NTNU – Trondheim
Norwegian University of
Science and Technology

Controlling a Signal-regulated Pedestrian Crossing using Case-based Reasoning

Øyvind Shahin Berntsen Kheradmandi
Fredrick Strøm

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Agnar Aamodt, IDI

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

Problem description

The aim of this project is to investigate whether today's signal controlled pedestrian crossings can be improved by using methods within the field of artificial intelligence, and particularly the use of Case-based reasoning. This includes identifying what information can be extracted and used to improve safety and efficiency, for both motorists and pedestrians. This will be the basis of an implementation of an experimental system for controlling a signal controlled pedestrian crossing. The implementation of such a system will also consist of integrating the Intention-based Sliding Doors system, created by John Sverre Solem in an earlier MSc project at IDI, to interpret the intention of pedestrians.

The system will be experimentally tested, to uncover its potential, and it will be compared with today's systems to reveal whether it is more favorable. A study should also be performed to identify if there are any information sources not available at the time that can be used to improve the systems performance in the future. The project will be completed in collaboration with The Norwegian Public Roads Administration.

Assignment given: 11. January 2012

Supervisor: Agnar Aamodt, IDI

Co-supervisors: Jo Skjermo, Norwegian Public Roads Administration

Anders Kofod-Petersen, IDI

Abstract

The traffic domain, and in particular the domain of traffic control, is a highly complex and uncertain domain. A large network of roads, signal controlling systems, vehicles, pedestrians and other traffic units makes the domain intractable. There are great amounts of data available from different parts of traffic, thus there is a need for a method that can take advantage of this data in a systematical manner.

In this thesis, we present a prototype Case-based Reasoning (CBR) system which purpose is to execute traffic at a signal controlled pedestrian crossing. The system uses pedestrian- and vehicle data to take decisions in real-time. The system is created as an OSGI bundle and uses the CVIS (Cooperative Vehicle-Infrastructure System) framework to enable communication with other traffic systems and traffic units. myCBR is used as a framework for making the process of retrieving and reusing cases easier. Experts from Norwegian Public Roads Administration were an important resource in defining the structure of the cases and for filling the case base with useful cases. Pedestrian data is obtained by using a Kinect sensor, and the Intention-based Sliding Doors system created by Solem, a previous MSc at our group, is integrated for interpreting the intention of pedestrians at the crossing. Vehicle data is obtained by using simulation software called SCANeR Studio.

The results of the project showed that the CBR system adapted to the current traffic situation, and that correct cases were retrieved. These tests were performed in a limited test environment, and to evaluate the system properly, tests in a real environment is necessary.

Sammendrag

Trafikk domenet, og spesielt trafikkkontroll, er et komplekst domene, med mange usikkerheter. Store nettverk av veier, signal-regulerte systemer, kjøretøy, fotgjenger og andre trafikkenheter gjør domenet vanskelig å håndtere. Mengdene data som er tilgjengelig fra de forskjellige delene av trafikken er store, og det trengs derfor metoder som kan dra nytte av disse dataene på en systematisk måte.

I denne masteroppgaven presenterer vi et eksperimentelt system som tar i bruk Case-basert Resonnering (CBR) for å utføre trafikk i et signal-regulert gangfelt. Systemet bruker fotgjenger- og kjøretøy data for å ta avgjørelser i sanntid. Systemet er laget som en OSGi bundle og bruker CVIS (Cooperative Vehicle-Infrastructure System) rammeverket for å muliggjøre kommunikasjon med andre trafikksystemer og trafikkenheter. myCBR er blitt brukt som rammeverk for å gjøre prosessen med å hente ut og gjenbruke caser, enklere. Eksperter ved Statens vegvesen har vært en viktig ressurs i å definere casenes struktur og for å fylle opp case basen med nyttige caser. Fotgjenger data er blitt innhentet ved å bruke en Kinect sensor, og systemet "Intention-based Sliding Doors" laget av Solem, en tidligere MSc i vår gruppe, har blitt integrert for å tolke intensjonen til fotgjengere ved gangfeltet. Data om kjøretøy har blitt innhentet ved å bruke en trafikksimuleringsprogramvare kalt SCANeR Studio.

Resultatene av prosjektet viste at CBR systemet tilpasset seg den nåværende trafikksituasjonen, og hentet ut riktige caser. Testene ble utført i et begrenset testmiljø og for å evaluere systemet grundig er det nødvendig å teste systemet i et mer realistisk miljø.

Preface

This Master thesis constitutes the final work of our 5 year Master of Science studies in Computer Science. The work has been carried out at the Department of Computer and Information Science at the University of Science and Technology (NTNU), in cooperation with the Norwegian Public Roads Administration (NPRA). The work started in the autumn of 2011 with a specialization project.

We would like to thank our main supervisor Agnar Aamodt for his valuable assistance throughout the project. It has been an honor working with him, and hopefully gaining some of his knowledge. Without it, this project would not have been as valuable to us as it has been.

We would also like to thank the NPRA for facilitating our work on this project, and especially our co-supervisor Jo Skjermo. He has been of great value for understanding the traffic domain and also in creating and testing our prototype system. We would also like to thank our other co-supervisor, Anders Kofod-Petersen, for giving us great ideas along the way and for giving us aid in integrating the Sliding Doors project. We also wish to thank Kristin Kråkenes and Helge Stabursvik at the NPRA for valuable input during the project.

Last, but not least, we would like to thank our family and friends for all the support they have given during our years of studying.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we want to study the use of Case-based Reasoning (CBR) in the domain of traffic control, particularly in controlling a single signal controlled pedestrian crossing. It is a complex domain, with large amounts of available data. By using different detectors for acquiring information about pedestrians and vehicles, we use solutions to earlier experienced situations for changing the signals in the crossing. The amount of available data is much greater when it comes to vehicles than pedestrians. For example, there are no detectors for monitoring the movement or amount of pedestrians. As a result of this, we have integrated a system created by Solem[3] for creating an intelligent sliding door.

