 3 years of video input. A system running at half the speed of the incoming data would need 6 years to process 3 years of data, which would mean that the events happening after 3 years of video input would be detected after 6 years of processing. So what about scale? Isn't processing power scalable? Can't two processors process twice the amount of data a single processor can process? To some degree the processing capabilities of a system is scalable, and with big enough processing clusters, a super computer with hundreds of processors would probably process an unlimited amount of data, at least in the context of video analysis. Besides improving processing power, algorithms can also be optimized to increase speed, and special hardware can be used to process the most process heavy parts of the data (a graphical processing unit (GPU) could perform the video analysis before the video data reaches the CPU).

These claims of scalable processing power would make our real-time argument useless, since processing power always could be increased. So let us not measure the system's performance without limitations in processing power, but instead we use strict limitations in processing power. Let us limit ourself to a system where each video feed is processed by one CPU, and where one CPU processed and combines the data after being analyzed. The measured prototype run on a 1.83 Giga-Hertz CPU with two cores (Let us say that these two cores represent two CPUs). No optimizations are done regarding algorithms, and the prototype run in a virtual java environment (Java divides the processing power in half, compared to native code). The prototype analyzes one video feed from a single camera, so our idea about one CPU per camera and one CPU for central processing fits the hardware the prototype is running on.

Let us first look at the video analysis performance. The prototype analyzes 5 frames per second of high resolution video (352 x 288 pixels). With a lower resolution the speed increases up to 20 frames per second. The video data sets used to test the system have a frame rate at 25 frames per second, so 5 frames per second means 20% of real-time speed. When it comes to handling and filtering pixels, optimization of algorithms is very important, since each operation is repeated for each pixel (352 x 288 x 25 = 2.5 million pixels per second). If we look at other video analysis systems (for example filters running on regular home computers) they are highly optimized and can handle much higher frame rates then 5 per second.

Acquiring the video frames from the video file takes about 45% of the video analysis processing time because of reading from disk, and decompressing the video file. By simply feeding the video frames directly from a camera, time used to acquire a video frame could probably be divided in three, and our system is already running at 7 frames per second. My point is that video analysis and optimization of video analysis processing algorithms are solved, and if this was the focus of our thesis, a frame rate above 25 frames per second would be obtained. The objects and features found by the video analysis step flows in about 60 kilobytes per second, so a single CPU could process the video input stream, and send it to the central processing CPU.

Let us look at the central processing performance. The central processing CPU would handle everything except the video analysis and consists mainly of the ANN and the CBR. In our prototype these stages can handle about 100 to 200 frames per second (4 to 8 times the video frame rate). This means that our central CPU could handle analysis result data from 4 to 8 cameras simultaneously, if we had two central CPUs, this doubles.

Our measurements indicates that our un-optimized, not running native code prototype can *analyze* one video stream per CPU, and *reason* on about 4 to 8 video streams on a single CPU. A subway station with 10 cameras would then need 10 + 2 CPUs to run at real-time speed. These processing requirements are not unreasonable, and by using the latest and fastest optimized algorithms, the needed processing power could probably be divided by 3.

Data-set			
Video name	**Frames**	**Event type**	**E. Frames**
F1	755	FIGHT	395
F2	513	FIGHT	193
F3	480	FIGHT	215
C1	4675	N/A	-
C2	3054	N/A	-
B1	2533	LEFT LUGGAGE	276
B2	1100	LEFT LUGGAGE	232

Table 6.15: List of videos in data-set used for experiment: ”Detecting multiple types of evets”

6.1.6 Detecting multiple types of events

Requirement 8 in the *cognitive measurement matrix* is about the system's ability to distinguish between different types of events, not to only detect them. The system needs to distinguish between different types of detected events. We will test this by training the system both with FIGHT scenarios and LEFT LUGGAGE scenarios. The categories, features and ANN conditions are the same as in previous experiments.

Let us review table 6.15: The *video name* is the name of the video, and will be used as the video's reference name. The *frames* column holds the number of video frames in that video. The *event type* column tells us what kind of event we can find in the video, and the *e. frames* column contains the number of frames that contains the event. For example, video F1 is 755 frames long, and 395 of those frames cover the interval of a FIGHT event. The LEFT LUGGAGE event should be fired when a bag is left by its owner, and the owner leaves the scene (Just leaving the luggage is not enough to fire the alarm). To test that the system can distinguish between different types of events, we will now train it with two types of events, and then test it with two types of events. The training data consists of F1 and B1, and the testing data consists of F2,F3,C1,C2 and B2.

