NTNU
Innovation and Creativity

Case Based Surveillance System

Thomas Aron Aasen

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Ketil Bø, IDI

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

Problem Description

The master thesis is an extention of the project report "learning from experience" that took part during fall 2005. The tasks are to further design, implement and test a system that is able to achieve learning based on previously experienced cases. The domain is automatic video surveillance and the method used is case-based reasoning.

Assignment given: 20. January 2006
Supervisor: Ketil Bø, IDI

Abstract

Many problems in the field of automatic video surveillance exists today. Some have yet to be overcome. One of these problems is how a computer system automatically can determine if a situation should cause an alarm or not. To resolve this problem, the use of Case-based reasoning (CBR) is proposed. CBR is a technique that allows a system to reason about different situations and to learn from them. The aim is to produce a system that utilizes these abilities. The system should learn to recognize the situations that causes different alarms. When a situation is recognized and categorized, these false alarms can be completely avoided. This master thesis explains and shows the advantages of using such a system together with advanced image processing techniques.

Preface

This master thesis is the result of my work from January 20th to June 16th 2006. It was carried out at the Department of Computer and Information Science (IDI), at the Norwegian University of Science and Technology (NTNU).

I would like to thank my advisor, Ketil Bø, for guiding me and answering my questions.

Thomas Aron Aasen

Trondheim, Norway
16. June 2006

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Automatic video surveillance is common today and its usage will only grow in the future. Systems for security, traffic monitoring etc. are in widespread use throughout the world. All these systems aims to provide the user with correct information about what happens in the monitored area. An important characteristic in automatic video surveillance systems is the fact that it is automatic. This means that the system itself has to interpret the actions it gets from the camera feed. And here lies the difference between automatic video surveillance systems and traditional video surveillance systems. Before these automatic systems came, there always had to be someone watching the feed from the camera. As video surveillance becomes more and more common, the need for automatic systems becomes larger.

A problem in the field of automatic video surveillance is the fact that the systems are not free from errors. It is very hard to produce a perfect system, and there is always a small chance that a system makes the wrong decision once in a while. An alarm sounding from a burglary in progress could just be the wind blowing extra hard in the threes. But how can a computerized system distinguish such situations from others? How can a computerized system determine if an alarm suddenly raised is a false or a real alarm? This paper proposes the use of case-based reasoning (CBR) to solve these problems. CBR is a field within the artificial intelligence community that is widely researched today. The strength of CBR is that it enables a system to learn from previous experienced situations. This again implies that it enables a system to improve over time and with each situation that occurs. In the system proposed, advanced image processing techniques first interprets the image that raised the alarm. This interpretation is made up from several different features extracted from the image. These features again are what makes up the contents of a case. When a case is composed by the image processing part of the system, it passes it on to the CBR part. Now, the idea is that there exists a database with several different cases. CBR is then used to reason about the new case, compare it to the ones in the database and come up with a solution.

The remainder of this thesis is structured as follows. In the first section, a background on related research, automatic video surveillance and existing systems is given. The section continues to give a thorough background on different commonly used image

processing methods. The last part of the sections is devoted to CBR. This part aims to explain what CBR is, and how it allows systems to learn from experience. After the background section, the paper continues to give a presentation of the work done during this master thesis. This section first explains how the system as a whole is designed, before explaining respectively how image processing and CBR is used in the implementation of a system that solves the problem. In the section following, a selection of tests and results from these tests are shown in detail. In the end a discussion of the solution proposed and some insight into alternative solutions is given before the concluding remarks summarizes the achievements made during the work period.

Chapter 2

Related research

There are many cameras mounted around in shops, on streets, in office-buildings etc. today. As the amount of video surveillance and monitoring grows, the amount of data to monitor also grows. With this growth, the number of human eyes required to watch the monitors is simply to high. The cost of setting up a monitoring device is also cheap compared to the price of a human observer. By replacing the human eye with automatic monitoring systems, this problem is solved. Many different systems for automatic monitoring exist today, from very simple ones, to highly advanced systems. The most simple systems are often used in private homes. In these systems, reacting to any kind of movement can be sufficient. In places that needs a high level of security, advanced systems are used, and the human eye often acts as a backup. An example can be in airports around the world. Security is top priority here, and the surveillance system has to work properly and raise alarms when something out of the ordinary happens. Advanced systems can e.g. track moving targets and classify them [5][6], perform face recognition [7], or do other similar tasks.

Automatic video surveillance is really all about automatic video interpretation. That is, interpreting and understanding the different scenes in a video. There exists various beliefs on how this can and should be done. This paper presents the idea of using CBR together with image processing to construct a system for automatic video surveillance. This is a quite new approach to the problem, and the success of this idea is elaborated during the next chapters. However, there also exists other theories and ideas on a solution for this problem. All these ideas have their differences, but they often have many things in common and they share the same goal. The different stages in all common systems are:

1. Something happens in the scene monitored.
2. Acquire the information.
3. Interpret the information.
4. Understand the interpretation.
5. Make a decision.

These five points are included in all systems. When something happens in the scene that is monitored, the system has to acquire information about the scene. This information is not any good without an interpretation, so the next step would be to interpret this information. The system also has to understand the interpretation and lastly make a decision about the scene. This basically means that a working system has to be able to get all relevant information from the scene, and understand what happens in the scene.

Automatic video interpretation has been an area of focus for a couple of decades in the field of cognitive vision. The goal of all these systems is to recognize different scenarios involved in the scene captured on video. Several research units are now defining and finding new solutions to systems that is able to understand the activities that goes on in the scene. The different approaches most commonly used to recognize scenarios can be categorized in the following three categories:

- Probabilistic/neural network combining potentially recognized scenarios.
- Symbolic network that stores totally recognized scenarios.
- Symbolic network that stores partially recognized scenarios.

A natural approach for the computer vision community would be to base a system on probabilistic/neural networks. Taking this approach, each node in the network usually corresponds to scenarios recognized at a given instant with a certain probability. This approach is used in [8], where a system for automatically driving a car based on visual data is elaborated and tested. Another quite similar approach is used in [9], where they use Radial Basis Function (RBF) networks, a type of artificial neural network for application to problems of supervised learning [10]. In that paper they introduce adaptive vision techniques for use in video-conference applications. Such techniques are applied to track motion and recognize faces in the video. With this approach, the network has to be trained extensively with several different versions of a face. A face can be shown in several different angels, and all these angels needs to be incorporated in the training for successful tracking later. Tracking objects can be a very good way to interpret what is going on in the scene. However, it requires the system to use several frames from the video continuously, and that requires a lot of processing power.

For the artificial intelligence community, the natural approach would be to use symbolic networks and to store fully recognized scenarios. In this approach, nodes in the network usually correspond to boolean recognition of scenarios. In [11], the aim is to incrementally recognize certain situations, like states of the scene, events and scenarios, in a video stream, in order to understand what happens in the scene. Several facts are pre-defined, which can be things like type, geometry and property. These facts are all given attributes from certain pre-defined concepts. A concept can be a person or such. From analyzing images from the video, these facts are given different concepts. A scenario is defined as a combination of different facts and concepts. So this means that different scenarios can be recognized from these facts and concepts. The problem here is how to recognize these concepts from the different video streams.

The third approach mentioned in the list is to use symbolic networks and to store

partially recognized scenarios. For example, [12], has used the terminology chronicle to express a temporal scenario. A chronicle is represented as a set of temporal constraints on time-stamped events. The recognition algorithm keeps and updates partial recognition of scenarios using the propagation of temporal constraints based on RETE algorithm[1]. This approach is also used in [13], where an adaptation of temporal constraints propagation for video surveillance is made.

Another interesting approach is described by the work done in [14]. This work presents a new scenario recognition algorithm for video interpretation. The idea here is to use a priori knowledge about the scene. In the scene there can be static objects (counters, plants etc.) and there can be moving objects (people etc.). These objects are all actors in the scenario. Different scenarios are stored with actors and sub-scenarios in a database. Each sub-scenario can be a movement by some actor in the scene from one place to another. The sequence of these sub-scenarios is what determines which main scenario the system believes is in action at any given moment. So, if a certain pre-programmed sequence of sub-scenarios suddenly occurs, and this pre-programmed sequence represents an alarm situation, the alarm is raised.

One of the more extensive projects within the field of video surveillance is the Video Surveillance and Monitoring (VSAM) project that ran from 1997 to 1999 [1]. The research was conducted by the Robotics Institute at Carnegie Mellon University (CMU) and the Sarnoff Corporation. They basically developed a system for autonomous Video Surveillance and Monitoring. The technical approach uses multiple, cooperative video sensors to provide continuous coverage of people and vehicles in a cluttered environment. The system is able to detect and track moving objects in a video sequence by using a combination of temporal differencing [15] and template tracking. To determine if there is movement in a video sequence, the system uses a background subtraction with a three-frame differencing algorithm. Three-frame differencing implies the use of three frames to determine movement from differences in pixel values from frame to frame. Basically the method classifies a pixel as moving if there is change between the first and second frame, and change between the first and third frame. The changes in pixels are captured in pixel intensity profiles, which shows how the intensity change varies over time. An example of such intensity profiles is shown in figure 2.1. Based on these intensity profiles, the system can interpret the movement of objects in the image. For further information on the VSAM project, see [1].

[1]The RETE algorithm is an efficient pattern matching algorithm for implementing rule-based ("expert") systems. The RETE algorithm was designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1979. RETE has become the basis for many popular expert systems.

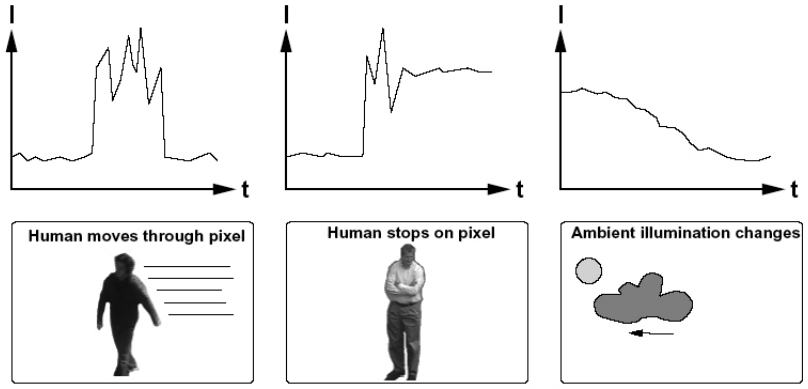

Figure 2.1: Three intensity profiles with corresponding changes in movement [1].

Chapter 3

Background

This chapter gives the reader a background in video surveillance software as well an introduction to several image processing techniques and CBR.

3.1 TrollEye

TrollEye is a system for automatic security and surveillance. The system is developed by TrollHetta AS. It is based on modern methods in image processing and pattern recognition. The standard edition of the system is used for surveillance in homes and small businesses, but the system also comes in an expert version which allows for more intelligent detection algorithms and sensors to be used. Figure 3.1 shows the program with a live camera feed. Several cameras can be added just by connecting them to the USB port of the computer. When a camera is added to the system, alarm areas and sensors can be added directly to the feed from the camera. An alarm area is a four-sided polygon which can be shaped in any form and direction. An example of a defined alarm area is shown in figure 3.2. It is in this alarm area that TrollEye looks for events or changes that can trigger an alarm.

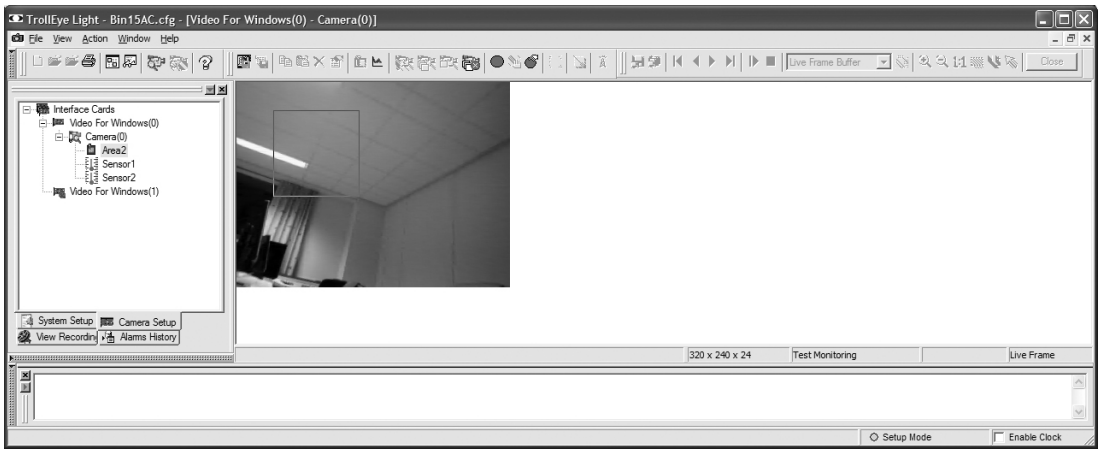

Figure 3.1: TrollEye in setup mode with a live camera feed.

TrollEye uses sensors, devices which raises the alarms should it be necessary. The

Figure 3.2: An alarm area defined by a four-sided polygon, and a dialog bow appearing when an alarm has been raised.

sensors are connected to a sensor card, and sensors can be assigned to a specific camera to detect alarms. A single sensor can also be assigned to multiple cameras at once. Rules are used to link and process alarm areas or sensors. A rule can be defined as a mathematical expression. One example of a rule can be that there has to be an alarm situations in two alarm areas at the same time for the alarm to be raised. By default, one rule is applied for each alarm area. This rule is defined so that an alarm is raised if an alarm occurs in the area. The alarm area is monitored for changes when the system is in recording mode. To follow the movements or changes in an alarm area, a statistical graph can be used. An example of such a graph is shown in figure 3.3.

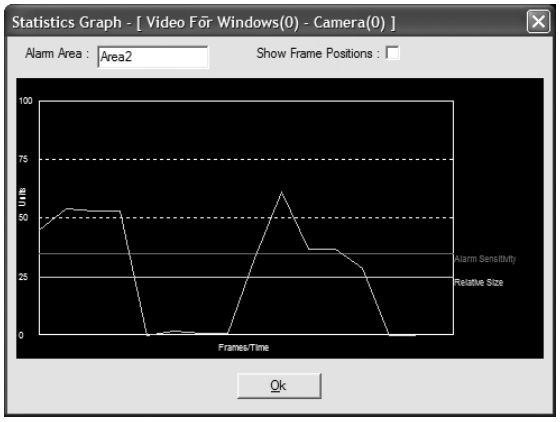

Figure 3.3: The statistical graph shows movement in the alarm area. The threshold for raising an alarm (the alarm sensitivity) based on the movement is also shown in the graph.