The intelligent sliding door uses artificial intelligent methods to interpret whether an individual approaching the door has the intention to enter, or to just walk by. This can in many ways be compared to the task of inferring the intention of a pedestrian approaching a signal controlled pedestrian crossing. It is reasonable to believe that such an interpretation can speed up the process of performing traffic control, since the system does not need to wait for the pedestrians to execute the signaling. It can also provide the system with the number of pedestrians that wish to cross the road. This is an important value, since it can give indications on how important it is to let the pedestrians pass, and how long transition time they should get. In addition to this, the information can contribute in replacing the push button that pedestrians use to signal the system that they wish to cross the road, since the system makes the discovery on its own.

We will also investigate if there is any other knowledge that can be utilized, to make more intelligent decisions. This includes uncovering what vehicle related information can be utilized to control a signal controlled pedestrian crossing, like the amount of traffic

or the speed of vehicles. We will also look at the possibility to classify the type of pedestrians that wish to cross the road, to make it possible to customize the crossing for slower pedestrians.

A prototype system has been developed to demonstrate the potential of CBR in this domain. The system uses the number of pedestrians that have the intention to cross the road, along with other important traffic features, to retrieve cases that represent earlier experienced situations, and uses the solutions from these situations to solve new situations.

In the next section, we present the goals of this project. Section 1.2 give some the motivational factors of this project, before an overview of the report is presented in section 1.3.

1.1 Goals

1. Study to what extent Case-based reasoning is a suitable technology in the domain of traffic control, and particularly in controlling a single pedestrian crossing.

2. Create an experimental CBR system for controlling traffic in a signal controlled pedestrian crossing that can perform better than today's systems in terms of:

   (a) Efficiency

   (b) Safety

   (c) User friendliness

1.2 Motivation

The traffic domain, and in particular the domain of traffic control, is a highly complex and uncertain domain. A large network of roads, signal controlling systems, vehicles, pedestrians and other traffic units makes the domain intractable. There are great amounts of data available from the different parts of traffic, thus there is a need for a method that can take advantage of this data in a systematical manner.

We want to uncover whether CBR is a method that can solve problems in the domain of traffic control, and particularly in controlling a single pedestrian crossing. Our theory

is that CBR can use earlier experienced situations in traffic to solve new problems, by looking at available data from sensors or similar sources. Compared to a similar method, rule-based reasoning, CBR can seem more applicable in a domain where it is hard for experts to define some rules cover the whole domain. It is reasonable to believe that this is the case in the traffic domain, since many situations can be similar to earlier experienced situations, but not necessarily the same. For example, if a normal pedestrian, in terms of speed and behavior, approaches a crossing, the system should give the pedestrian a normal transition time. But if the pedestrian approaches the crossing along with 10 of his normal friends, the pedestrians should get increased transition time, since they will probably need more time to cross. And if one of his friends is sitting in a wheelchair, they might need even longer transition time. But what if the traffic is high? Then maybe the transition should be shorter or it should be executed later, to make the traffic flow as good as possible. These ever-changing situations might be difficult to cover with rules, because it would require a very large number of rules, with a high amount of conditions.

Another thing that might make CBR more suitable is that it is easy to change the cases in the case base. Adding, updating or removing rules in a rule-based system can be a time-consuming process, since changing one rule can involve changing all rules that are dependent of the modified rule. In CBR, all cases are independent of each other, and therefore a new case can be learned, or manually added, without it changing the rest of the case base. This is an obvious advantage in a domain where changes can occur at any time. For example, if an intersection is changed, or the roads in the area close to the intersection are changed, then the traffic through the intersection will also change. In a CBR system, this can be handled by either adding cases to cope with the changes, or by letting the system itself learn how the changes has altered the traffic.

There are also motivational factors that come from the cooperation with the Norwegian Public Roads Administration (NPRA) (Norwegian: *Statens vegvesen*). The NPRA is responsible for planning, construction and operation of the national and county road networks in Norway. Today's systems are in many ways not intelligent and will often need supervision of human experts. Therefore, the NPRA wanted to investigate if it is possible to create more intelligent ways to execute traffic control. In this project, we want to reveal if it is feasible to remove the push button at pedestrian crossings by detecting pedestrians with camera sensors. It is important to point out that we in this project will

test the system as if the push button is still there, but that it is a small step towards actually removing the push button.

Removing the push button can improve safety, since it can be difficult for some pedestrians (e.g. blind pedestrians) to locate the button[4]. It may also enable the system to make faster decisions, since it does not have to wait for pedestrians to push the button. In addition, some pedestrians push the button, but still cross the road before the light changes.

Figures from NPRA[5] show that about 36 % of all traffic related accident in Oslo occurs in intersections. In approximately 21% of traffic related accidents, pedestrians are involved. In Norway, about 35 % of all that are killed or seriously wounded in traffic accidents are pedestrians (46 % in the world[6]). These numbers show how important it is to improve the safety for pedestrians, and particularly at intersections and pedestrian crossings.

Other factors which are important for the NPRA and the Norwegian government are economy and the environment [7]. Reducing the duration of red light for vehicles will result in reduced emission of gasses that can cause damage to the environment. Economically, optimizing execution of traffic can lead to reduced delay for both vehicles and pedestrians, which again can lead to reduced costs.

The last important motivational factor, from the NPRAs point of view, is universal design. The Norwegian Ministry of Transportation and Communication describes a universally designed transport system as: “A Transport system that as far as possible can be used by all, without the need for adjustments or special facilities” [7]. Removing the push button of the crossing can make it easier for disabled pedestrians to cross the road, if it is done properly. In addition, knowing what types of pedestrians intends to cross the road can be used for adjusting the transition time for pedestrians, by giving extended transition time to slower pedestrians and giving shorter transition time to faster pedestrians.