If we look at these results (Table 6.16) we see that the system can detect both fights and left luggage. The longer event free videos (C1,C2) have no false detections, the FIGHT scenarios are detected as fights (F2, F3, F4), and the LEFT LUGGAGE scenario is detected as left luggage (B2).

Summary table of detection percentage (Total) results:						
Training-data	**F2**	**F3**	**F4**	**C1**	**C2**	**B2**
F1+B1	86,35%	60,00%	27,62%	100,00%	100,00%	81,18%

Table 6.16: Results from detecting multiple types of events

If we look at the 27% result of F4, it is important to mention that in this run, most of the event free video frames were detected correctly, but only about 10% of the fight was detected as a fight. This means that a fight would have been reported (but the reported frame interval would be wrong). Since no false events were detected in the longer event free videos (C1,C2), the system seem run correctly when nothing is happening, but maybe only a part of the fight is detected as a fight when the FIGHT scenario finally occurs. Since the system is supposed to alert the guards only once per fight (as the surveillance operator would do in the real world), 10% of the fight is enough to alert the guards, actually 1 single frame would be enough if there were no false alarms.

6.1.7 Explanation of detected events

Requirement 9 in the *cognitive measurement matrix* is about the system's ability to explain what it has detected. When the system alerts the guards that an unwanted event is happening, it would be beneficial if the system could explain why the event was detected. The prototype has no functionality that enables it to do such a thing, but we are still claiming that this is possible to some degree. Let us look at the categories and scenarios detected in our previous experiments. An example of a case detected as a fight could be something like table 6.17.

FIGHT case's category probabilities	
PERSON	95%
WALKING	45%
STATIC	5%
GROUP	89%
MOVES IRREGULARLY	72%
LEFTBYOTHER	3%

Table 6.17: Example set of category probabilities

If we look at the categories with high probabilities (Table 6.17) we see that the most dominant ones are PERSON, GROUP and MOVES IRREGULARLY. As we saw in the ”Learning an event” section, the category MOVES IRREGULARLY was the category that divided between between fight cases and fight free cases. By analyzing the case base one could find the category probabilities that was most important for this case to recognized as a fight scenario. After finding the most important categories, they could be combined into a sentence that in this example would look something like this: *”A **group** of **people**, **moves irregularly**”* .

6.1.8 Aiding guards

Requirement 10 in the *cognitive measurement matrix* is about the system's ability to aid the guards to handle an event, after the event has happened. No attempts are done in this master's thesis to develop abilities for the system to aid the guards, but the system is already tracking objects, so tracking the person who left his luggage, or who just finished a fight should be no problem. This tracking of people could aid the guards in catching perpetrators.

Chapter 7

Conclusion

This chapter start with a summary of the thesis. The next part is a discussion regarding the findings of the thesis, and finally we make a proposal for future work.

7.1 Summary

Cognitive vision is an interesting field of research that tries to create vision systems with cognitive abilities. Automated video surveillance is increasingly needed to watch our public transport systems, either as a totally automated system or as an operator assistant. In this thesis a design for a cognitive automated video surveillance system is proposed. A hybrid ANN/CBR behavior analysis system is proposed as a cognitive extension of existing video tracking system, and a prototype is developed. By investigating the current state of cognitive vision and automated video surveillance and by looking at existing hybrid approaches to combine them, it is argued that this approach is an area of research where few designs are implemented and tested. Several frameworks for both ANN/CBR hybrids are proposed in literature, and so called emergent cognitive system frameworks are also presented, but few implementations have been done. As far as our literature review has spanned, no automated video surveillance system is attempted with the use of an ANN/CBR hybrid. Our implemented video analysis system builds directly on existing research, and shows that video tracking is an improving field and that the technologies are ready for real applications. We managed to build a sufficient video analyzer that was able to track people in a video stream. The video analyzer found objects, and gave the objects a set of features. The features were then mapped to categories with the help of an artificial neural network. The ANN worked as a set of soft rules trained to find category patterns in the stream of features. The categories was used as input data to our simplified case based reasoning system. Through example based learning, the CBR was able to recall scenarios from its training session, and match these with the current set of categories. By looking back at previously experienced scenarios, the CBR was able to decide which of these scenarios the current scenario was most similar to, and then propose if the current scenario contained an event we wanted to detect or not. By investigating the requirements of what a cognitive vision system should be capable of, we found a design that fulfilled most of these requirements, and the experiments conducted with the prototype was also chosen to explore the cognitive abilities of our prototype and design.