TrollEye uses several different DLLs[1] for input devices, pre-analysis, analysis and action. DLLs can be added or removed according to what the user wants the system

[1] A Dynamic Library, also referred to as a Dynamically Linked Library, is a computer library that implements the concept of dynamic linking.

to do. The system also allows for external DLLs to be added. So users can make their own DLLs and incorporate them into the system. The dialog box for adding or removing DLLs is shown in figure 3.4. In the figure, two action DLLs are currently active. The lower one is called EmailAlarm.dll. This is an example of a typical action performed by the system. In this case, an e-mail is sent to some predefined address when an alarm is raised. Input Device DLLs are used for finding and determining the source of the camera feed. Pre-analysis DLLs contains methods that are applied to the alarm area before analysis. These methods can perform different image processing tasks to the alarm area, maybe to make the alarm area more clear, lighter or darker as needed.

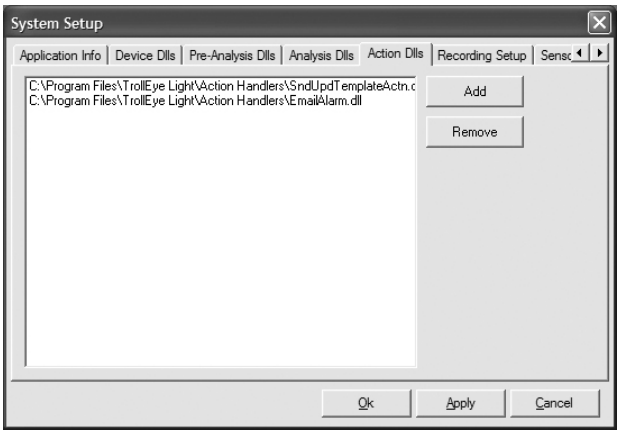

Figure 3.4: The system setup dialog box where various DLLs can be added.

Another important feature in TrollEye is the ability to capture an image which is referred to as a normal scene. A normal scene is used as a reference image and tells the system what the scene looks like when there is nothing going on. The normal scene image is used for continuously comparison with the monitored scene to determine if there are objects of any kind in the image.

3.2 Image processing

3.2.1 Image acquisition

Image acquisition is simply the task of obtaining the image for further use. Images are acquired in the RGB[2] color format. In this format the image is stored as a three-dimensional matrix, where two dimensions determine the location of each pixel while the third dimension determine values for red, green and blue respectively. Storing images in this way makes it easy to work with the images in further processing.

3.2.2 Image pre-processing

This section and the following section are mainly based on work described in [4].

[2]The RGB color model utilizes the additive model in which red, green, and blue light are combined in various ways to create other colors. The very idea for the model itself and the abbreviation "RGB" come from the three primary colors in additive light models.

Pre-processing is a common name for operations to images at the lowest level of abstraction. The aim of pre-processing is a general improvement of the image and to enhance features of the image that are important for further processing. Pre-processing of any image does not increase the amount of information in the image, but it typically decreases amount of information. So it can be argued that pre-processing is such a bad idea, that it is better to not do any pre-processing at all. But pre-processing can be very useful as well. It can help suppress information in the image that is not relevant for the task to be performed. There is typically a lot of redundancy in images. Pixels corresponding to an object in an image, often have the same or similar brightness value. By using this knowledge, distorted pixels can be picked out and corrected by e.g. setting the value to the average pixel value around the distorted pixel. Image pre-processing can be divided into four categories; pixel brightness transformations, geometric transformations, local neighborhood processing and image restoration. Local neighborhood transformation is the one that is most important for this system.

Local pre-processing

In local pre-processing the idea is to use the neighborhood of a pixel in an image to compute the new pixel brightness value. This kind of processing is also known as filtering, because it always implies the use of a filter. There are two main local pre-processing procedures. The first, smoothing, is used to suppress noise in an image. Smoothing also blurs all the sharp edges in the image, which leads to loss of important information. The second procedure, gradient operators, can be seen as the opposite as smoothing. It uses local derivatives in the image. When taking the derivative of an image, areas in the image with rapid change in brightness, are seen as bigger. Edges and other rapid change areas in the image then becomes more clear by using gradient operators. But noise is also an area with rapid change, so by applying this procedure, noise level is also increased. Although smoothing and gradient operators have quite conflicting goals, there exist procedures that allows for smoothing and edge enhancement at the same time.

Smoothing aims to suppress noise in an image, and uses redundancy in the image. As implied, smoothing reduces edge information in the image, but there are smoothing methods that are edge preserving. An example of smoothing is averaging. It uses a filter (also called convolution mask) over the whole image. This way, all pixel values will be averaged by the pixel values in their neighborhood. The size of the filter determines how large the neighborhood is. A 3x3 filter is often used:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.1}$$

This filter will reduce noise but also blur the image, without preserving edge information. Another filter, which preserves the significance of the center pixel and its 4-connected

neighbors, is shown here:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.2}$$

Another method for efficiently reducing noise in an image, is called median[1] filtering. Median filtering is a smoothing method that replaces the current pixel with the median pixel value from a neighborhood. Noisy pixels are efficiently removed as they does not contribute to change the median value in the neighborhood. Median filtering does not blur edges much, so it preserves most of the important information in the image. The algorithm for median filtering is as follows:

Algorithm 3.1 Median filtering [4].

1: Set
$$th = \frac{mn}{2}$$
2: Position the window at the beginning of a new row, and sort its contents. Construct a histogram H of the window pixels, determine the median *med* and record *ltmed*, the number of pixels with intensity less than or equal to *med*.
3: For each pixel p in the leftmost column of intensity p_g perform
$$H[p_g] = H[p_g] - 1$$
Further, if $p_g < med$, set
$$ltmed = ltmed - 1$$
4: Move the window one column right. For each pixel p in the rightmost column of intensity p_g, perform
$$H[p_g] = H[p_g] + 1$$
If $p_g < med$, set
$$ltmed = ltmed + 1$$
5: If $ltmed > th$ then go to 6. Repeat
$$ltmed = ltmed + H[med]$$
$$med = med + 1$$
until $ltmed \geq th$. Go to 7.
6: Repeat
$$med = med - 1$$
$$ltmed = ltmed - H[med]$$
until $ltmed \leq th$.
7: If the right-hand column of the window is not at the right-hand edge of the image, go to step 3.
8: If the bottom row of the window is not at the bottom of the image, go to step 2.

In algorithm 3.1, a median window of m rows and n columns is used. The size of the median window is very important to consider as it has great impact on the result of the filtering. Figure 3.5 shows the use of median filtering with a 3x3 median window and the resulting median value. The pixel that is being processed in the figure, has a value

[1] The median is the value that is found in the middle when the values are sorted.

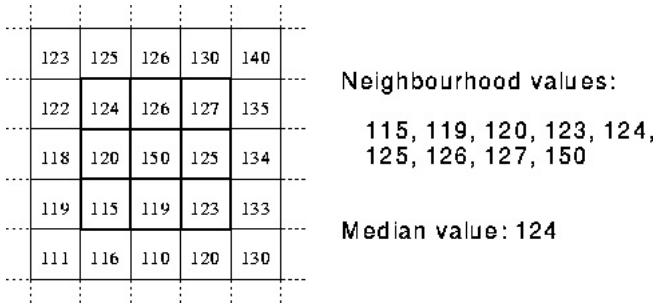

Figure 3.5: Median filtering with a 3x3 median window.

of 150. The new value of the pixel is the median of the neighborhood, which is 124. If for example this pixel instead has a noisy value of 40, the new value of the pixel will still be 124. This shows that median filtering is an efficient approach to noise removal in an image.
For more information about image pre-processing, see [4].

3.2.3 Image segmentation

The aim of image segmentation is to distinguish objects from background in images. The process is an important step towards analysis and further processing of image data. Segmentation can be of a complete or partial nature. In complete segmentation, the result is a set of disjoint regions that completely corresponds to the objects. To achieve complete segmentation, high-level features and specific knowledge of the image needs to be taken into account. But the level of segmentation achieved depends a great deal on the contents in the image. If the background is of a uniform intensity, objects can easily be recognized, and complete segmentation is achieved. An example is black letters printed on a white background. No knowledge of the image data is here needed to obtain complete segmentation. On the other hand, partial segmentation, results in regions that does not completely correspond to the objects in the image. The image is here divided into several regions that are homogeneous with respect to some low-level feature such as brightness. The partially segmented image can contain some overlapping regions, and must then be subjected to further processing based on high-level features. Image segmentation can be divided into three main approaches; threshold techniques, edge-based segmentation and region-based segmentation. A more extensive elaboration on threshold techniques and region-based segmentation follows.

Thresholding

Thresholding is the simplest segmentation method. It works very well for images that contains objects that are quite clearly separated from the background. Basic thresholding is based only on the value of each pixel and does not take any neighboring pixel values into account. The transformation of an input image f to an output image

$$o(i,j) = \begin{cases} 1 & \text{if } f(i,j) \geq T \\ 0 & \text{if } f(i,j) \leq T \end{cases} \tag{3.3}$$

where T is the threshold value. The value of this threshold is very important for successful segmentation. There are several ways to accomplish this. The trivial solution is to apply a global threshold i.e. use the same threshold value over the whole image. However, this method seldom gives very good results, as background and object intensity often varies with the position in the image. Another solution is to use adaptive thresholding. Adaptive thresholding allows for the use of local threshold values that can be a function of local image characteristics. This means that the threshold value can be different depending on the position in the image. One way to do this, is to divide the image into several subimages. Local threshold values can then be determined from the intensity levels in each subimage. Global thresholding is determined from the whole image f as:

$$T = T(f) \tag{3.4}$$

Adaptive, local thresholding is also dependent on position in the image:

$$T = T(f, f_c) \tag{3.5}$$

f_c is here the part of the image where the local threshold is applied.
Many other threshold techniques also exist. One of these other procedures is called semi-thresholding. With this procedure, the objects keep their original intensity level:

$$o(i,j) = \begin{cases} f(i,j) & \text{if } f(i,j) \geq T \\ 0 & \text{if } f(i,j) \leq T \end{cases} \tag{3.6}$$

With this procedure, it is easier for the human eye to analyze the objects in the image. The only change is in the background, which is set to a constant intensity level.

Region-based segmentation

The aim of region-based segmentation, unlike the previous mentioned technique, is to locate and create regions right away. The main strength of region-based segmentation is that it works very well with images with much noise. Region-based segmentation divides the image into regions based on homogeneity in regions of the image. Homogeneity in image regions can be based on gray-level, color, shape, etc. The complete segmentation of an image R is expressed by a finite set of regions R_1,...,R_S, such that

$$R = \bigcup_{i=1}^{S} R_i \quad R_i \cap R_j = \emptyset \quad i \neq j \tag{3.7}$$

In region-based segmentation, two extra assumptions are needed:

$$H(R_i) = TRUE \quad i = 1,2,...,S \tag{3.8}$$

$$H(R_i \cup R_j) = FALSE \quad i \neq j, \quad R_i \text{ adjacent to } R_j \tag{3.9}$$

S is her the total number of regions and $H(R_i)$ is a homogeneity evaluation of the region R_i. After segmenting, the resulting regions must be homogeneous. The resulting regions must also be maximal, which means that when one region is merged with an adjacent region, the homogeneous criterion is false. The simplest criterion to use when determining homogeneity is simply the gray-level. Region-based segmentation

is often referred to as region growing, and the most straight-forward way to do this, called region merging, is to think of each pixel in the image as a region on its own. A pixel on its own will seldom satisfy (3.9), so the procedure is to merge regions as long as (3.8) is satisfied. Differences in the result can occur because merging can be done in different ways and in different order.

3.2.4 Mathematical morphology

Mathematical morphology [16] is a technique that affects shapes and boundaries of regions in an image. In this application it is used to erase small objects and to group fill holes in detected objects.

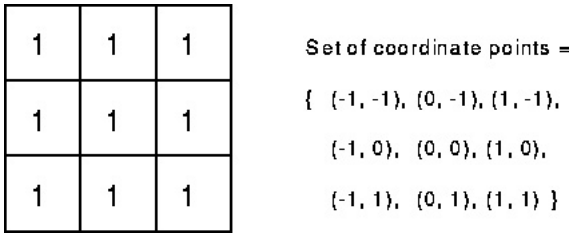

Figure 3.6: A structuring element used in mathematical morphology [2].

Erosion

Erosion is one of the the two (erosion and dilation) basic operations within mathematical morphology. When applied to binary images, it is used to erode away the boundaries of white pixels. The erosion operation takes two parameters. The image which erosion is to be applied to and a structuring element. The structuring element is what decides the the effect of the erosion on the image. The mathematical definition of erosion for binary images is as follows:

- Suppose that X is the set of Euclidean coordinates corresponding to the input binary image, and that K is the set of coordinates for the structuring element.
- Let K_x denote the translation of K so that its origin is at x.
- Then the erosion of X by K is simply the set of all points x such that K_x is a subset of X.

The effect of erosion applied to an image can be seen in figure 3.7. The white areas have become smaller as the black areas are growing.

Dilation

Dilation is the other typical operation within the field of mathematic morphology. It is basically the opposite of erosion. As erosion is used to erode away the white pixels, dilation is used to enlarge the boundaries of the white pixels. The mathematical definition of dilation for binary images is as follows:

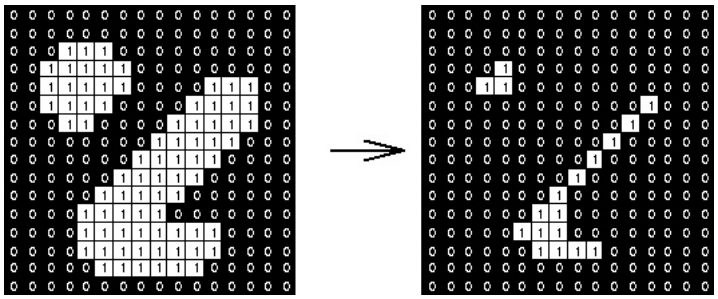

Figure 3.7: The effect of applying erosion to an image [2].

- Suppose that X is the set of Euclidean coordinates corresponding to the input binary image, and that K is the set of coordinates for the structuring element.
- Let K_x denote the translation of K so that its origin is at x.
- Then the dilation of X by K is simply the set of all points x such that the intersection of K_x with X is non-empty.