1.3 Overview of the report

In the next chapter, we look at some important background information, both related to the traffic domain and to the different approaches we aim to use during the project. In chapter 3, we present some related research within the traffic domain that uses Case-Based

Reasoning. Chapter 4 describes the tools, approaches and methods used for implementing and evaluating the system. In chapter 5, we describe the process of implementing the CBR system. This includes creation of the case base, a description of how the values for the features were acquired, a thorough description of the classes in the system, and a walkthrough of the system. Chapter 6 presents the results of the evaluation of the system, along with a discussion of the results. In chapter 7, we propose some further work and conclude on the results of this project.

Chapter 2

Background

In this chapter, we present the background of this research area. In the first section, we talk about the specialization project we conducted in the autumn of 2011, which forms the basis of the work done in this thesis. Next, we give a brief introduction to how traffic control and traffic in general works to day. Section 2.3 describes a research project called CVIS (Cooperative Vehicle-Infrastructure Systems), which objective is to enable systems that use vehicle to infrastructure communication. Next, a short introduction to the CBR methodology is given. Section 2.5 presents the Intention-based Sliding Doors project. Finally, we describe evolutionary algorithms, which is a mechanism for performing intelligent search. An evolutionary algorithm was used in the system to intelligently estimate the optimal weights of the features in the cases.

2.1 Specialization project

In the autumn of 2011, we conducted a specialization project, where the goal was to investigate whether CBR could be a reasonable method in the domain of traffic control. Traffic control is a wide concept, ranging from controlling a large traffic network through a centralized control station, to controlling a single signal controlled crossing. Many problems were studied to determine if there were any areas in this domain where the strengths of CBR could be utilized. The study included reading scientific papers concerning CBR as a method, CBR in the traffic domain and reading about the traffic domain in general, to be able to fully understand what problems could be solved. The project resulted in a system specification, where the purpose of the system is to control a signal controlled pedestrian

crossing. This specification has been the basis for the system presented in this report. In figure 2.1, the task structure of the system specification presented in the specialization project is shown. It includes essentially the same parts as the system presented later in this report.

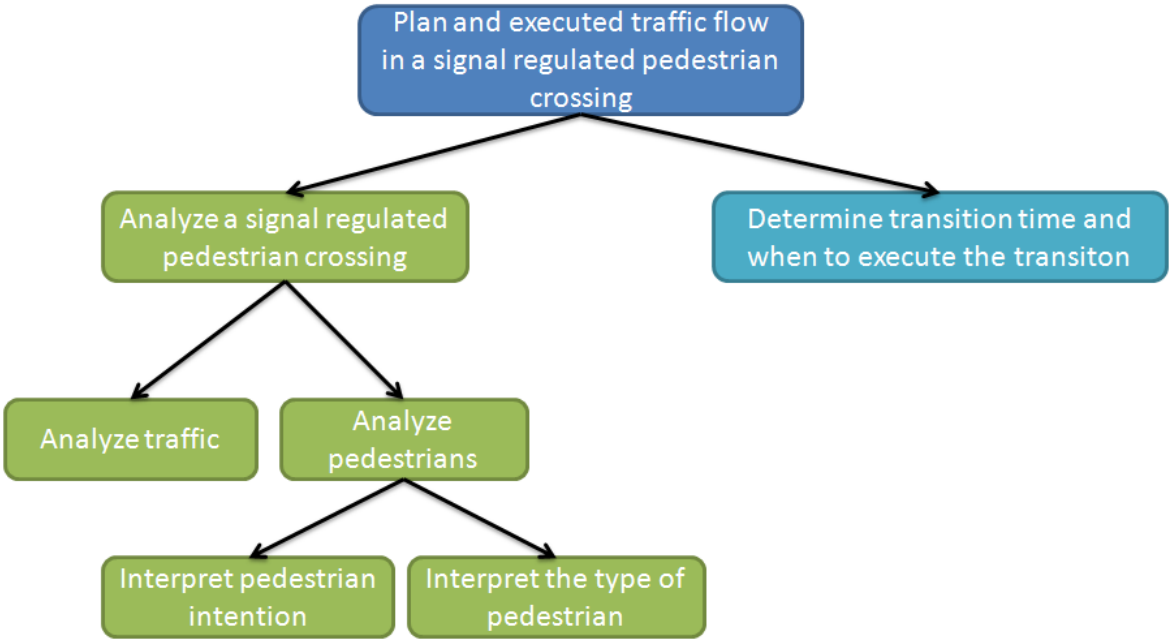

Figure 2.1: Task structure of the architecture introduced in the specialization project

2.2 Traffic today

In today's society, there are many varieties of intersections and pedestrian crossings. An intersection can be of two different shapes; either X-cross or T-cross. Pedestrian crossings can have no signal control, be signal controlled or it can be a PUFFIN-crossing[1]. Signal controlled pedestrian crossings can be either time controlled or traffic controlled. When a pedestrian crossing is time controlled, it means that the system ignores pedestrians and vehicles, and is only controlled by static or dynamic time intervals (dynamic intervals can be changed e.g. by experts sitting at a centralized control station). Time controlled crossings may be used in urban areas where there will be pedestrians waiting at almost all times. Traffic controlled crossings can either detect pedestrians (by using a push button) or detect both pedestrians and vehicles (by using inductive loops that lie beneath the

[1]PUFFIN - Pedestrian user-friendly intelligent crossing

Figure 2.2: Push button with signaling light for pedestrians

asphalt). Detecting both pedestrians and vehicles is most common in today's systems.