7.2 Discussion

This section will discuss different aspects of this thesis. The experimental results of our prototype is discussed. Cognition and our success of applying it to the task of video surveillance is discussed. Finally we end with a discussion about the relevance of the thesis regarding the field of artificial intelligence.

7.2.1 Experimental results

Through our experiments we show that our system is capable of learning and detecting events. Our low level ANN analyzer finds patterns in the form of categories in the stream of features provided by the video analyzer system, and our high level CBR implementation reasons over the detected categories through example based reasoning. A simple way for the CBR to explain its findings is proposed, and by implementing the full system prototype, we show that our design is capable of what we proposed.

Let us review our experiments. In experiment 7.1.1, *Detecting single type of event*, we showed that our system was capable of detecting a fight in a video stream. The detection of the fight ranged from 56% of the fight detected, to 100% of the fight detected. The most promising part of this experiment was that not a single frame, from the longer videos with no fight event, was falsely detected as a fight. These videos were much longer then videos containing fight events, so if we looked at the total of all frames in all testing video, we would find a very high percentage of correctly classified frames. It is also important to remember that *all* fight events were detected, event if only a part of the frames containing the fight event was classified correctly. No misses. Trying out the system on a much larger data-set would reveal the approach's actual performance, but the results obtained are promising. By comparing our ANN/CBR hybrid with a pure ANN version, and a pure CBR version, we see that the three approaches gives similar results, but some positive advantages of our ANN/CBR hybrid are found. The pure ANN version has trouble with the longer event free videos, and in one of the compared tests, the ANN/CBR outperforms the other version with almost the double percentage of correctly classified frames. A larger data-set should also be used to test these differences even more. In experiment 7.1.2, *Learning an event*, we show how the ANN and the CBR learns the events, and as tested in experiment 7.1.1, the system learns the events to a degree where it is able to detect those events. Since both the ANN and the CBR are self-organized, a larger study of how they organize internally would give us a better understanding of how the internal self organizing actually works when detecting events. What is detected and what is represented would be interesting to investigate, what is *a fight* to the system? In experiment 7.1.3, *Adaption and relearning*, the design is discussed regarding adaption and relearning, and in experiment 7.1.4, *Performance in noise and missing data*, we measure the ANN/CBR hybrid, the pure ANN version and the pure CBR version in an environment where noise is added to the input data. These results are somehow varying, so yet again, a larger data-set should be used to find patterns in how the different versions behaves in noise. What is interesting is that even at 50% added noise, none of the versions fail totally or collapse. The versions still detect events, but with a lower accuracy. If this robustness is a product of the use of an underlying tracking system, or an other part of the design is hard to say. In experiment 7.1.5, *Performance in time*, the processing power required to run the system is discussed, and based on the fact that the prototype already run almost real-time on my experiment computer, we argue that the approach isn't limited by unreasonable large processing loads. In experiment 7.1.6, *Detecting multiple types of events*, we show that the system can distinguish between a fight event and a left luggage event, and still stay error free in the longer event free videos. Finally an argument is made that the system's internal representation of categories could be helpful if an attempt to deliver natural language explanations when a unwanted event is detected, and the system's capability to aid guards even further is also discussed.

7.2.2 Cognition

Our proposed system is designed to fulfill the requirements of what is thought of as a cognitive vision system. Some requirements regarding embodiment of the system, and movement is left out since these are unneeded for our automated video surveillance system, but except for those unfulfilled requirements, the overall formula of a cognitive vision system is followed. The ANN/CBR hybrid extends the underlying video analysis system with cognitive capabilities. The system can learn and reason in the real world domain of subway surveillance through video input.