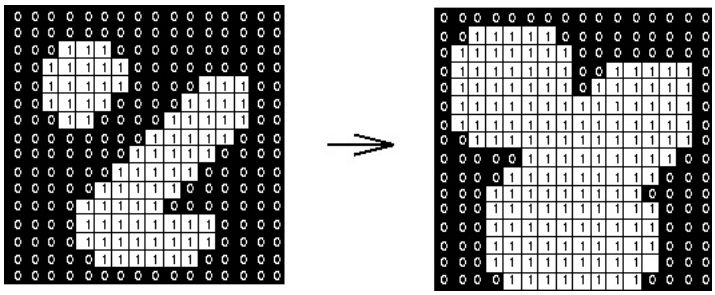

Figure 3.8: The effect of applying dilation to an image [2].

The effect of dilation applied to an image can be seen in figure 3.8. The white areas have become larger as the black areas are getting smaller.

Opening

Opening, together with closing, are two other important operation in mathematical morphology. They both are the result of combining erosion and dilation. An opening is performed by taking an erosion followed by a dilation, using the same structuring element for both operations. Doing this removes small white holes in a black object, i.e. fills gaps.
Figure 3.9 shows the effect of applying an opening to a binary image.

Closing

Closing is basically the opposite of opening. It is performed by taking a dilation followed by an erosion.
The effect of performing a closing on a binary image is shown in figure 3.10.

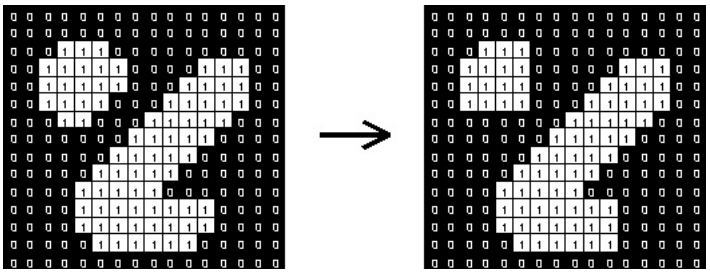

Figure 3.9: The effect of applying an opening to an image [2].

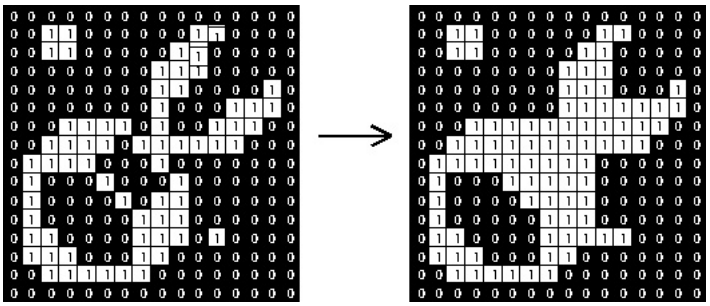

Figure 3.10: The effect of applying a closing to an image [2].

3.3 Case-based reasoning

Case-based reasoning (CBR) is a relatively new technique in the field of problem solving and learning [3]. Over the last few years is has gotten more and more attention as its opportunities has become more clear. CBR is a paradigm for problem solving, that is somewhat different from other AI problem solving paradigms. Different in the way that CBR does not only rely on general domain knowledge, but also on the specific knowledge of previously experienced cases. CBR has been applied in several different systems and in several different domains. Reasons for applying CBR can be many. In Casey [17] it is applied to achieve a more effective reasoning about a complicated domain. Casey is one of the most successful CBR systems. It reasons about heart deceases diagnostics, and acts as a effective version of the Heart Failure program [18]. Casey is also a safe system, because if it does not find a solution it uses the Heart Failure program as backup. Another system that is implemented is IBP [19]. This system uses CBR to predict the outcome of trade-secret lawsuits. Other systems such as PROTOS [20] and CARMA [21] also shows very promising results within their domains.

The basic idea of CBR is to use a database of previous cases to find a solution for any new case. When a new case is presented to the system, specific knowledge from cases in the database is used to locate the cases in the database which are most similar. Specific knowledge here can be anything that is relevant for the problem which the system is aiming to solve. This is obviously very different from system to system, depending on what features are important in each problem scenario.

The general steps in any CBR system are:

- Identify all features from the new case presented to the system.
- Locate the most similar case from the database.
- Use the most similar case to suggest a solution for the new case.
- Evaluate this solution.
- Have the system learn from this experience.

3.3.1 Learning with CBR

Learning is a very important part of CBR systems. The idea comes from the field of machine learning, where the rules for the system are derived in the system itself and not by the programmer. Rules are made and updated continuously while scenarios or cases are presented to the system. Learning in CBR in mainly the case update procedures. Cases can and will usually be updated after each new case is solved. The experience from solving a new case is retained in order to solve similar problems in the future. CBR favors learning from experience, since it is usually easier to learn by retaining a concrete problem solving experience than to generalize from it. Learning in this kind of way also resembles the way that humans learn. Humans uses previous experience to reason about a current situation, and learn more and more as more situations are encountered. But this kind of learning really requires a set of well formulated and worked out methods for deciding which parts of the experience is relevant to learn from. Learning methods needs to extract all relevant knowledge and discard knowledge that is of no use. If this is not the case, the system can be reasoning wrongly about future cases, and the more cases presented, the more mistakes the system is likely to make.

There is always some kind of learning when a new case is presented to the system. The case base is updated regardless of how the problem was solved, and each time the case base is updated, the system is learning something new. There are mainly three issues in case-based learning. These are how to extract relevant knowledge, how to index the case after learning and integration of the new knowledge learned.

Extraction of relevant knowledge can be done in many ways. The most obvious way is to store all the problem specific information as knowledge. This is in many systems sufficient and useful. Other systems also store an explanation or some sort of justification of how and why the solution to the problem was found. By including the explanation, cases in the future can be compared by means of the explanations for better modification. Another quite typical source of relevant knowledge to store is the method that was used to obtain the correct solution. This can be extremely useful in later case problem solving. Simply by reusing the method the old solution was obtained by, the new case can be solved correctly.

Indexing of cases and knowledge in the case base is very important for future retrieval. The problem is really how the features of each case should be indexed and compared.

One solution is to think of all the features as equally important, and do a full comparison between a new case and cases in the case base. But with this approach all the features acts as indexes. Another way is to give features different scales of contribution depending on the importance of the feature. If one feature is generally found to be very important for determining the solution, the cases in the case base can be indexed on behalf of this feature. When the new case is presented, it will be compared by this feature first. Doing it this way will help speed up the system and the retrieval phase.

Integration of new knowledge into the case base basically means changing the indexing. One example is that the new knowledge can lead to a change in the strength of a distinct feature in one or more cases. This has to be done in a smart way, so that the system can improve with every case, and provide better reasoning for future cases.

3.3.2 The CBR cycle

In general, CBR can be described by four steps:

1. RETRIEVE the most similar case
2. REUSE the information from this case to solve the problem
3. REVISE the solution case
4. RETAIN the parts of this experience likely to be useful in the future

These steps can be viewed on as a cycle, see figure 3.11, where each step is processed in turn.

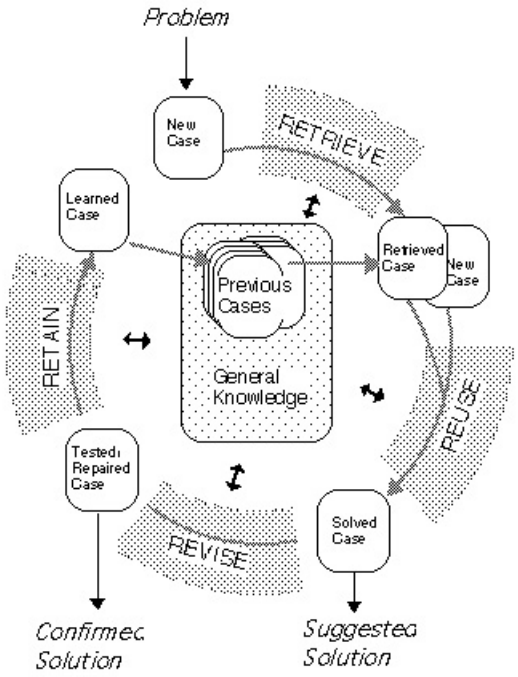

Figure 3.11: The CBR cycle [3].

The cycle shows the different stages of a CBR system, and how these stages communicates with cases and knowledge. Previous cases and general knowledge is drawn in the middle, and makes up the knowledge base. An explanation of the different stages follows.

Case retrieval

The first stage in the cycle is case retrieval. It starts out with a description of the new case, and uses this description to locate similar cases from the case base. All relevant features from the new case will have to be extracted, and compared to features from all cases in the case base. There are several problems that must be addressed in this phase. First of all, deciding which features are most important for determining case similarity. A new case can often be similar to an old case in many ways, but maybe some important features of the cases are totally different. Two really relevant cases can also sometimes appear to be quite different. This is called the matching, or similarity-assessment, problem. One way to deal with this problem is to compare cases in several different ways. Another problem can be that a feature is relatively unknown, and occurs only seldom. It is then hard to determine if the value of this feature is big or small, because there is no other value to compare it to. This is called the situation-assessment problem.

The goal of this task is to end up with one or a few previously experienced cases which are similar to the new case presented. To select the most similar cases, a similarity threshold is applied. All cases which are above this certain threshold will be fetched from the case base. When the case base is quite big, this often means that several cases is found to be above the threshold value. To find only the most similar case, some sort of selection method has to be applied. When only one case, which will be the proposed solution, is found, this stage is over.

Case reuse

Next in the cycle is the case reuse stage. This stage is also often referred to as case adaptation. Reuse of the retrieved previous case solution focuses on two things, namely the differences between the past and the new case, and what part of the retrieved case can be transferred to the new case. The most simple way to do this is to abstract away all differences between the cases, and to use the solution of the retrieved case directly as the solution for the new case. This is a somewhat trivial example of reuse. Another approach is to take the differences into account. This requires an adaptation process that deals with these differences. There are mainly two different ways to reuse past cases. The first is to reuse the past case solution, which is known as transformational reuse. With this approach, the past case solution is not used directly as a solution for the new case, but there exists knowledge in the form of transformational operators, T, that when applied to the past case solution creates a solution for the new case. Transformational operators can be indexed around the differences detected among the retrieved and new cases. A strong domain-dependent model in the form of transformational operators is required, because it focuses on the solutions instead of how the problem is solved.

A second approach is known as derivational reuse. Here, the method that leads to the solution is reused. Every case in the case base has to hold on to information about the steps that led to the solution. These steps are reused for the new case, and a solution for the new case is created.

Case revise

In the revise stage, the solution is tested and evaluated by the system. If the outcome of the evaluation is successful, the solution is approved. If it's not, the solution case will be repaired either by some method or by user interaction. The stage can be divided into two parts, evaluate solution, and repair fault. In the evaluation part the solution is applied in a real environment and the result is approved or disapproved. If the result from the evaluation is approved, then this stage is over. But if it is disapproved, the errors are detected and explanations are generated.

Case retain

The final stage is retain. This is where the learning takes place. This task will update the knowledge base with the parts of this experience that are useful to retain. Learning is triggered by the success or failure of the case evaluation. Subtasks to retainment are usually selecting what information to store, in what form to store it, how to index the case in the case base, and how to integrate the case in memory.

No matter what the outcome of the reasoning is, there is always some learning involved. Either the case was solved by the use of some previous case, which can lead to the creation of a new case or a generalization of the old case, or the case was solved by some other method, which will lead to the creation of a new case. In any case a decision must be made about what to use as source for learning. Description of problem and solution are usually a good source for learning, but explanations and justifications of why solutions are correct can also be included as learning sources. Extraction of the method that solved the problem is also a possible source.

3.3.3 CBR and image processing

The use of CBR in image interpretation systems is proposed in [22]. As image interpretation systems are becoming increasingly popular in several different applications, the need for robust and flexible systems are getting bigger. CBR is a strategy that can provide robustness and flexibility. It is a powerful method for controlling the image processing in all phases. As yet, CBR is not widely used in image interpretation systems. This may be because CBR is not very well known within the image interpretation community. Images are a difficult domain to model. CBR generally does not rely on a well-formulated domain theory, and can therefore be a very good approach in image interpretation systems.

Chapter 4

Solution

This chapter focuses on the implementation of the system, and gives an explanation of the different choices made.

4.1 General system design

The system is designed and implemented in two major parts, as seen in figure 4.1.

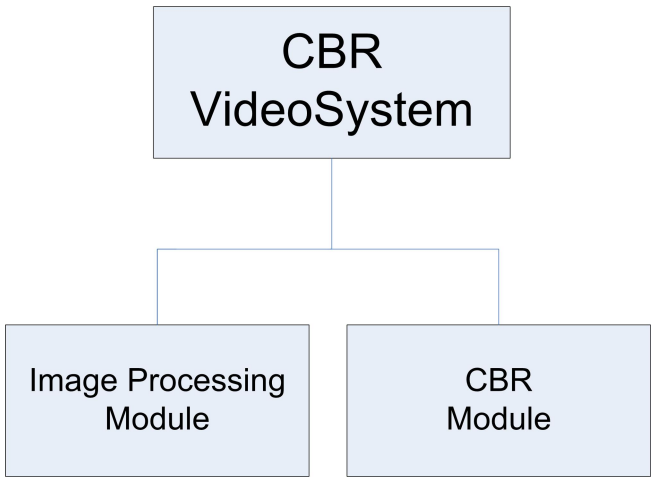

Figure 4.1: System overview.

As this system really deals with reasoning about images, the two different modules are the image processing module and the CBR module. The main task in the image processing module is to extract features from images, while the CBR module deals with all reasoning tasks. These two modules are quite independent of eachother, however, there are also parts that ties them together. One factor that is shared by both modules is knowledge about how a case is built up. Both module needs to have this knowledge. The image processing module is supposed to fill in the information about each case, and the CBR module is supposed to use this information. Therefore, should a change in case structure occur, both modules needs to be changed in various places. The overall main flow of information in the system is shown in figure 4.2.

This figure shows that the system trigger is when an alarm is raised. This information is passed on to the image processing module in the form of a reference image and an alarm image. Features from the images are here extracted, and sent to the CBR module which reasons about the case and produces a solution.

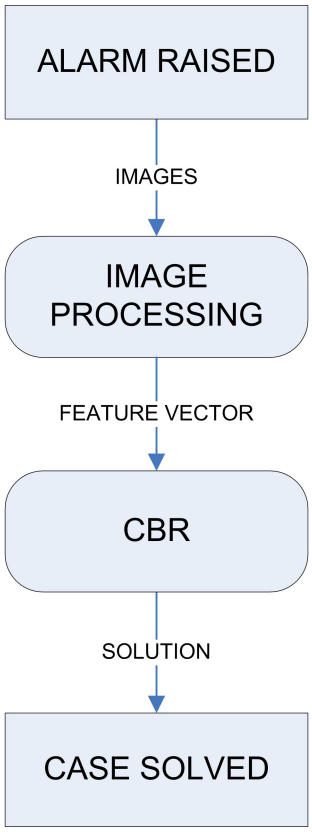

Figure 4.2: System information flow.