An intersection will typically consist of two to four signal controlled pedestrian crossings, and is therefore much more complex. Today, each crossing cannot be controlled separately, and parallel crossings are controlled connectedly. This may obviously impair the traffic flow, since it prevents the system from being able to send vehicles through on one side, while letting pedestrians cross on the other side.

PUFFIN crossings are the type of crossings that probably are most similar to the prototype system presented in this report. It originally stems from United Kingdom[8]. It differs from normal crossings in several ways:

- The lights for signaling pedestrians (see figure 2.2) is on the same side of the road as the pedestrians, and turned so that the pedestrian can both monitor the traffic, and look at the light.
- An on-crossing detector ensures a red light for the vehicles until the pedestrian has crossed (within practical limits).
- Detection of pedestrians on the sidewalk makes it possible to cancel requests to change the lights, if the pedestrian crosses prematurely or walks away.

In a PUFFIN crossing, there will be detectors oriented towards both the crossing and

the sidewalk. The detector oriented towards the crossing is used to adjust the red light for vehicles, by monitoring pedestrians to see when they have crossed the road. A traditional crossing will use the average speed of 1.2 m/s of pedestrians and not adapt to the fact that pedestrians can walk both faster and slower.

The detector oriented towards the sidewalk can be used to cancel requests for pedestrians to cross the road. This will typically occur if the pedestrian pushes the button to signal that they want to cross the road, but either changes their mind or jaywalk. The system will detect this behavior and cancel the request so that the light never changes. The concept is facilitated to work in single signal controlled pedestrian crossings, and not intersections. It also works better in areas where the number of pedestrians is small, since a large amount of pedestrians often will lead to long red periods for vehicular signal groups.

Studies performed by the Norwegian Public Roads Administration[9] showed that the accumulated time of a red light to vehicles was reduced when introducing the PUFFIN concept. The average waiting time for motorists went down from 15 seconds to 13.6 seconds, which corresponds to a 9 % reduction. It is also important to emphasize that these numbers does not include annulment of requests to let pedestrians cross. 9 % of requests to let pedestrians cross were not used, which according to the studies would give a total reduction in delay for motorists of 17.5 %. There are several PUFFIN crossings installed in the world and also in Norway. In Oslo, the PUFFIN concept has been tested in two crossings, one at Rv155 Enebakkveien and another at Rv160 Bærumsveien.

In this project, we present a system that we believe can act more intelligently compared to the existing PUFFIN crossings, and can give more promising results. If detection of pedestrians that have intention to cross the road is accurate enough, the system can be able to work without a push button to signal change of lights. It can also only detect pedestrians with intention to cross the road, instead of detecting all pedestrians that enters the range of the sensors.

Another relevant system is the SPOT/UTOPIA system[10]. It is an adaptive control system for signal regulated areas. The system is developed by Mizar Automazion, in Italy, and its goals as a control system is that:

- No public transport vehicles should be stopped by the signal controlling units
- Other traffic should have as good or better termination conditions as before

SPOT/UTOPIA performs optimization based on predicting the traffic volume. It uses detectors to build a profile of the traffic arriving at each intersection. The profiles form the basis of the optimization of the signal switching in the intersections. Traffic controlled pooling can both terminate more traffic, as well as giving priority to selected groups of road units, than e.g. a time controlled system. SPOT/UTOPIA is in use in a number of cities like; Oslo, Gothenburg, Copenhagen, Malmo, Trondheim and in several cities in Italy.

2.3 CVIS

In this section, we present a framework called CVIS, which was used in developing the CBR system. According to the developers of CVIS [11]:

> CVIS (Cooperative Vehicle-Infrastructure Systems) is a major new European research and development project aiming to design, develop and test the technologies needed to allow cars to communicate with each other and with the nearby roadside infrastructure.

CVIS use OSGi (Open Services Gateway initiative framework), which is a framework to make the process of creating module based systems easier in Java[12]. Figure 2.3 illustrates some areas where CVIS is intended to be used. In OSGi, applications are called bundles. The difference between a bundle and a standard Java application is that bundles can be remotely installed, started, stopped, updated and uninstalled without requiring any reboot. The objective of the CVIS project is to increase efficiency and safety in traffic by enabling Vehicle-to-Vehicle and Vehicle-to-Infrastructure cooperation. The communication is made possible by using wireless networking and GPS sensors. Road side units (RSU) are placed on the side of the road, and vehicles in an area around the RSUs can download applications from it. This makes it possible to create context-aware applications, which can serve the motorists in a more useful manner. The RSUs can also communicate simple information to the vehicles, like letting the motorist know if there are dangerous conditions ahead. In the future, more and more vehicles will be equipped with touch screens, which will enable systems to communicate visually with the motorists. Vehicles can also communicate information to the RSUs. Examples are speed