7.2.3 Where is the artificial intelligence?

Through our thesis we have visited perception, reasoning and learning. We have tried to create an artificial automated video surveillance operator, requiring what can be argued to be intelligence. We have used artificial neural networks for learning and pattern recognition, and case based reasoning for learning and reasoning. In this thesis we haven't only investigated all these areas of artificial intelligence, but we have also integrated them and made them work together in a real world environment through our prototype.

7.3 Future development

The design should be tested at a much larger scale. With a larger data-set better empirical results could be achieved, and a better understanding of the system's capabilities could be acquired. By installing the system in a real subway station, much knowledge could be acquired in the system's capabilities in the real world video surveillance domain. This implementation would require a larger architecture and the project would easily grow out of the possible scope of a master's thesis.

When building the prototype many simplifications was made. Recurrent ANNs was left out, and the CBR system was simplified. By investigating more advanced ANNs and a more advanced CBR system, the design's possibilities would be better understood. The system's ability to communicate with human guards is also left out. Investigating the possibilities of integrating natural language to the system would be interesting, and by improving the CBR's capabilities to communicate its results, the system could in addition to detecting events, also crude describe events, and answer questions from guards to improve the level of aid it delivers. Finally, the system should learn from every detected event while running and not only in a pre-processed learning stage. This learning could be supervised and confirmed by communicating with the guards, what went right and what went wrong when handling the different events.

The self organizing nature of the system hides the system's internal representation of an event. By analyzing these internal representations, the way the system understands an event would be better understood. The internal representations could also tell us how we should extend our perception system to increase performance of the behavior analysis; what data do we need?

Initially our prototype was supposed to extend an existing video analysis system. By extending an existing, robust and mature video analysis system, our focus would only cover the behavior detection part of the whole system and not the low level video analysis. Our prototype analyzer was a limited version of other existing systems, and to some degree we reinvented the wheel. The problems we encountered when trying to access other existing systems could be solved in future development.

Chapter 8

Reference

Aamodt A., Plaza E., ”Case-based reasoning; Foundational issues, methodological variations, and system approaches”. AI Communications, Vol.7, No.1, pp. 39-59, March 1994.

ADVISOR, Annotated Digital Video for Intelligent Surveillance and Optimised Retrieval (IST199911287)

Aloimonos Y., ”What I have learned,”, CVGIP: Image Understanding, 74-85, July 1994.

Auer P. et al, ”A Research Roadmap of Cognitive Vision”. ECVision: The European Research Network for Cognitive Computer Vision Systems, www.ecvision.org, 2005.

Borg M. et al, ”Video Surveillance for Aircraft Activity Monitoring”. The 8th International IEEE Conference on Intelligent Transportation Systems (ITSCA05) in Vienna, Austria on the 13th to 16th os, September 2005.

Bremond F., Moenne-Loccoz N. & Thonnat M., ”Recurrent bayesuan network for the recognition of human behaviours from video”. In ICVS2003, 2003.

Brooks R. A., ”Intelligence without representation”. In Artificial Intelligence 47, 139-159, 1991.

Cangelosi A., Parisi D., ”How nouns and verbs differentially affect the behavior of artificial organisms”, In J.D. Moore & K. Stenning (Eds.), Proceedings of the 23rd Annual Conference of the Cognitive Science Society, London: Lawrence Erlbaum Associates, 170-175, 2001.

Cangelosi A., Greco A. & Harnad S., ”From robotic toil to symbolic theft: Grounding transfer from entry-level to higher-level categories”. Connection Science, 12(2), 143-162, 2000.

CAVIAR Project. Caviar test case scenarios, 2004. [http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1; accessed 8-May-2007]

Christensen W.D. & Hooker C.A., ”Representation and the meaning of life”. In Representation in Mind: New Approaches to Mental Representation, The university of Sydney, June 2000.

Clark A., Mindware - ”An Introduction to the Philosophy of Cognitive Science”. Oxford University Press, New York, 2001.

Corvee E., Velastin S.A., Jones G.A., ”Occlusion Tolerent Tracking using Hybrid Prediction Schemes”. In ’Acta Automatica Sinica’, Special Issue on Visual Surveillance of Dynamic Sc 23(3) pp. 356-369, 2003.

Fuentes L.M., Velastin S.A., ”Tracking people for automatic surveillance applications”. Iberian Conference on Pattern Recognition and Image Analysis, Springer Science Online, June, Mallorca, Spain, pp. 238-245. ISBN/ISSN 3-540-40217-9, 2003.