4.2 Image processing module

The image processing module deals with all image processing tasks in the system. It is one of the two large modules (image processing and CBR). This module uses a lot of methods from the open source library IPL 98 as well as other implemented methods. The major goal of the image processing module is to prepare images for feature extraction and to perform the feature extraction itself. Figure 4.3 shows the class diagram of the Ip class that is the implementation of the image processing module. The basic flow within the image processing module can be seen in figure 4.4. The figure shows the two images first being loaded into the the program. When this is accomplished, the next step is to convert both of them to grayscale images. They then both undergo the same pre-processing procedures before a difference image is computed. This difference image also goes through pre-processing procedures before the features are extracted from both the input images and the difference image. This module then ends up with a feature vector which is passed on the CBR module.

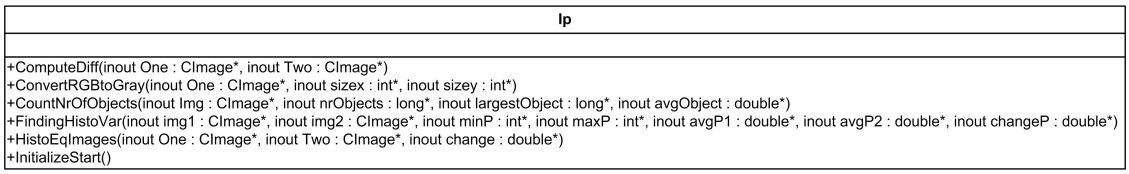

Figure 4.3: Class diagram of the image processing module.

4.2.1 IPL 98

In the implementation, an existing open source image library called IPL (Image Processing Library) 98 [23] is used. This library provides several methods for different image processing tasks. From simple ones such as loading a bmp[1] file, to heavier processes such as segmentation techniques. In this system IPL 98 is used to:

- Load images
- Save images
- Perform median filtering
- Perform mathematical morphology

IPL 98 loads and saves images in bmp format.

4.2.2 Pre-processing

Image pre-processing is very important for the result and for the rest of the image processing modules functions. The system uses pre-processing in two stages. At first the grayscale reference image and alarm image are pre-processed by median filtering. The second stage of pre-processing happens when the difference image is computed. The difference image also goes through median filtering. In addition, the difference image is processed by mathematical morphology, to remove any small disturbance left in the image after median filtering.

Median filtering

The images are median filtered in several stages. First, the grayscale converted original images are median filtered to remove any noise that may be present in the images. Both the reference image and the alarm image are median filtered using the exact same filter method. These images will later be compared to produce a difference image, so it is important that all the same actions are taken on both images. This difference image is also median filtered to further remove any noise caused by the production of the difference image. Using a medial filter on the difference image removes small objects in the image. The whole process and an example of the result can be seen in figure 4.5. The reason for using median filtering is that is removes noise, but at the same time preserves details in the image. Median filtering preserves details better than for example a mean filter [24], which is another alternative for noise removal.

[1] Bmp (Bitmap Picture) is a bitmapped graphics format used internally by the Microsoft Windows graphics subsystem, and used commonly as a simple graphics file format.

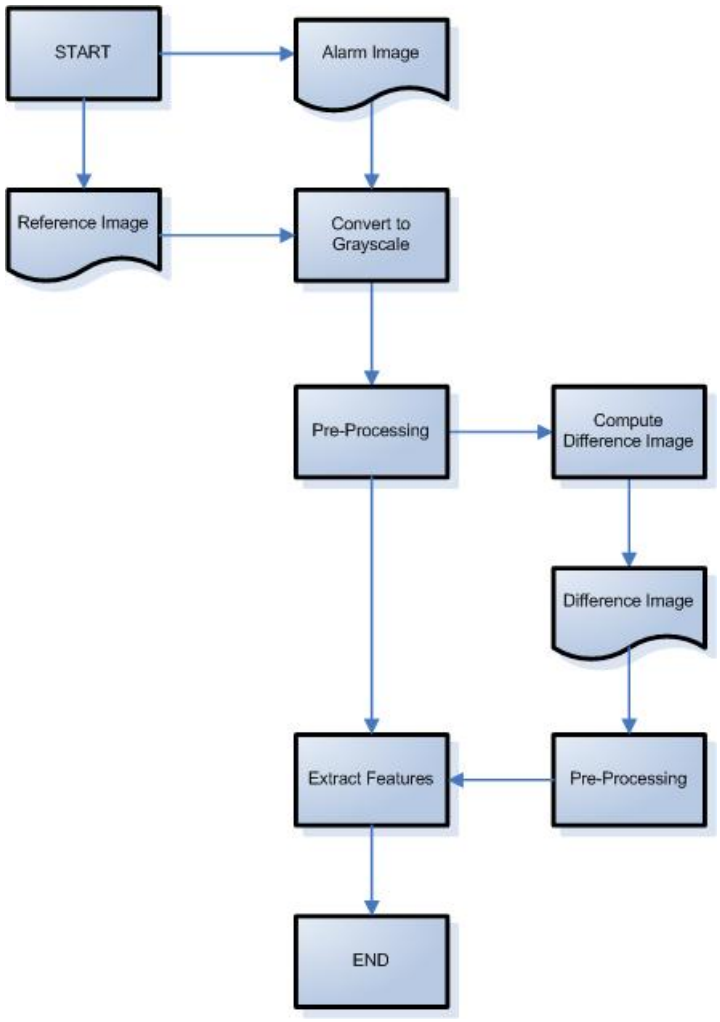

Figure 4.4: Flow in the image processing module.

In figure 4.5 it is clear to see that noise has been removed in both stages of median filtering. Another positive outcome of median filtering can be seen in the latter stage of filtering (where the difference image is median filtered). The small white holes in the big black object has been filled, so that the object is also better segmented by applying median filtering.

The actual algorithm for median filtering can be seen in algorithm 3.1. The program uses a built in version from IPL 98 to perform median filtering. This version applies median filtering in two stages. First it applies a four connected filtering, see figure 4.6, that sets the pixel X to the median value of the five gray pixels shown in the figure. Second, a diagonally connected filtering, see figure 4.7, is applied. This time the pixel X is set to the median value of the five diagonally connected gray pixels shown in the figure.

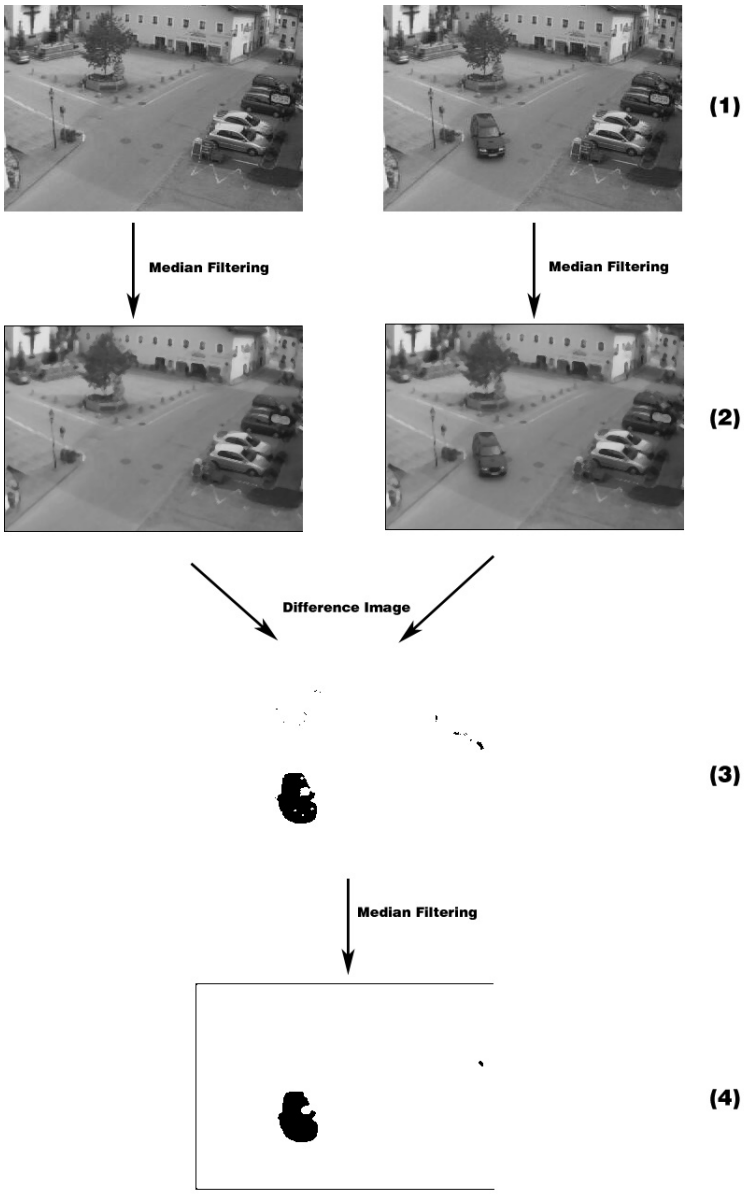

Figure 4.5: (1) The two original images (reference image and alarm image). (2) Result of applying median filtering to (1). (3) The difference image calculated from (2). (4) Result of applying median filtering to (3).

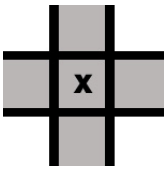

Figure 4.6: Median filtering with four connected regions.

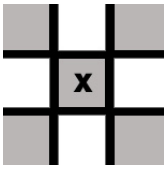

Figure 4.7: Median filtering with diagonally connected regions.

Mathematical morphology

Mathematical morphology is applied to isolate the objects in the image that are of particular interest. Small pixel variations from noise and moving trees can quickly be reduced or eliminated by correct use of mathematical morphology.

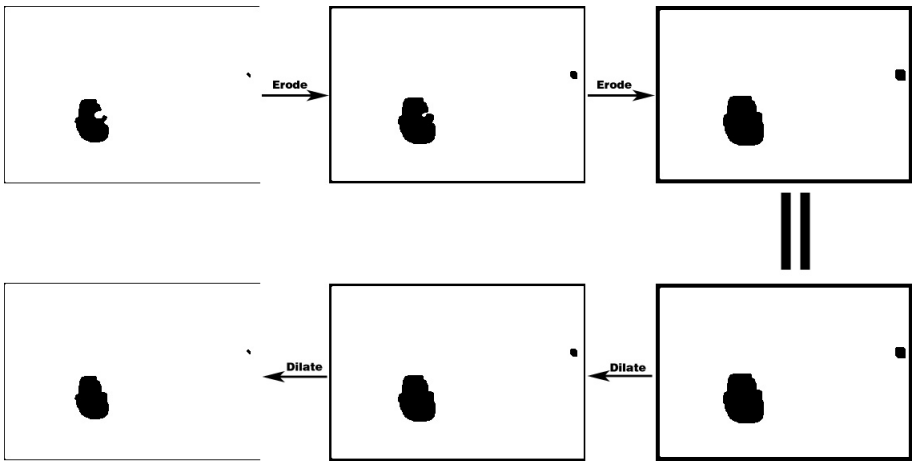

Figure 4.8: Image going through two phases of erosion and two phases of dilation.

If figure 4.8 the effect of mathematic morphology on a typical difference image is shown. The original difference image is shown in the upper left corner, and the result is shown in the lower left corner. Notice the small white dent on the right side of the large object. During the morphology phase the image goes through two erosions and two dilations. In figure 4.8 it is clear to see the changes in the large object during each stage. During the erosion the object is getting bigger, and white holes are filled with black pixels. During dilation the object is getting smaller or, in other words, returning to the original size of the object. The result clearly shows that the unwanted white dent in the large object is filled, and the object has straighter boundaries which reflects the reality of the object better.

4.2.3 Feature extraction

Extracting of the features representing each case is done during the image processing module. These features are what makes different cases unique, so determining which features to extract is an important decision. As this system is quite simple and ought to run fast, the features chosen are features which are quite easy to extract. The selected features focuses on the different intensities in the images as well as information about objects in the image. Selected features are:

- average intensity in the alarm image.
- change in intensity from reference image to alarm image.
- average object size in difference image.
- largest object size in difference image.

- number of objects in difference image.

The reason for focusing on objects found in the alarm image is obvious. These objects have most likely triggered the alarm. The number of objects and their sizes are often the only features separating a case from another. To store the information about these features, the structure "a_case" is created. A structure is a very nice way to store information about the cases, because all the information needed can be contained within the structure.

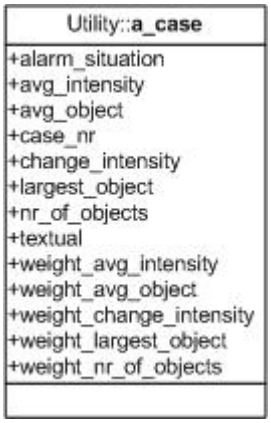

Figure 4.9: Structure a_case

As seen in figure 4.9, the structure contains several different variables that describes the case. These are:

- alarm_situation: defines if the case represents an alarm or not.
- avg_intensity: the average intensity in the alarm image.
- avg_object: the average size of the objects found in the alarm image.
- case_nr: a number to identify the case.
- change_intensity: the change in intensity from the reference image to the alarm image.
- largest_object: the size of the largest object found in the alarm image.
- nr_of_objects: the number of objects found in the alarm image.
- textual: a textual description of the case.
- weight_avg_intensity: the weight of the avg_intensity variable.
- weight_avg_object: the weight of the avg_object variable.
- weight_change_intensity: the weight of the change_intensity variable.
- weight_largest_object: the weight of the largest_object variable.
- weight_nr_of_objects: the weight of the nr_of_objects variable.

4.2.4 Converting a color image to grayscale

This function, called ConvertRGBtoGray, is used to convert an image represented in RGB colors to grayscale. The reason for converting the images to grayscale is that colors are of no concern in this system. It only works with intensity images and get all the information it needs from these images. Doing this straight away also reduces processing time, as image pixels are stored with 8 bits in intensity images as opposed to 24 bits in full color images.
The output from this function is an intensity image with values from 0 to 255. An example of a result from this particular grayscale conversion is shown in figure 4.10.

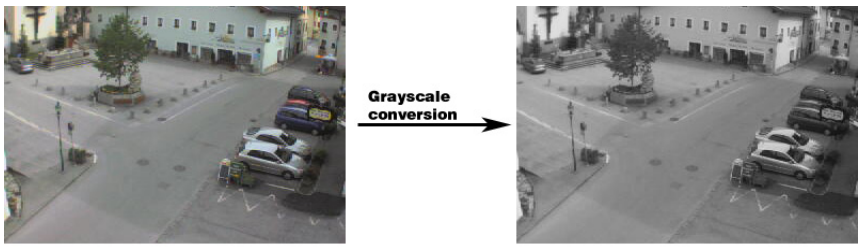

Figure 4.10: Result of grayscale conversion.