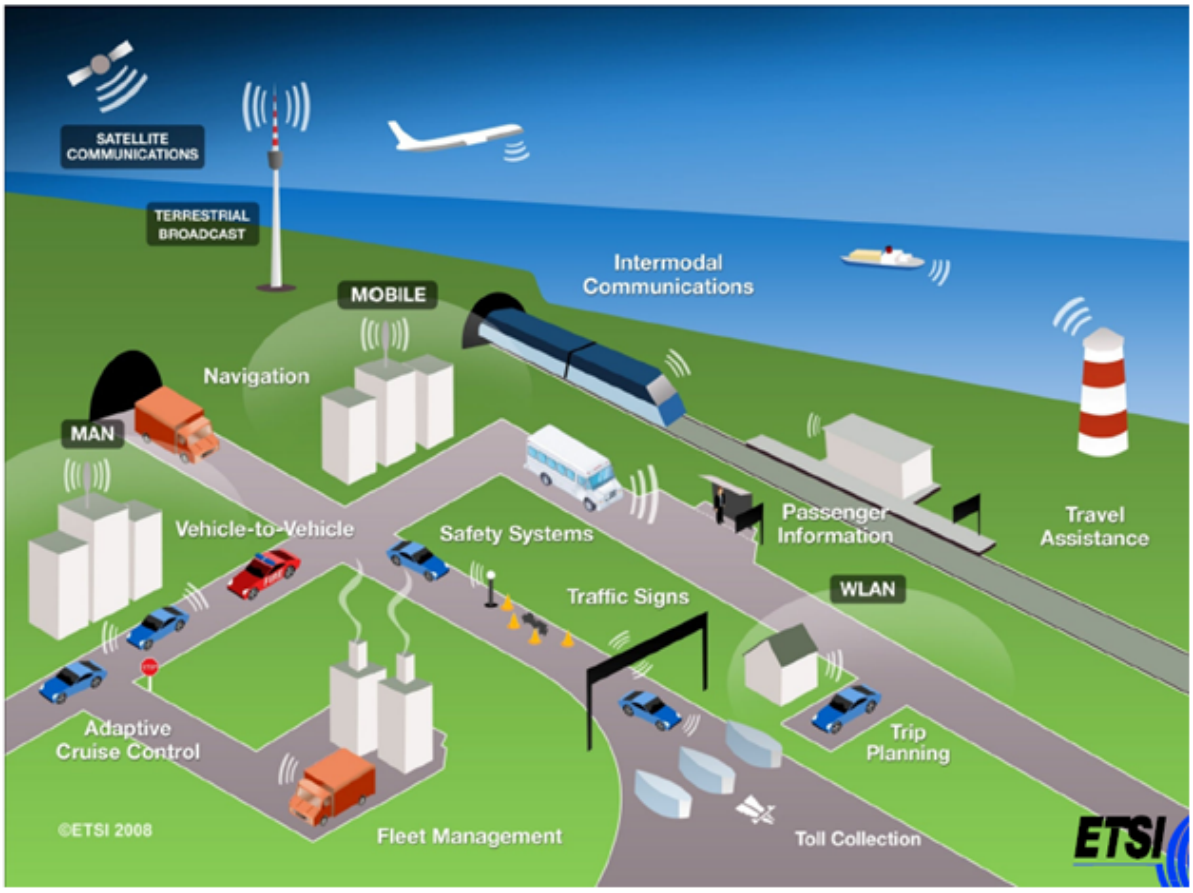

Figure 2.3: An illustration of how CVIS is intended to be used

of the vehicle or that the vehicle is "acting strange" (i.e. the vehicle can communicate if the driver is doing something they should not do).

Communication in CVIS can be achieved in several ways. CVIS uses an architecture called "Communications access for land mobiles" (CALM) for communications. CALM enables the following communication modes:

- Vehicle-to-Infrastructure (V2I): communication initiated by either roadside or vehicle (e.g. petrol forecourt or toll booth)
- Vehicle-to-Vehicle (V2V): peer to peer ad-hoc networking amongst fast moving objects following the idea of MANET's/VANET's.
- Infrastructure-to-Infrastructure (I2I): point-to-point connection where conventional cabling is undesirable (e.g. using lamp posts or street signs to relay signals)

The CVIS project chose 20 applications that were developed, to show that the project is feasible, and to demonstrate its potential. Here, we will briefly describe two of them:

The first application is used to improve safety for pedestrians. It is a vehicle alert system, which informs an RSU that a vehicle with abnormal behavior is approaching (e.g. if the driver is a drunk driver). The RSU (at the intersection) can receive the information and evaluate whether it should let pedestrians cross. This information can also be forwarded to vehicles (touch screen inside the vehicle) and pedestrians (e.g. on a Smartphone) close to the crossing[13]. The second application was designed to do strategic routing of traffic[14]. To achieve this, the application sends information from vehicles, about the area around the vehicle and the vehicles destination, to a centralized unit. In this way, the system can route each vehicle optimally, and at the same time balance all the individual routes, to give an optimal overall traffic flow.

As mentioned earlier, universal design is a key area for the Norwegian Public Roads Administration. Liao et al. has developed a system to make it easier for blind and partially sighted people to cross the road in an intersection[15]. An application for Smartphone's were developed to make it possible for the disabled pedestrian to signalize to the intersection that it has a desire to cross the road. It can also give site specific information, such as size of the intersection and the number of crossings in the intersection. The application was not implemented in CVIS, but is a good example of a possible CVIS application.

In Trondheim, there is a test site for CVIS at E6 going north towards the city center (see figure 2.4). The blue circles are rooftop stations and the red circles are street stations. Since the hardware and software is already installed, it enables the possibility to test CVIS applications in actual real-world scenarios.

Figure 2.4: Test site for CVIS in Trondheim

2.4 CBR

The roots of Case-based reasoning (CBR) can be traced back to the early 1980s with Roger Schank's work at Yale University in the U.S with his dynamic memory model[16]. The model was the basis of Janet Kolodner's system CYRUS, which was the first CBR system to be implemented[17]. CYRUS was a question/answer system about travels and meetings of and US former Secretary of State (Cyrus Vance).