Granlund G.H., ”The complexity of vision”. Signal Processing, 70:101-126, 1999.

Granlund G., ”Organization of architectures for cognitive vision systems”. In H. I. Christensen and H.-H Nagel, editors, Cognitive Vision Systems: Sampling the Spectrum of Approaches, LNCS, pages 39-58, Heidelberg, Springer-Verlag, 2005.

Greenhill D.,Renno J.R., Orwell J., Jones G.A., ”Learning the Semantic Landscape: Embedding scene knowledge in object tracking”. In ’Real Time Imaging’, Special Issue on Video Object Processing 11, pp. 186-203, 2005.

Kelso J.A.S., Dynamic Patterns - ”The Self-Organization of Brain and Behaviour”. MIT Press, 3rd edition, 1995.

Lees B., Corchado J. M., ”Integrated Case-Based Neural Network Approach to Problem Solving”. XPS 1999: 157-166, 1999.

McGarry K., Wermter S., & Maclntyre J., ”Hybrid neural systems: from simple coupling to fully integrated neural networks”. Neural Computing Surveys, http:// www.icsi.berceley.edu/ jagota/NCS, 1999.

Melzer T., Reiter M. & Wildenauer H., Cats and cognitive vision, In Ondrej Drbohlav, editor, Proc. of Computer Vision Winter Workshop, pages 103-108. Czech Pattern Recognition Society, 2003.

Nagel H.-H., ”Steps toward a cognitive vision system”. AI Magazine, 25(2):31-50, Summer 2004.

Orwell J., Remagnino P., Jones G.A., ”Optimal Color Quantization for Real-time Object Recognition”. In ’Real Time Imaging’, 7(5) Academic Press, October, pp. 401-414. ISBN/ISSN 1077-2014, 2001.

PRISMATICA, PRo-active Integrated systems for Security Management by Technological, Institutional and Communication Assistance (GRD1200010601), D7P Innovative Tools for Security in Transports, http://www.prismatica.com

Remagnino P., Shihab A.I., Jones G.A., ”Distributed Intelligence for Multi-Camera Visual Surveillance”. In ’Pattern Recognition’, Special Issue on Agent-based Computer Vision 37(4) Elsevier, April, pp. 675-689. ISBN/ISSN 0031-3203, 2004.

Remagnino P., Jones G.A., ”Classifying Surveillance Events from Attributes and Behaviour”. British Machine Vision Conference, September 10-13, Manchester, pp. 685-694. ISBN/ISSN 1901725162 , 2001.

Renno J.R., Remagnino P., Jones G.A., ”Learning Surveillance Tracking Models for the Self-Calibrated Ground Plane”. In ’Acta Automatica Sinica’, Special Issue on Visual Surveillance of Dynamic Sc 29(3) pp. 381-392, 2003.

Rolls E.T. & Treves A., ”Neural networks and brain function”. Oxford University Press, Oxford, U.K, 1998.

Rumelhart D.E., Hinton G.E. and Williams R.J. ”earning internal representations by error propagation”. Parallel Distributed Processing (David E. Rumelhart and James L. McClelland, eds.), vol. 1, The MIT Press, Cambridge, MA, 1986.

Simon H.A., ”Artificial intelligence: an empirical science”. Artificial Intelligence, v.77 n.1, p.95-127, August 1995.

Troscianko T., Holmes A., Stillman J., Mirmehdi M., Wright D. & Wilson A., What happens next? The predictability of natural behaviour viewed through CCTVcameras. Perception (In Press), 2005.

Ultsch A. & Korus D., ”Integration of neural networks with knowledge-based systems”. In Proc. IEEE Int.Conf.Neural Networks, Perth, 1995.

Watson I., ”Applying Case-Based Reasoning: Techniques for Enterprise Systems”. Morgan Kaufmann Publishers Inc. 250 pages, paper. ISBN 1-55860-462-6, July 1997.

Zehraoui F., Kanawati R. & Salotti S., ”Case base maintenance for improving prediction quality”. In K.D. Ashley and D.G. Bridge, editors, ICCBR 2003, volume 2689 of Lecture Notes is Artificial Intelligence, 703-717. Springer-Verlag, 2003.