The color image is converted to grayscale using the algorithm 4.1.

Algorithm 4.1 Converting RGB to grayscale.

1: $A \leftarrow$ read RGB image
2: $Gray \leftarrow$ A
3: **for** $i = 0$ to $length(A)$ **do**
4: **for** $j = 0$ to $height(A)$ **do**
5: $Gray[i, j] = 0.299 * A[i, j, 1] + 0.587 * A[i, j, 2] + 0.114 * A[i, j, 3]$
6: **end for**
7: **end for**

4.2.5 Computing the difference between two images

This function, called ComputeDiff, is used to compute the difference image from the reference image and the alarm image. The reference image and the alarm image are taken in as parameters and a difference image is returned as output. The function compares each pixel value in the two input images, and turns the corresponding pixel in the output image on or off depending on the value of difference from the two input images. If the difference is above a predefined threshold, the corresponding pixel in the output is set to black. If they are sufficiently similar, the pixel is set to white. The value of the threshold is important. If this value is set too low, too many of the small disturbances in the image are passed on the difference image. On the other hand, if the threshold value is too high, important features in the image may be ruled out, and wont be passed on to the difference image. The threshold value can very easily be changed to test different values, and to find the best one. Figure 4.11 shows what the input images and the resulting difference image when the threshold is set to 30.
The algorithm for computing the difference image is shown in algorithm 4.2.

Figure 4.11: Computing the difference of two images with intensity threshold 30.

4.2.6 Performing intensity correction on the images

This function is called HistoEqImages and is used to correct intensities in both the reference and the alarm image to enhance the quality of the difference image. If the difference in intensity between the two images are too large, this can have a bad effect on which objects that are found in the difference image. Many areas in the image that are not really objects, but maybe just a shadow can be mistaken as objects if this difference is too large. By reducing this difference, this can be avoided. Both images are corrected to make the average intensity in the images as close to 122[2] as possible. Algorithm 4.3 shows what happens during this intensity correction. The

[2] 122 is the gray value right in the middle of the scale that goes from 0 (black) to 255 (white).

Algorithm 4.2 Compute difference image.

```
1: A ← read reference image
2: B ← read alarm image
3: for i = 0 to length(A) do
4:     for j = 0 to height(A) do
5:         if abs(A[i, j] − B[i, j]) < threshold then
6:             OUTPUT[i, j] = 255
7:         else
8:             OUTPUT[i, j] = 0
9:         end if
10:    end for
11: end for
```

important thing that happens in this algorithm is that a correction value for each image is computed. These values represent the gap between the average intensity values in the images and 122. The latter part of the algorithm corrects each pixel in the images by these correction values, but only if the resulting intensity ends up between 0 and 255.

Algorithm 4.3 Intensity correction.

```
1: A ← read reference image
2: B ← read alarm image
3: SumA ← sum of values in image A
4: SumB ← sum of values in image B
5: for i = 0 to length(A) do
6:    for j = 0 to height(A) do
7:       SumA+ = A.GetPixel(i, j)
8:       SumB+ = B.GetPixel(i, j)
9:    end for
10: end for
11: CorrectA ← the correction value for image A
12: CorrectB ← the correction value for image B
13: CorrectA = 122 − (SumA/(length(A) ∗ height(A)))
14: CorrectB = 122 − (SumB/(length(B) ∗ height(B)))
15: for i = 0 to length(A) do
16:    for j = 0 to height(A) do
17:       if (A.GetPixel(i, j) + CorrectA) > 255 or A.GetPixel(i, j) + CorrectA) < 0
          then
18:          DoNothing
19:       else
20:          A.SetPixel(i, j) ← A.GetPixel(i, j) + CorrectA
21:       end if
22:       if (B.GetPixel(i, j) + CorrectB) > 255 or B.GetPixel(i, j) + CorrectB) < 0
          then
23:          DoNothing
24:       else
25:          B.SetPixel(i, j) ← B.GetPixel(i, j) + CorrectB
26:       end if
27:    end for
28: end for
```

4.2.7 Finding the histogram variables

This function, called FindingHistoVar, is used to find several intensity attributes in the reference image and the alarm image.
From the reference image it computes:

- average intensity in image.

From the alarm image it computes:

- average intensity in image.
- value of the lowest intensity in the image.
- value of the highest intensity in the image.

By using the average intensity values from the two images, it also computes the change in the intensity from the reference image to the alarm image. Algorithm 4.4 shows what happens during this procedure. The algorithm is pretty straightforward.

Algorithm 4.4 Finding the change in average intensity and histogram max and min.

$A \leftarrow$ read reference image
$B \leftarrow$ read alarm image
$CounterA \leftarrow$ counter for values in reference image
$CounterB \leftarrow$ counter for values in alarm image
$max = 0 \leftarrow$ variable that remembers highest intensity value
$min = \infty \leftarrow$ variable that remembers lowest intensity value
for $i = 0$ to $length(A)$ **do**
  for $j = 0$ to $height(A)$ **do**
    $CounterA = CounterA + A.GetPixel(i, j)$
    $CounterB = CounterB + B.GetPixel(i, j)$
    if $B.GetPixel(i, j) > max$ **then**
      $max = B.GetPixel(i, j)$
    else if $B.GetPixel(i, j) < min$ **then**
      $min = B.GetPixel(i, j)$
    end if
  end for
end for
Computing the average intensity and the change in the average intensity:
$AVGINT_{refimage} = \frac{CounterA}{length(A) * height(A)}$
$AVGINT_{alarmimage} = \frac{CounterB}{length(B) * height(B)}$
$AVGINT_{change} = AVGINT_{alarmimage} - AVGINT_{refimage}$

4.2.8 Counting objects in the alarm image

This function, called CountNrOfObjects, is used to find object information from the difference image computed from the reference image and the alarm image. It uses region identification techniques to label each object in the image. By doing this it can quickly find the number of objects in the image, and the size of each of these objects. 8-connected region identification is used in this system. Objects that are close enough to be identified as the same object in 8-connected region identification can sometimes be identified as two different objects in 4-connected region identification. In this system there is no reason to classify two such close objects as two different objects, therefore 8-connected region identification is preferred. Algorithm 4.5 shows the steps taken in region identification.

Algorithm 4.5 4-connected and 8-connected region identification.

1: First pass: Search the entire image *R* row by row and assign a non-zero value *v* to each non-zero pixel *R(i,j)*. The value *v* is chosen according to the labels of the pixel's neighbors, where the property *neighboring* is defined by figure 4.12 ('neighbors' outside the image *R* are not considered),

- If all the neighbors are background pixels (with pixel value zero), *R(i,j)* is assigned a new (and as yet) unused label.
- If there is just one neighboring pixel with a non-zero label, assign this label to the pixel *R(i,j)*.
- If there is more than one non-zero pixel among the neighbors, assign the label of any one to the labeled pixel. If the labels of any of the neighbors differ (*label collision*), store the label pair as being equivalent. Equivalence pairs are stored in a separate data structure - an equivalence table.

2: Second pass: All of the region pixels were labaled during the first pass, but some regions have pixels with different labels (due to label collisions). The whole image is scanned again, and pixels are re-labeled using the equivalence table information (for example, with the lowest value in an equivalence class).

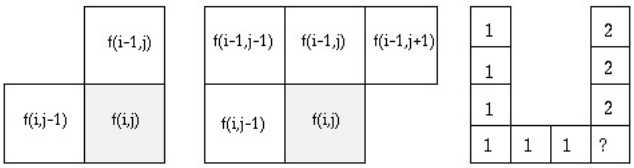

Figure 4.12: From left: Mask for identifying 4-connected regions. Mask for identifying 8-connected regions. Label Collision

4.3 CBR module

The CBR module deals with the reasoning parts of the system. It is started when a new feature vector is supplied by the image processing module. It then performs a search in the case database, locates the best cases and reasons about these. The module ends with a solution to the case in the form of a selected case from the database or the creation of a totally new case. A class diagram from the implementation of this module is seen in figure 4.13.

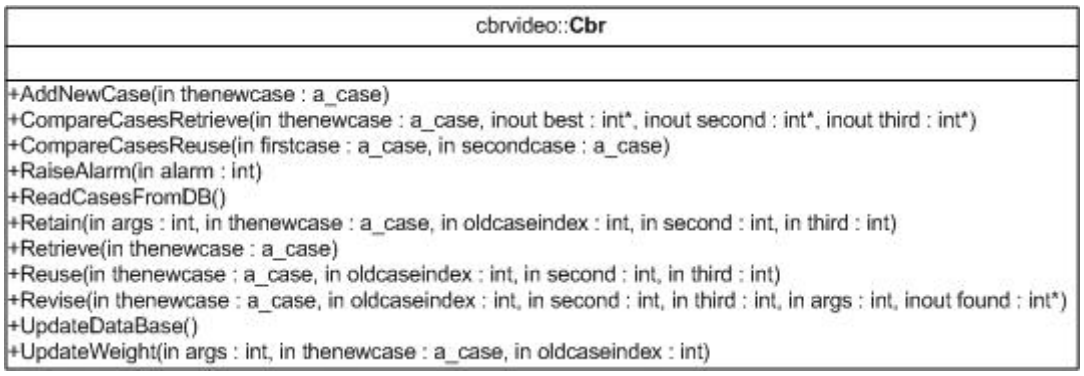

Figure 4.13: Class diagram of the CBR module.

4.3.1 Case database

The case database is where all cases is stored. It contains all the information needed for each case. For each case it holds the following information:

- A separation point to notify that a new case is starting.
- A number on the case.
- All data from the structure a_case.

The entire case database gets read into the program when a new case is presented. Values that needs to be updated are updated in the system itself and not directly in the database. After the program is finished running, the database is updated, and all necessary changes are made to the database.

4.3.2 Retrieve

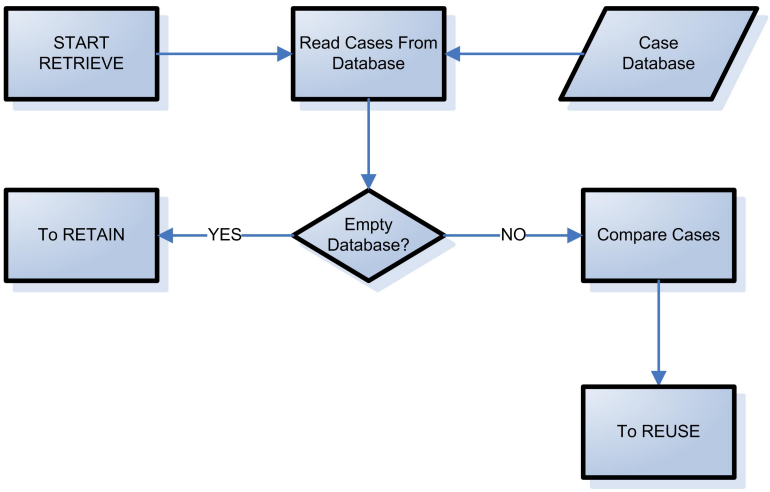

Figure 4.14: Flow diagram of the retrieve stage.

The retrieve stage is the first stage in the CBR module. Its main task is to compare the new case with all cases in the database, and to select the best cases to pass on to the next stage. Initially, the stage reads all cases from the database into the system. This is done by reading a case at a time from the database, and putting the case into an array of cases locally in the program. When all cases are read, it moves on to the next stage, which is to compare the new case with all these old cases. This stage compares the new case with each case one by one. This specific comparison procedure also considers the weights corresponding to each feature. The weights makes sure that each feature only counts on the total case resemblance value according to the value of the weight. The algorithm used can be seen in algorithm 4.6.
The output from the retrieve stage is the three best matching cases extracted from the database, except when there are no cases in the database previously. When the database is empty, the retrieve stage exits and tells the retain stage to create a new case. However if there are cases in the database, the three best cases are passed on to the reuse stage.

4.3.3 Reuse

Reuse is the next stage. Its main task is to test if the best matching case and the new case are sufficiently similar to be used automatically as solution. It only has to to test the single best matching test, because if this case is not similar enough to be automatically approved, the two other retrieved cases neither are. In fact, the two other retrieved cases are always further from the new case in similarity value that the best matching retrieved case. Another threshold value is used to determine if this similarity value is good enough or if the user has to approve the solution manually. The algorithm used to determine similarity is algorithm 4.7.
If the value from algorithm 4.7 is lower than the predefined similarity threshold, the case is automatically approved and passed on directly to the retain stage. On the other hand, if the cases are found to not be sufficiently similar, the case has to be approved or

Algorithm 4.6 Comparing cases in the retrieve stage.

```
1: if there are no cases in the database then
2:    exit and create a new case.
3: else
4:    Solution ← buffer value
5:    A ← best value
6:    B ← second best value
7:    C ← third best value
8:    Anr ← number on best matching case
9:    Bnr ← number on second best matching case
10:   Cnr ← number on third best matching case
11:   A = 100000000000
12:   B = 100000000000
13:   C = 100000000000
14:   for j = 0 to j = number_of_cases do
15:      for k = 0 to k = number_of_features do
16:         Solution+ = (thenewcase.featurenr[k] − casenr[j].featurenr[k])^2 ∗
            casenr[j].weightnr[k]
17:      end for
18:      if Solution < A then
19:         C = B
20:         B = A
21:         A = Solution
22:         Cnr = Bnr
23:         Bnr = Anr
24:         Anr = oldCases[j].caseNr
25:      else if Solution < B then
26:         C = B
27:         B = Solution
28:         Cnr = Bnr
29:         Bnr = oldCases[j].caseNr
30:      else if Solution < C then
31:         C = Solution
32:         Cnr = oldCases[j].caseNr
33:      end if
34:      Solution = 0
35:   end for
36: end if
37: return Anr, Bnr, Cnr
```

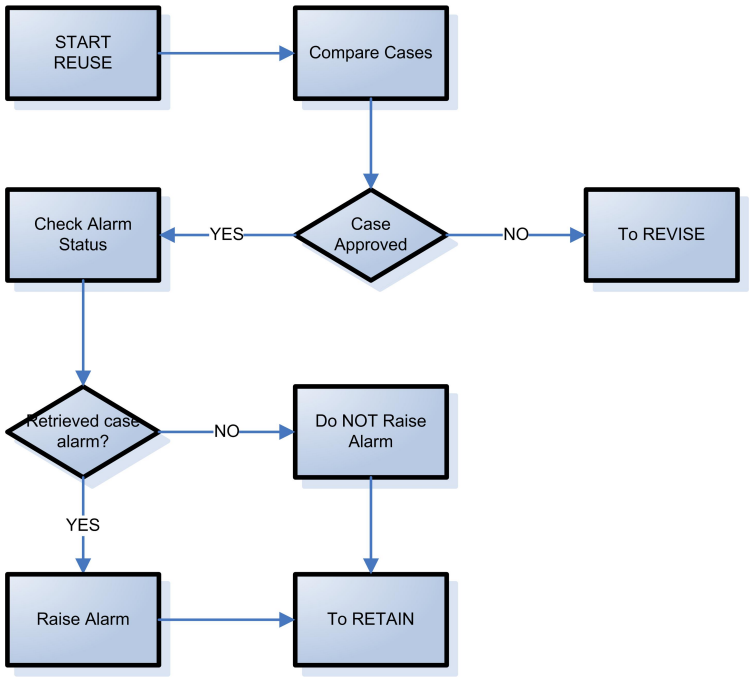

Figure 4.15: Flow diagram of the reuse stage.