The idea behind CBR is that situations have a tendency to occur more than once and that earlier experienced situations can be used to solve new, similar situations. CBR is inspired by the way we humans solve problems, by being reminded of a similar problem that has been experienced earlier, and use that experience to solve the new problem. Aamodt and Plaza proposed to break the reasoning process into 4 steps; Retrieve, Reuse, Revise and Retain[1] (see Figure 2.5). In the retrieval step, cases that are an exact or a close match with the current problem will be retrieved. Reuse takes one or several cases and tries to adapt the solutions so they can solve the problem. In the revise step the solution is evaluated by testing in the real world or by asking an expert. If the solution

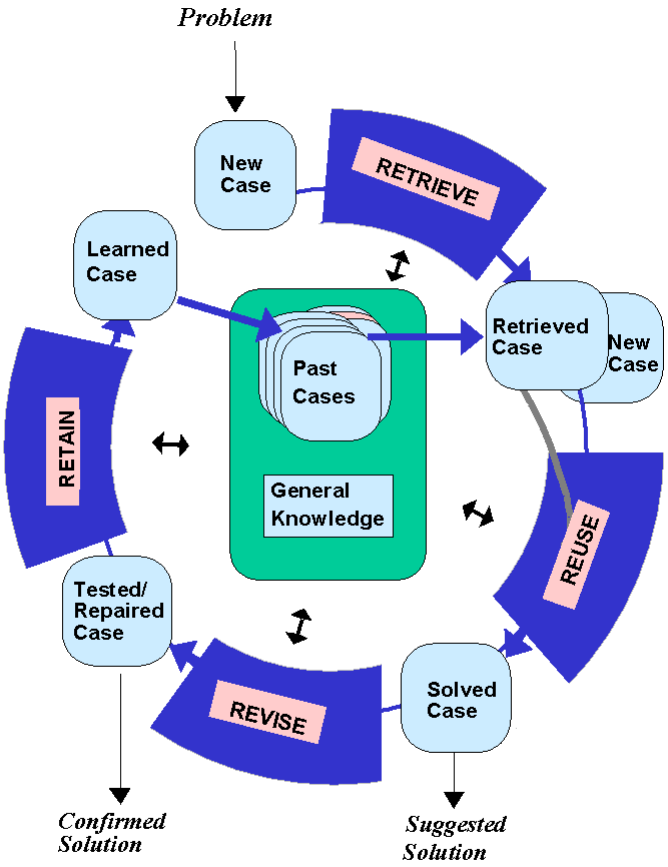

Figure 2.5: The CBR cycle by Aamodt and Plaza[1]

is satisfactory, it will be stored in the last step, the retain step.

There are several differences between CBR and other instance-based methods. For one, cases in CBR can have a very complex case structure (e.g. the cases can be an integrated part of an ontology or a semantic network). In the traffic domain, there is a lot of general domain knowledge that could be included in a system. This is not possible with simple instance-based methods. Moreover, in CBR there are typically fewer cases that cover greater parts of the problem domain, instead of many instances that each covers a small part of the domain. Creating a large amount of examples can be a more difficult process than creating some cases with more knowledge integrated. Another thing that separates CBR from other instance-based methods is that it adapts the cases to fit the new situation, instead of just finding the closest possible match. All of this makes it reasonable to believe that CBR is a stronger method in the field of traffic control and traffic in general.

2.5 Intention-based Sliding doors

An important feature of the system we present in this report is that it can interpret whether pedestrians intend to cross the road and use this to make faster and more accurate calculations. To do this interpretation, we have integrated a system made by Solem[3], which purpose was to infer the intention of humans walking towards a sliding door. The motivation behind the system was that sliding doors today are mostly based on simple detection mechanisms, and that a more intelligent sliding door could perform better. It uses a Kinect sensor to detect human activity in front of the door. Strategic points on the human body (hips, shoulders and torso) were detected to infer the user's intention. They used a rule-based reasoning mechanism to decide whether the user intended to enter or not. The results of the from the study showed that the system could infer the correct intention of the user in 77-86 %[2].

2.6 Evolutionary algorithms

In a CBR system, adjusting the weights of the features is important, because there will often be a difference in the importance of the features. It can be hard to find the optimal weights, because of the high amount of possible combinations. Evolutionary algorithms (EAs) are a way of searching through a solution landscape, like the weights of features, in a more intelligent manner than random search. It is inspired by the evolutionary process seen in nature, where individuals are born and only the best individuals are allowed to reproduce.

The basic cycle of an EA is shown in figure 2.6[2]. The process starts with creating a number of random individuals to explore the complete search space. Each individual is a given a genotype (typically a bit-string), which can be directly or indirectly translated into a phenotype. The phenotype normally represents a solution to the problem. Next, the solution is evaluated by a fitness function, where better solutions are given a higher fitness value. It is important in this step to not only give credit to optimal solutions, but also solutions that are good in some way. After fitness is given to all individuals, the parents, i.e. the individuals that will be used in reproduction of child individuals, are chosen. This is normally done by choosing a number of adults that have the highest

[2] 86 % when removing test cases that was classified incorrectly due to hardware limitations.