Algorithm 4.7 Comparing cases in the reuse stage.

1: $A \leftarrow$ difference measure
2: **for** $k = 0$ to $k = number\_of\_features$ **do**
3: $\quad A+ = featurenr[k]^2$
4: **end for**
5: $A = \sqrt{A}$
6: **return** A

disapproved manually by a user, and is therefore passed on to the revise stage together with the two other retrieved cases.

4.3.4 Revise

In the revise stage the main task is to have the user manually approve or disapprove the retrieved cases. The revise stage is only invoked if the best retrieved case was rejected by the reuse stage. During this stage the user will get a textual description of the retrieved cases. At first the user has to deal with the best matching case. The user will then have two choices. The best retrieved case can be approved as the solution case for the new case, or it can be rejected. If the user approves the retrieved case, he will also be prompted to tell if the new case represents an alarm case. This info is only intended as a control to see if the approved case has the same alarm status as the retrieved case. If this check passes, the retrieved case is sent to the retain stage to update the information about the case. The user also has the option to disapprove the best retrieved case. If this happens, the same procedure is started with the second best matching case. If this case also is rejected, it moves on to the third best matching case.

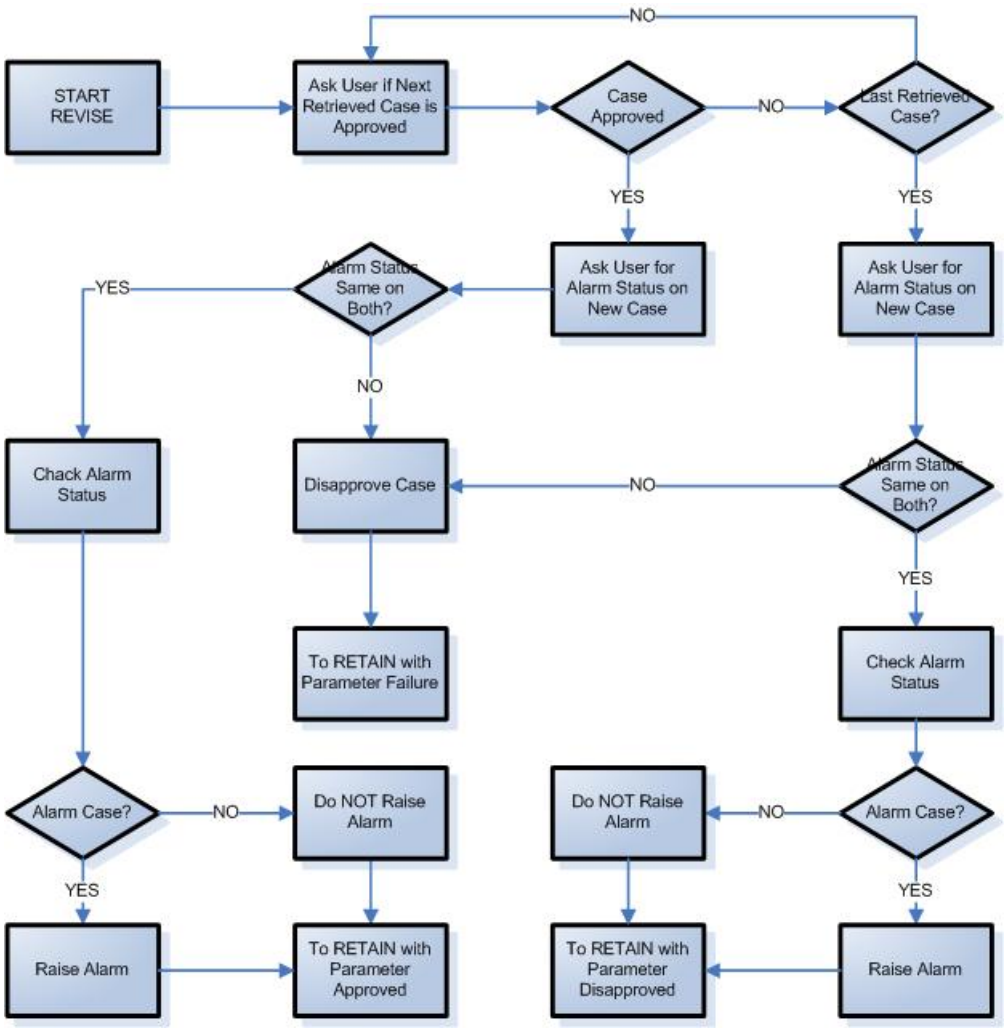

Figure 4.16: Flow diagram of the revise stage.

If all three cases are rejected, a new case will be created. The user is then prompted for a description of the new case and the alarm status. A new case is created and the system moves on to the retain stage to update information about the wrongly retrieved cases.

4.3.5 Retain

The last stage of the CBR module is retain. This is where the system gets updated according to what happened during the previous stages. A case can enter this stage in several different ways. The first, and least frequent way is if there are no cases in the database. When this occurs, the retrieve stage only have to make a new case. A second way is when there are cases in the database, but the new case did not resemble any of these cases. This situation also leads to the creation of a new case, but in addition the weights of the retrieved cases are updated. The third alternative is when on of the retrieved cases was approved as solution for the new case. Now there is no need to

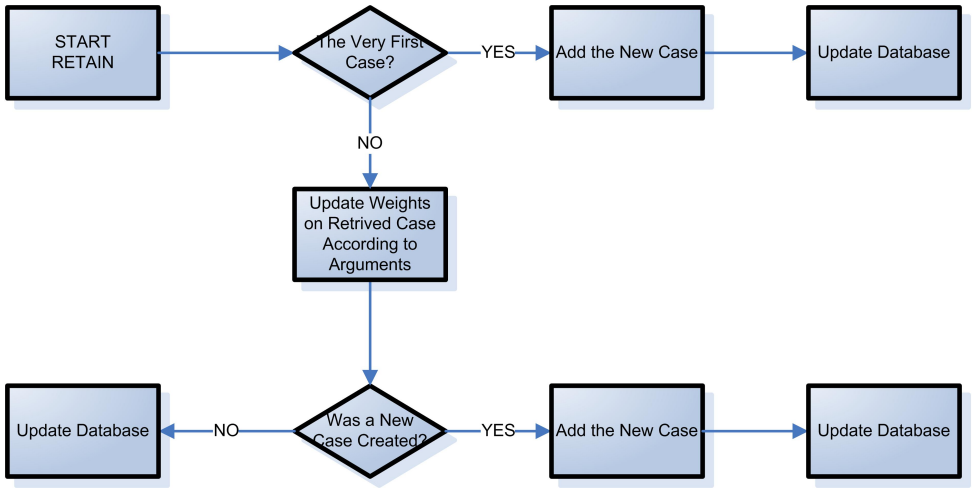

Figure 4.17: Flow diagram of the retain stage.

create a new case, and the only thing that happens is that the weights of the retrieved cases are updated. All three alternatives finishes with an update of the database.

Updating the Weights

The task of updating the weights are an essential part of the system. Essential because this is what gives the system the ability to learn. After a new case is passed through the system, the weights of the retrieved case are modified. The idea is that if the retrieved case was accepted as the solution case for the new case, features that were similar in the two cases are the most important ones. These features should therefore count more than the other features in this exact case. By strengthening the weights on the important features and weakening the weights on the others, the case gets more specific and the chance that the system retrieves the correct case next time is increased. On the other hand, if the retrieved case is rejected by the user, the weights are updated in the opposite way. This is done because the system chose the wrong case, and if the weights are updated the opposite way, there is less chance of doing this mistake again.

Algorithm 4.8 shows how the weights of the retrieved case are updated. Several variables are used to control what happens to the weights in different circumstances. The threshold value used is a ratio between the feature in the new and in the retrieved case. This is the value that decides if the exact feature is important for the case or not. Two percentage values are also used. One high that changes the feature in that is being compared, and one low that changes the other features in the opposite direction to compensate. Exactly the same happens for all features. When this is done, the features are scaled back to proportion, which means that the sum of all weights are the same after update as before. This is done because after very many runs, the weights can become very large, and that can make the system taking wrong decisions later.

Algorithm 4.8 Update weight vector.

```
1: A ← read feature vector from new case
2: B ← read feature vector from retrieved case
3: WB ← read weight vector from retrieved case
4: Setthreshold
5: Sethighpercentage
6: Setlowpercentage
7: for i = 0 to length(A) do
8:    if abs(A[i] − B[i]) < threshold then
9:       for j = 0 to length(WB) do
10:         if j == i then
11:            if Case Accepted as Solution then
12:               increase WB[i] by a predetermined high percentage
13:            else
14:               decrease WB[i] by a predetermined high percentage
15:            end if
16:         else
17:            if Case Accepted as Solution then
18:               decrease WB[i] by a predetermined low percentage
19:            else
20:               increase WB[i] by a predetermined low percentage
21:            end if
22:         end if
23:      end for
24:   else
25:      for j = 0 to length(WB) do
26:         if j == i then
27:            if Case Accepted as Solution then
28:               decrease WB[i] by a predetermined high percentage
29:            else
30:               increase WB[i] by a predetermined high percentage
31:            end if
32:         else
33:            if Case Accepted as Solution then
34:               increase WB[i] by a predetermined low percentage
35:            else
36:               decrease WB[i] by a predetermined low percentage
37:            end if
38:         end if
39:      end for
40:   end if
41: end for
42: Adjust weights so that the total weight sum is same as before
```

Chapter 5

Testing and results

This chapter gives a review of the tests done with the system.

5.1 Goals of testing

The main idea behind these tests is to see if a CBR module can be used to better the performance of an automatic video surveillance system. Basically this means the tests should locate the scenarios that represent false alarms and distinguish these from those that represents alarms. To check if this is accomplished, three goals are made:

- Reliability in the system
- Minimization of wrong solutions selected
- System has the ability to learn

These three goals has to be accomplished to conclude that the system is working well. The first goal says that the system has to be reliable. Reliability is a very important factor in all computer systems [25]. When a system is said to be reliable, it implies that the system has to perform well over time. It should always produce the same results to the same inputs. This can be tested by running the same tests several times, and check if the results are the same. Minimization of wrong solutions selected is the basic criteria for the success of this system. Testing with different testcases, should give a good answer to this goal. The last important piece of the system is that is has to have the ability to learn. This means that the system should learn from each case, and perform better next time. This can be tested by manipulating user inputs to try to make the system make different choices.

5.2 Procedure of testing

To test the system and to really see what happens on every run, a system log is created with every run. This log contains a written part of everything that happens during a run, as well as images saved at every stage. The written log provides the user with a thorough understanding of the system, as each step gets documented. To fully understand the system, viewing images as they evolve with each processing step is an

advantage. When the system is in full working order it is supposed to have two images as input, and by using image processing and CBR, figure out what to do. Testing should be performed in the same manner. During testing the system gets two input images, a reference image and an alarm image. Using these two images, the system should figure out which case to use as solution.

5.2.1 Selection of test data

Selection of what kind of data to be used for system testing is very important. The data needs to be varied and it needs to represent many different thinkable situations. [26]

The Reference Image

The reference image (figure 5.1) used in these tests is an image from a regular street in a town. This particular image is selected because it has many different elements in it. The scene has several cars and buildings in it. It also include a large three and a shadow from a building outside the image borders. By selecting such an image, many aspects can be tested at the same time which is a great advantage.

Figure 5.1: The reference image used for testing.

The Testcases

The testcases are shown in figures 5.2 to figure 5.7. These testcases are variations of the reference image with different properties. Some of them have been tampered with to create different natural scenes like snow and changing lighting conditions.

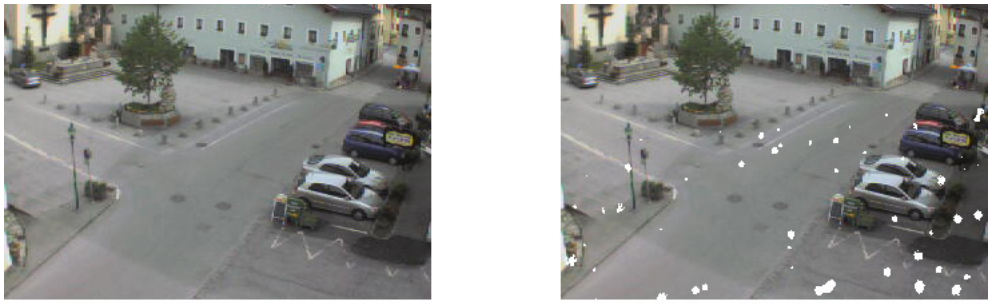

Figure 5.2: (1a) Testcase 1. (1b) Testcase 1 with snow.

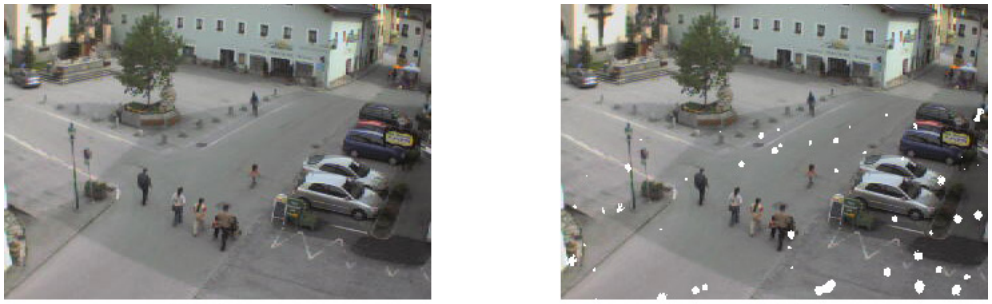

Figure 5.3: (2a) Testcase 2. (2b) Testcase 2 with snow.

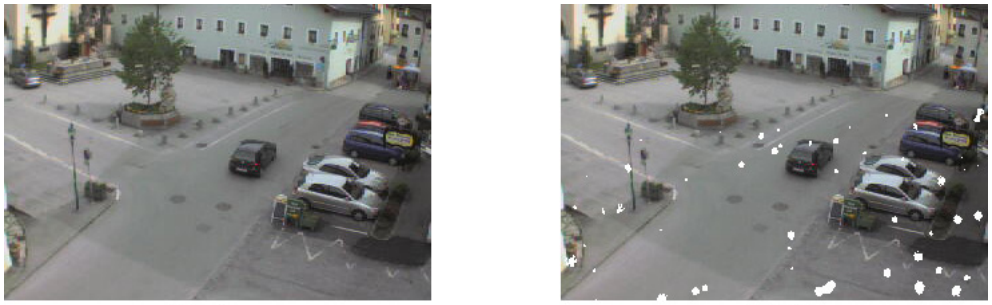

Figure 5.4: (3a) Testcase 3. (3b) Testcase 3 with snow.

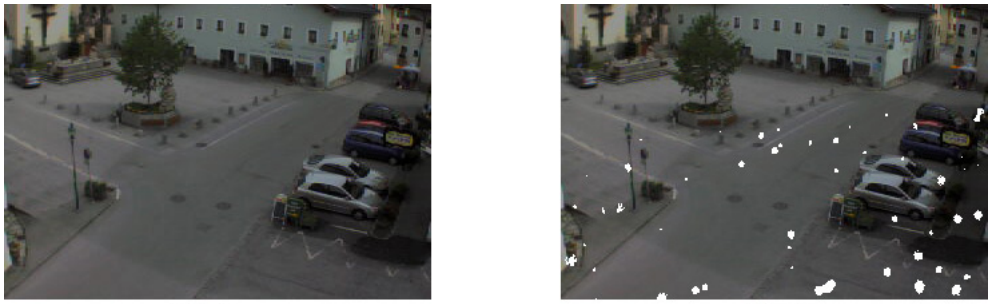

Figure 5.5: (4a) Testcase 4. (4b) Testcase 4 with snow.

Figure 5.6: (5a) Testcase 5. (5b) Testcase 5 with snow.

Figure 5.7: (6a) Testcase 6. (6b) Testcase 6 with snow.

5.2.2 Filling up the database

Before the testing can begin, there has to be some cases in the database. To fill the database the system is run as usual with cases representing different situations. For this particular set of tests, the database is filled with four different cases. These four different cases represents enough situations to put each testcase into at least one of them. The selected cases are:

- A case where nothing at all happens
- A case where there are people walking in the street
- A case where there is a car driving in the street
- A case where there is a lot of snow in the image

These four cases acts as containers during testing. Each one of the tests should fall into one of these containers. A more detailed description is shown in table 5.1.

Cases inserted in the database before testing			
Case:	**Description:**	**Image:**	**Raises alarm:**
1	Nothing happens	1a	No
2	People are walking in the street	2a	No
3	There is a car driving in the street	3a	Yes
4	There is only snow in the image	1b	No

Table 5.1: Cases inserted in the database.

5.3 Results

Test 1

Test 1 (table A.1) includes an alarm image that is much darker than the reference image. The alarm input image used in this test is image 4a (figure 5.5) This situation could occur if heavy clouds or something is blocking light from entering the scene. The system automatically approved the case where nothing happened as a solution, which is the correct solution. This implies that the system is able to deal with changes in lighting conditions, especially when changes are the same throughout the whole image.

Test 2

Test 2 (table A.2) is very similar to test 1. It also includes an alarm image that is much darker than the reference image. Test 2 uses image 5a (figure 5.6) as input. In addition to the change in lighting conditions, this image also includes some people that are walking in the street. This time, the system did not automatically approve a solution case. It suggested the case where people are walking in the street as best solution. This was the correct solution, and was approved by the user. This test further implies that the system is able to deal with changes in lighting conditions.

Test 3

Test 3 (table A.3) includes an alarm image that is much brighter than the reference image. The image used in test 3 is image 6a (figure 5.7). In addition to the change in lighting conditions, this image also includes a car in the street. The system did not automatically approve a solution case, but it suggested the case where there is a car in the street as best solution. This was the correct solution, and was approved by the user. This test further strengthens the assumption that the system is able to deal with changes in lighting conditions.

Test 4

Test 4 (table A.4) uses image 2b (figure 5.3). This is the same image as 2a, only with added snow. This means that this image includes some people walking in the street as well as a lot of snow. This time the system automatically approved the solution case, which was the case were people were walking in the street. This was the correct solution. The outcome of this test shows that the system did not care about the snow, but was able to locate the important features of the image, and choose the correct solution case based on these.

Test 5

Test 5 (table A.5) uses image 3b (figure 5.4). This is the same image as 3a, only with added snow. This means that this image includes a car in the street as well as a lot of snow. This time the system din not automatically approve the solution case. However, the system chose the correct solution case as best case. The retrieved case was the one with the car in the street. This was the correct solution. The outcome of this test shows, as test 4, that the system did not care about the snow.

Test 6

Test 6 (table A.6) uses image 4b (figure 5.5). This is the same image as 3a, only with added snow. This means that this image includes a car in the street as well as a lot of snow. This time the system din not automatically approve the solution case. However, the system chose the correct solution case as best case. The retrieved case was the one with the car in the street. This was the correct solution. The outcome of this test shows, as test 4 and 5, that the system did not care about the snow.

Test 7

Test 7 (table A.7) is very similar to test 2. It also includes an alarm image that is much darker than the reference image. Test 7 uses image 5b (figure 5.6) as input. In addition to the change in lighting conditions and some people that are walking in the street, this image also contains a lot of snow. This time, the system did not automatically approve a solution case. It suggested the case where people are walking in the street as best solution. This was the correct solution, and was approved by the user. This test further suggest that the system is able to disregard the snow. The alarm situation is now much

darker and many new objects (snow) are presented in the scene. It still chooses the correct solution case, which is very good.

Test 8

Test 8 (table A.8) includes an alarm image that is much brighter than the reference image. The image used in test 8 is image 6b (figure 5.7). In addition to the change in lighting conditions, this image also includes a car in the street as well as a lot of snow. The system did not automatically approve a solution case, but it suggested the case where there is a car in the street as best solution. This was the correct solution, and was approved by the user.

A more extensive review of the tests together with logs can be seen in appendix A.

5.4 Is the system reliable?

To figure out if the system shows reliability in the test results, the same tests were run over and over again in circles. Each test was performed four times. Table 5.2 shows the outcome of this test.

Reliability testing				
Test Nr:	**Run 1:**	**Run 2:**	**Run 3:**	**Run 4:**
1	1	1	1	1
2	2	2	2	2
3	2	2	2	2
4	1	1	1	1
5	2	2	2	2
6	1	1	1	1
7	2	2	3	2
8	2	2	2	2

Table codes:
1: Correct case was retrieved and approved by the system
2: Correct case was retrieved and approved by the user
3: Wrong case was retrieved

Table 5.2: Reliability testing.

As seen in table 5.2 the eight tests was conducted four times each. The table shows that the system is very reliable. However, one of the tests produced a wrong result. During the third run of test 7, the system retrieved the wrong case as best solution. To be exact, it picked out the case with only snow in the image instead of the case with people walking in the street. To try to explain this mistake made by the system, the difference images from the two cases and test 7 is shown in figure 5.8. The figure shows three difference images. Image (b) is the difference image from test 7. Image (a) is the difference image from the case with people walking in the street. This was the case that should have been selected as the best solution. Image (c) is the difference image from the case with only snow. This was the case that the system selected as

best case. As seen, difference images (b) and (c) are quite similar. In the first two runs the correct case was selected, which is represented by (a). A possible solution is that during the first two runs, the system has been learning that an important feature for the case with only snow is the number of objects in the image. The number of objects in (b) and (c) are very much the same, so this could have caused the failure.

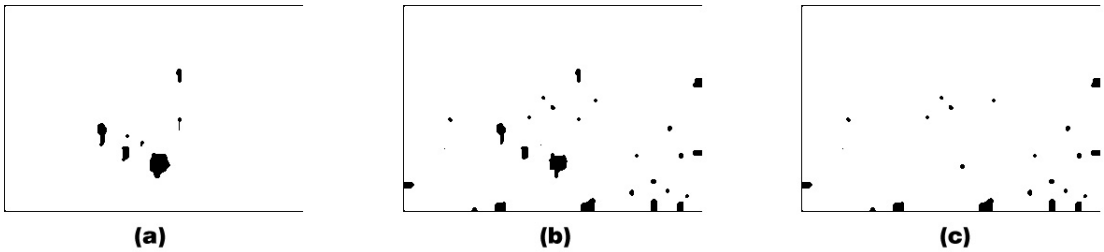

Figure 5.8: (a) Difference image from the case with people in the street. (b) Difference image from test 7. (c) Difference image from the case with only snow.

5.5 Does the system minimize number of wrong solutions selected?

The overall test results shows that the system really is minimizing the number of wrong solutions selected. During the testing, there was only one wrong solution selected. Except this one wrong selection, the system chose the correct solutions every single time.

5.6 Is the System Learning?

The tests up until now have been successful. This really that the image processing module is doing its job perfectly. But it does not say much about the CBR module, except the fact that it locates the right case based on a comparison between the cases. To really test the CBR module, the tests has to be manipulated. To accomplish this, image 7a (5.9) is used. This image shows the scene with two cars and a lot of snow.

Figure 5.9: (7a) Testcase 7.

The first run of this test can be seen in table A.9. In this run, the system selected the case where there is a car driving in the street. This is the correct solution case, but now the CBR module should be put to the test. To achieve this, the system is told that the case where people are walking in the street is the correct solution case. This case was during the first run picked out as the second best matching case. In the second run (table A.10 the same case was chosen as the best solution. The case with the people walking in street was again chosen as the second best case, and the system was told that this was the solution case. Now, in the third run (table A.11), the goal of this test was accomplished. The case with people walking in the street was selected as the best matching case by the system. This implies that the system now have learned to recognize this situation as people walking in the street. Of course, this is the wrong solution, but the system has learned that this is the correct solution from the user inputs during each run. This test clearly shows that the system is able to learn.

Chapter 6

Discussion and future work

The intended use of the system is to exploit the functionality of CBR together with image processing techniques. This is achieved by implementation of a series of methods within both fields. The system has been tested for overall functionality and the results from these tests shows that the system is working well. However, as the system only has been tested at a high level, there are still several uncertainties that has to be addressed. Many of these uncertainties forms the basis for further work and further testing. The several issues are addressed under each module below.

6.1 Image processing module

The image processing module is used to extract relevant features from the alarm image. An important part of this module is how the objects are segmented. This system uses two images, a reference image and an alarm image, to locate the different objects in the image. This makes the system very dependent on the reference image. At this stage, there is no control over what the reference image contains. If the reference image itself contains passing objects, this could have very bad effects on the outcome of the segmentation. A further development of the system should therefore include a control check of the reference image to make sure there are none unusual objects in it. An advantage of taking this approach to segmentation is that it is very easy to locate the relevant objects from the alarm image. Only the objects of interest gets selected. Alternatives that only requires the alarm image could also be used for segmentation. Color- and texture-based segmentation is described in [27]. Such methods can also be included in the system alongside the reference image approach to have even better control on object shapes and sizes. Before the segmentation process there is also a pre-processing phase. Pre-processing is done in a fairly usual way, using median filtering. This approach seems to work fine. Noise and small objects are removed from the images. This system also incorporates the use of mathematical morphology to further remove the rest of the small objects in the images. Results shows that this is a good way to remove these objects. Morphology also fills holes in objects which is very good. The downside is that it often has a tendency to group objects. If two objects in the image are quite close together, there is a possibility that these two objects can be seen as only one object when morphology has done its job. But in this system, it really does not matter, because this happens with all such similar close objects. So if it

happens to a new case, the same thing would have happened to the cases already in the database. All in all the image processing module seem to do a good job of segmenting out objects.

6.1.1 Extracted features

The features that are selected for extraction from the images have great impact on the results in the system. This implementation only extracts features that are intensity-related and object-related. The most important features in this application is of course the information about the different objects located in the image. This information is restricted to sizes of the objects. Different kinds of object information can be color of objects and position of objects in the image. Colors of the different objects intuitively does not have any impact in this application. It really does not matter if the car found in the image is red or green. Based on this assumption color information is completely disregarded in this system. As for position of objects in the image, this can have an impact. An object in a specific position in the image can be a larger threat than the same object in a different position. This is an example of one feature that could be incorporated into the system to maybe improve the functionality. Feature extraction in general is widely discussed in [28] and in [29].

Considering the testing and the results from the testing, the features that are extracted seem to represent the scene in the image very nicely. But should there be added more features to each case, this will most likely serve as an improvement to the system, and enable the system to be even more precise in its case retrieving.

6.2 CBR module

The CBR module implemented in this paper uses a very common approach to CBR systems. Dividing the module into four submodules (retrieve, reuse, revise and retain) is a reasonable way to model the different tasks. The basic functions in this module works exactly as intended. However, there are several variables in the module that can have great impact on the results if modified. One of these is the threshold for automatically passing the reuse stage. Throughout the testing, the system kept selecting and retrieving the correct cases as solutions. But user interaction was often needed to manually approve the solutions. On the basis of the systems great ability to select correct solutions, this threshold can without a doubt be increased. Doing this means that the user does not have to approve the correct retrieved cases as often as during the testing. The risk by increasing this threshold is that the system has more of a chance at automatically approve a wrong case for whatever reason. But testing shows that this risk is quite minimal, so the threshold should be increased.

Another source of discussion is the values selected in the method that does the update of the weights of the cases. The issue here is most importantly how similar each feature has to be to increase its weight. Another threshold is used to determine this question. The value of this threshold is compared to the percentage change between each feature of the retrieved cases and the new case. The risk here is that the threshold can be set

too low or too high. If it is set too high, features that really are not that similar could be seen as important for the case and the weight is subsequently increased. Also, the system is at risk of choosing all features as important for the case. If this happens, there is no change whatsoever to the weight in the case. They stay the same as before, and the system will not have learned anything at all. On the other hand, if the weights are set too low, the opposite is true. This is just as bad for the system. A situation can occur where the selected case really is not very similar to the new case according to the features, but it is chosen as the correct solution by the user. This may lead to a situation where none of the features are similar enough to get its weight increased, and the system does not learn anything at all from this particular episode. The value of this threshold is very difficult to set correctly as differences vary from case to case. One way to overcome such problems is to implement a threshold that is not fixed, but can be changed from case to case. A solution can be to set the threshold just above the second most similar feature. If such a solution is implemented, at least two features will always get their weights increased. These are the two most similar features. This can be a source of further development of the system. This method also includes some variables which can be further discussed. These are the high and low percentages for the value of change in the feature weights. These variables has to be set at quite good values to ensure the good functionality of the system. Tuning these percentages to good values can be accomplished by further testing with different values.