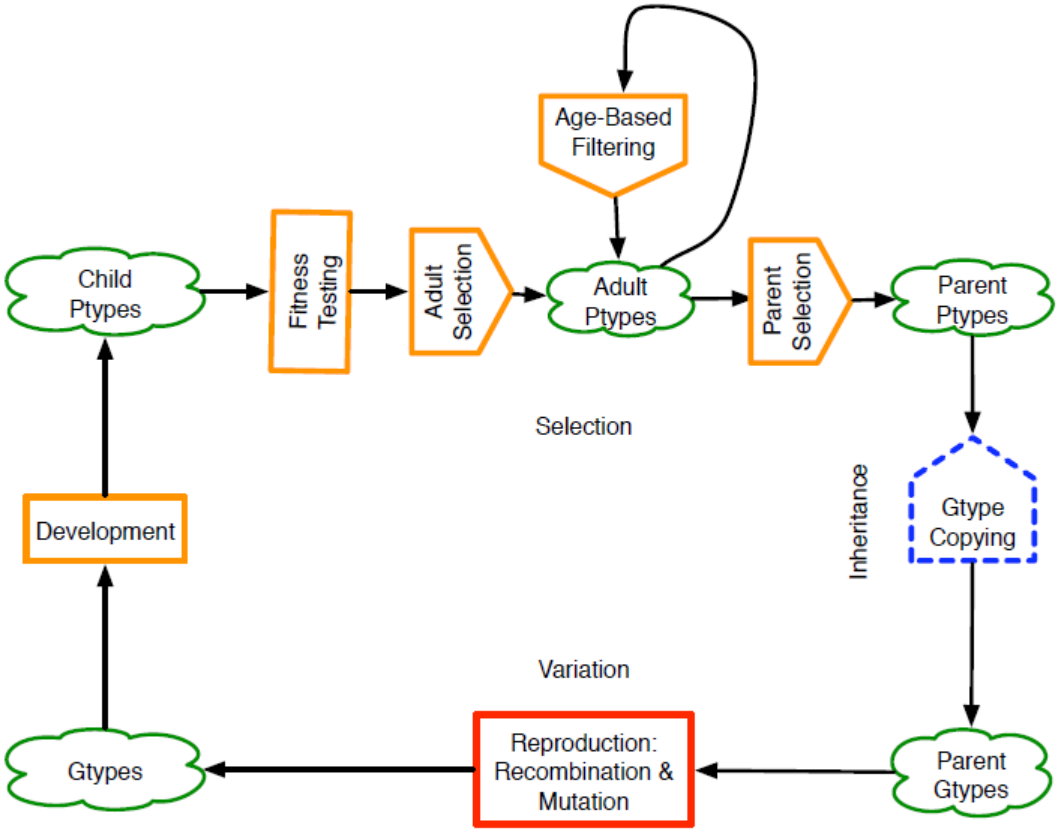

Figure 2.6: The basic cycle of an Evolutionary Algorithm[2]

fitness value to be the parent set. As in nature, the genotype of two parents are then combined and mutated, to form new individuals. This cycle is repeated until the fitness function gives satisfying results by one or more of the individuals.

Chapter 3

Related research

Expert systems have been used in the traffic domain for many years. Wentworth splits systems that are used in traffic into four very broad groups; "Traffic Management and Control", "Traffic Impact and Safety", "Highway Design and Planning" and "Highway Management"[18]. The first two groups are well represented in AI, and particularly by rule-based systems. However, rule-based systems have been considered to be somewhat limited in the traffic domain[19][20]. Waters and Li point out that traffic safety is a complex function of a wide range of factors, such as: "road network, environmental conditions, driver behavior and vehicle conditions". Creating a rule base that can cover such a complex problem domain can be very difficult.

In the group "Traffic Management and Control", there have been developed rule-based systems to support strategic routing of traffic[21][22]. Sadek et al. highlight some limitations to using rule-based systems in traffic management. For one, they argue that it can be hard find the optimal places to split traffic, because it is of a more algorithmic nature. Second, it can be difficult for an expert to formulate their experiences of routing traffic, into simple rules.

As mentioned before, CBR can be seen as a parallel to rule-based systems, but that CBR does not need an explicit model of the domain. A case contains the experience of a significant situation. This makes it easier to expand the case base with new cases, because cases can have some overlapping between what problems the cases solve, without it affecting the reasoning process. In rule-based systems, it is much more important to keep the independence between rules, so that the rules will not be contradicting.

Even though CBR is not widely used in the traffic domain, some work has been done in

the area. SICAS [23] is a system created for sharing domain knowledge about traffic safety, between experts and analytics. The system uses a concept called organizational memory, where access to existing data is made easier for the user. Generalized domain knowledge was stored in a normal knowledge base, while cases describing earlier experienced situations were stored in a case base. They believe that generalized domain knowledge alone can solve simple problems, but that complex problems have to be solved with the experiences from the case base.

Case studies were carried out by experts, along with analysis of domain-related documents, to build an expertise model. The results from the case studies were also used for determining what knowledge pieces a case should incorporate. This includes the main features of the case, potential problems that can occur, contributing factors, possible correcting actions and the global conclusions, if there are any.

The authors state that CBR seems like a promising approach in this domain. The reason for this is that by having experts solve case studies, cases can be generated in a natural way. In this way, there exists an historical set of site analyses accumulated over several years. They also point out that experts often refer to case examples when solving a site analysis, thus it is a natural approach for the experts. Furthermore, since cases also can contain an explanation on why a situation has occurred, it can make the communication between the analyst and experts easier.

Lin et al. have developed a system called ISECR (Information system for estimating crash reductions)[24]. ISECR is described as a functional, intelligent database that consists of published literature, which quantifies the estimated reduction benefits for various road safety improvements. Users can query the system for cases that are similar to the problem at hand. To choose the best cases, they are weighted by quality, so that good cases will be retrieved before less good cases. The cases retrieved are then summarized to give an estimate of the range and reliability of how road improvement can improve safety (reduce crashes).

The contents of each case are separated into six sub cases, were each part contains different types of knowledge. For example, one part contain general knowledge (such as author of the study and title of the study) and another part contains knowledge about the location where the crash has happened (e.g. at a signalized intersection). Each sub case can contain multiple features. This is similar to how we considered representing cases in

our system, since problems can be represented by both pedestrian and vehicle knowledge, and the solution can be split into transition time and when to execute the transition. This is explained in section 5.3.

Another system that uses CBR in planning transportation systems, is a system called PLANiTS[25] (Planning and Analysis Integration for Intelligent Transportation Systems). PLANiTS gives support in planning transportation systems, by retrieving similar experiences, and presenting the solutions to the experiences in a structured and proper manner. In the system, cases consist of actions; to improve the transportation system, performance measures and environments; defined in terms of space, time and user/traveler descriptors. This forms a planning vector, where the values of the different features are used for calculating the distance between a query and the historical case. In addition, by letting the user change the level of stringency, the system can provide the user with different solutions. The CBR system can be used along with structured models, semi-structured expert systems and/or unstructured electronic support for human interactions.