6.2.1 The database

In this system the case database is stored as a plain text file. Although this works in this specific system, this is not an optimal solution. The reason it works fine in this system, is that the whole database is copied into the system at each startup. This makes it possible to manipulate the cases internally in the system, and the way the cases are stored between system runs is not that imperative. It is also a reasonably good solution as long as the database is quite small. However, in this system, the database is going to grow. New situations are going to occur, and this requires the creation of new cases. Fetching all cases from the database into the system will then be a major task, and much of the information copied into the system will be unnecessary to have there at all. A solution to this problem can be to store the cases in a typical database environment (SQL [30], Oracle [31] etc.). This would allow the system to perform queries into database, and only get the information needed at a given moment.

Another way to make the database better and more effective would be to have a smart sorting or indexing of the cases. This would allow to have links between the cases, and links between the different features. An advantage by having these links would be that searching the database for relevant cases can go faster. If a relevant case is found, the system can follow these links in some pre-defined way. This would certainly speed up the system. However, it is difficult to find a smart way to use such an indexing to improve the system.

6.2.2 Learning

A major part of this system is its ability to learn. And the important question in the learning field, is how well the system is learning? Especially, how well is the system learning using CBR compared to different machine learning techniques? The advantage of using CBR in this system is that the user of the system can assist in the training of the system. The user has the ability to manipulate the system to recognize certain situations as alarm situations. This is a kind of supervised learning [32]. In this particular system, this is the only form of learning that is possible. But supervised learning should be preferred in such a system, because this allows the system to be adapted for different usages. An alternative to using CBR would be to use neural networks [33] and [34]. This would allow for unsupervised learning as well and this could maybe improve a standardized system, but would not give the system the opportunity to be specialized.

The testing done on this system clearly shows that the system can be manipulated by the user. Or in other words, the user can teach the system how to respond to different situations. Testing showed that it only took a couple of runs before the system was persuaded into thinking differently. Also when the system made an error, it quickly corrected itself on the next run according to the user input. This proves that the system also learns very fast.

6.3 Is CBR a successful approach?

The experience with using CBR together with image processing to create an image interpretation system has been positive. During testing it has been shown that CBR is capable of making the correct solutions.

One of the strong sides of CBR is that it can solve problems that only are partially understood. In an image taken from a video feed there will always be small differences. No two alarm situations will be exactly the same. A CBR system does not need to have an exact match to find a solution, it just chooses the best candidate, and this is often the correct solution. CBR is also good at learning implicit rules. That is, deciding what knowledge that is important for each case. This is made possible be the feature weights in each case. The user does not have to decide which features that are important for each case. The system figures this out itself. In some cases, a feature can be completely disregarded. Such a characteristic goes well with image interpretation. An example could be that there is a car in the image. If the image also contains a lot of snow, the knowledge of snow should be completely disregarded, because the important thing in that special case is the car. From testing, it is obvious that this is accomplished.

One main advantage, as previously mentioned, of using CBR is that the system can be formed by the user needs. There does not have to be pre-defined situations that always trigger alarms. The user can, by user input, decide how the system should respond to each separate scenario. This is made possible by the fact that supervised training is applied throughout the system. With a neural network it would have been difficult for a user to supervise this in a system.

6.4 Future Work

Many of the parameters discussed has to be better tuned for the system to reach optimal performance. More extensive testing with different parameters and different input scenarios should also be done. A more practical suggestion for future work would be to incorporate these modules into TrollEye. Taking this action will allow the system to be tested in real-time and to really see the opportunities in using CBR in a surveillance system. Another option to the system would be to include a priori knowledge in the form of a knowledge base. Some common objects can be implemented into the system with shape and size. By combining this knowledge with the reasoning, the system can be improved to make even better choices.

Chapter 7

Conclusion

During this thesis the opportunities that lies in using CBR in an automatic video surveillance system has been explored. The theory behind it has been given and a working system has been made. The system has undergone thorough testing and results have been documented.

Several image processing techniques have been used to perform the first tasks in the system. This first task was prepare the reference image and the image that created the alarm. When this was accomplished, objects were located and features was extracted. All this was done to create the basis for the CBR module, which used this image information as a case and found a solution to it. This thesis has shown that CBR makes it possible for a computer system to distinguish certain situations from others. Situations where alarms should have been raised have been put into containers that represented alarms. And situations that represented false alarms have not been put into these same containers.

The main objective of this thesis was to find out if CBR performs well in an automatic video surveillance system. This thesis has showed that CBR is a very successful approach to image interpretation. The fact that it is very natural to create descriptions from images assists in this explanation. Whenever something can be described to the fully, cases can be created and reasoned about. This is the main strength of CBR. Another strength of CBR systems has been their ability to learn and to learn fast. Testing the system has showed that CBR is very capable of learning from its mistakes and correct these mistakes.

The system that is implemented works well as a platform for testing the significance and usefulness of CBR in the video interpretation domain. As the system is not yet implemented to work with a camera feed in real time, it is difficult to see the full possibilities of CBR. But the main achievement of this thesis, is that it has been showed that a computer system using CBR can determine if an alarm sounded is a false or a real alarm. The system is able to determine if there is a car in between a lot of snow, or if there is only a lot of snow. Although there is a long way to go before a computer system on its own can make perfect decisions each time, this thesis has provided a base and a start, and showed that such systems can be made.

Bibliography

[1] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt, and Lambert Wixson1. A system for video surveillance and monitoring, 1999.

[2] Robert Fisher, Simon Perkins, Ashley Walker, and Erik Wolfart. Mathematical morphology. http://www.homepages.informatics.ed.ac.uk/rbf/HIPR2/morops.htm.

[3] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications, 7(1)*, pages 39–59, 1994.

[4] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. PWS Publishing, 2 edition, 1999.

[5] A. Lipton, H. Fujiyoshi, and R. Patil. Moving target classification and tracking from real-time video, 1998.

[6] Nikos Paragios and Rachid Deriche. Geodesic active contours and level sets for the detection and tracking of moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):266–280, 2000.

[7] W. Zhao, R. Chellappa, A. Rosenfeld, and P. Phillips. Face recognition: A literature survey, 2000.

[8] N. Oliver and A. Pentland. Graphical models for driver behavior recognition in a smartcar, 2000.

[9] A. Jonathan Howell and Hilary Buxton. Active vision techniques for visually mediated interaction. *Image Vision Comput.*, 20(12):861–871, 2002.

[10] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Comput.*, 3(2):246–257, 1991.

[11] Nathanaël Rota and Monique Thonnat. Activity recognition from video sequences using declarative models. pages 673–680.

[12] M. Ghallab. Representation, on-line recognition and learning, 1996.

[13] N. Chleq and M. Thonnat. Realtime image sequence interpretation for video-surveillance applications. *ICIP*, B:801–804.

[14] V. Vu, F. Bremond, and M. Thonnat. Automatic video interpretation: A recognition algorithm for temporal scenarios based on pre-compiled scenario models, 2003.

[15] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[16] Ronei Marcos De. Anais do ix sibgrapi (1996) 357-358 image classification using mathematical morphology.

[17] Phyllis Koton. Reasoning about evidence in causal explanations. In *AAAI*, pages 256–263, 1988.

[18] William J. Long. Medical diagnosis using a probabilistic causal network. *Applied Artificial Intelligence*, 3:367–383, 1989.

[19] K. Ashley S. Brüningshouse. Combining case-based and model-based reasoning for predicting the outcome of legal cases. *ICCBR 2003. LNAI 2689*, pages 246–260, 2003.

[20] A. Aamodt. Knowledge acquisition and learning by experience - the role of casespecific knowledge, 1995.

[21] John D. Hastings L. Karl Branting and Jeffrey A. Lockwood. Integrating cases and models for prediction in biological systems. *AI Applications 11(1)*, pages 29–48, 1997.

[22] Petra Perner. Why case-based reasoning is attractive for image processing. *D. Aha and I. Watson (Eds.), Case-Based Reasoning Research and Development, Springer Verlag 2001, Inai 2080*, pages 27–44, 2001.

[23] R. Dencker. Image processing library. http://www.mip.sdu.dk/ipl98/mainipl.htm.

[24] Kh. Manglem Singh and Prabin K. Bora. Adaptive rank-ordered mean filter for removal of impulse noise from images.

[25] Jean-Claude Laprie and Karama Kanoun. Software reliability and system reliability. pages 27–69, 1996.

[26] John B. Goodenough and Susan L. Gerhart. Toward a theory of test data selection. *SIGPLAN Not.*, 10(6):493–510, 1975.

[27] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik. Color- and texture-based image segmentation using EM and its application to content-based image retrieval. In *Proceedings of the Sixth International Conference on Computer Vision*, 1998.

[28] Y. Bresler G. Harikumar. Feature extraction techniques for exploratory visualization of vector-valued imagery. *Image Processing, IEEE Transactions on Volume 5, Issue 9*, pages 1324–1334, 1996.

[29] R. Chellappa K. Etemad. Separability-based multiscale basis selection and feature extraction for signal and image classification. *Image Processing, IEEE Transactions on Volume 7, Issue 10*, pages 1453–1465, 1998.

[30] M. J. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[31] Sam R. Alapati and Martin Reid. *Expert Oracle 9i Database Administration*. APress L. P., 2003.

[32] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[33] Parekh R. G., Yang J., and Honavar V. Constructive Neural Network Learning Algorithms for Multi-Category Pattern Classification. Technical report, Department of Computer Science, Iowa State University, Ames, Iowa, 1995.

[34] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.

Appendix A

Test logs

Test 1	
Image:	4a
Description:	In this scenario there are no objects in the scene. The image has been darkened to represent clouds in front of the sun or late in the evening.
Supposed solution case:	Case 1: Nothing happens.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 1: nothing happens. The second best case is case 4: only snow in the image. The third best case is case 2: people are walking in the street. The cases are similar enough to pass reuse. The retrieved case is approved as solution and does not represent an alarm. ALARM IS NOT RAISED! Updating weights with oldcase: 1 and args 2. Updating the database.
Test outcome:	The new case was similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.1: Results from test 1.

Test 2	
Image:	5a
Description:	In this scenario there are some people walking in the street. The image has been darkened to represent clouds in front of the sun or late in the evening.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 2: people are walking in the street. The second best case is case 4: only snow in the image. The third best case is case 1: nothing happens. The cases are not similar enough to pass reuse. Case 2 approved by user: yes. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.2: Results from test 2.

Test 3	
Image:	6a
Description:	In this scenario there is a car driving in the street. The image has been brightened to represent bright light.
Supposed solution case:	Case 3: There is a car in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 3: there is a car in the street. The second best case is case 2: people are walking in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 3 approved by user: yes. Updating weights with oldcase: 3 and args 2. Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.3: Results from test 3.

Test 4	
Image:	2b
Description:	In this scenario there are some people walking in the street. There is also a lot of snow in the air.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 2: people are walking in the street. The second best case is case 4: only snow in the image. The third best case is case 1: nothing happens. The cases are similar enough to pass reuse. The retrieved case is approved as solution and does not represent an alarm. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Updating the database.
Test outcome:	The new case was similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.4: Results from test 4.

Test 5	
Image:	3b
Description:	In this scenario there is a car driving in the street. There is also a lot of snow in the air.
Supposed solution case:	Case 3: There is a car in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 3: there is a car in the street. The second best case is case 2: people are walking in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 3 approved by user: yes. Updating weights with oldcase: 3 and args 2. Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.5: Results from test 5.

Test 6	
Image:	4b
Description:	In this scenario there are no objects in the scene. The image has been darkened to represent clouds in front of the sun or late in the evening. There is also a lot of snow in the air.
Supposed solution case:	Case 4: Only snow in the image.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 4: only snow in the image. The second best case is case 1: nothing happens. The third best case is case 2: people are walking in the street. The cases are similar enough to pass reuse. The retrieved case is approved as solution and does not represent an alarm. ALARM IS NOT RAISED! Updating weights with oldcase: 4 and args 2. Updating the database.
Test outcome:	The new case was similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.6: Results from test 6.

Test 7	
Image:	5b
Description:	In this scenario there are some people walking in the street. The image has been darkened to represent clouds in front of the sun or late in the evening. There is also a lot of snow in the image.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 2: people are walking in the street. The second best case is case 4: only snow in the image. The third best case is case 1: nothing happens. The cases are not similar enough to pass reuse. Case 2 approved by user: yes. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.7: Results from test 7.

Test 8	
Image:	6b
Description:	In this scenario there is a car driving in the street. The image has been brightened to represent bright light. There is also a lot of snow in the air.
Supposed solution case:	Case 3: There is a car in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 3: there is a car in the street. The second best case is case 2: people are walking in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 3 approved by user: yes. Updating weights with oldcase: 3 and args 2. Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.8: Results from test 8.

Test 9 (First run)	
Image:	7a
Description:	In this scenario there are two cars driving in the street. There is also a lot of snow in the air.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 3: there is a car in the street. The second best case is case 2: people are walking in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 3 approved by user: no. Case 2 approved by user: yes. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Updating weights with oldcase: 3 and args 1. Updating the database.
Test outcome:	The wrong case was selected. Test was not successful.

Table A.9: Results from test 9. (first run)

Test 9 (Second run)	
Image:	7a
Description:	In this scenario there are two cars driving in the street. There is also a lot of snow in the air.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 3: there is a car in the street. The second best case is case 2: people are walking in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 3 approved by user: no. Case 2 approved by user: yes. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Updating weights with oldcase: 3 and args 1. Updating the database.
Test outcome:	The wrong case was selected. Test was not successful.

Table A.10: Results from test 9. (second run)

Test 9 (Third run)	
Image:	7a
Description:	In this scenario there are two cars driving in the street. There is also a lot of snow in the air.
Supposed solution case:	Case 2: People are walking in the street.
Log:	Done reading cases from DB. Comparing cases in the retrieve stage. The best case is case 2: people are walking in the street. The second best case is case 3: there is a car in the street. The third best case is case 4: only snow in the image. The cases are not similar enough to pass reuse. Case 2 approved by user: yes. ALARM IS NOT RAISED! Updating weights with oldcase: 2 and args 2. Case text: people are walking in the street Updating the database.
Test outcome:	The new case was not similar enough to the retrieved case to automatically be approved by the system. The correct container was chosen. Test was successful.

Table A.11: Results from test 9. (third run)