In another study, CBR was used together with GIS[1] technology to evaluate the traffic safety in rail-road intersections in Calgary, by analyzing collision history and site-specific data[19]. GIS was used for manipulating and analyzing collision data, while the CBR part of the system was used for identifying possible safety issues, by looking at historical collision data at the rail-road intersection. They use a CBR tool called eGain, the case description (the features of the case) is represented by questions, which has to be answered by the user of the system. As in the ISECR system, cases are split into multiple parts, here referred to as clusters, which deals with different parts of the problem. When the user has answered all the mandatory questions, the eGain knowledge reasoner searches for the most relevant cases and presents them to the user. The study does not specify how similarity between cases is calculated.

TIMELY is a system that was developed to generate an initial design for the signal phase in an intersection, and then simulate delays in traffic and adjust the signal phases accordingly[26]. It uses CBR to find a good initial phase design. The system searches through a case base, containing cases of earlier created intersections, were the solution is the signal phases at these intersections. Similarity between cases is calculated by looking

[1]Geographic Information System is a system used to capture, store, manipulate, analyze, manage, and present all types of geographical data

at the traffic volumes going into the intersection and the intersections geometry. If a case is retrieved successfully, its solution is reused, without any modification/adaption to the solution. This is an interesting system, since it attempts to find the optimal signaling phases for an intersection. In our system, the solutions given by a case is a nominal value that must be translated into a time interval. The performance of the system is highly dependent on that these intervals are correctly configured. Using a similar system as TIMELY to estimate these intervals for intersections or signal controlled pedestrian crossings would therefore be interesting.

All of the systems presented above differ from our system in that they are used for planning and offline safety issues only. Therefore, these systems do not need to address issues concerning using the system to make real-time decisions. The main issues concerning real-time decision are related to automatically building query cases and uncertainties that arise. For example, a system that relies on sensors to provide information for the query case must always take into account that the data may be incorrect. The next two systems are used for traffic control, and are taking decisions in real-time.

Schutter et al. describes a multi-agent decision support system that uses CBR to assist traffic control center operators in doing their work[27]. The system is used for giving support in taking decisions when unforeseen events occur (e.g. traffic accidents or unexpected weather conditions). Cases were generated offline by using macroscopic or microscopic traffic simulation, or by having experts consider actual traffic situations during a time period. When the system is used in a real traffic control center, a module for adaptive learning was proposed as a solution to make the system more effective when new problems arise. The traffic network is split into multiple sub networks, each representing a tractable part of the traffic network. Each sub network has its own case base, where the cases describe the traffic situation in the sub network, along with the predicted inflow demands and outflow restrictions, the control measures and the incident status.

The most interesting part of this study is the use of multiple case bases to make the system scalable when presented with larger traffic networks. The predecessor of the system did not scale well when it was tested on a large traffic network. A similar approach could be used in the future with our system, since it is important that the CBR controlled pedestrian crossing can "cooperate" with the rest of the traffic network. This is discussed further in section 7.1.

Li and Zhao presented in 2008 a system that uses CBR in urban intersection control[28]. The system is similar to the system presented in this report, in that both are used for real-time traffic control, with no experts available to correct or evaluate the decisions taken by the system. It uses a three-step process. First, a case is built by gathering data from the detecting and surveillance system. Second, the most similar previous case is retrieved from the case base. If no similar case exists, actuated control is used to take a decision. If this happens, the system will temporarily store the solution used by the actuated control. It then uses feedback information from the detecting and surveillance system to evaluate if the solution was successful. If the feedback information tells the system that traffic congestion is more serious after the solution was applied, the system will delete the case. If, on the other hand, the traffic congestion is less serious, the case will be permanently stored, so that it can be used in future decision making.

As can be seen, many of these applications use CBR along with one or more other methods, to solve problems. Of the seven researches mentioned in this chapter, 3 focus on safety [23][24][19], 2 focus on design and planning[26][25] and 2 focus on traffic management[27][28]. None of the articles concerns highway management. Also, none of these studies addresses problems concerning pedestrians. This also seems to be the general trend in systems in this domain. It is surprising, since pedestrians are a great part of traffic, and particularly in urban traffic.

We have not been able to find any CBR systems that solve problems related to pedestrians, but there have been some work on this matter in the field of AI, especially in computer vision. Hogg et al. predicts the movement of a pedestrian by continuous observation of long image sequences[29]. Although this study was performed to investigate the surveillance problem of identifying abnormal situations, the ideas can still be linked to the traffic domain. A model of pedestrian movement was learnt, in an unsupervised manner, by tracking objects over long image sequences. It is based on a combination of a neural network, implementing vector quantization and a type of neuron with short-term memory capabilities. Models of the trajectories of pedestrians could also be used to assess 'incidents of interest' within a scene or predict future object trajectories. Assessment of abnormal situations could be useful also in the traffic domain, since it can be used to prevent accidents. It is reasonable to believe that these predictions could be used to interpret the intention of pedestrians. By looking at the direction of the trajectories, if

the trajectory goes into the crossing or towards it, it could also be used to predict that the pedestrian intends to cross the road.

Many systems have been developed that uses computer vision to detect pedestrians [30][31][32]. Gavrila presents a prototype system that detects pedestrians from a moving vehicle. The system used a two step approach for detecting objects. The first step involves looking at contour features and a hierarchical template matching approach, to create a set of candidate solutions. The second step involves filtering out the best solutions from the candidate solutions. Detecting pedestrians from a moving vehicle can prevent accidents involving pedestrians, by for example automatically stopping the vehicle if it detects a pedestrian close to the front of the vehicle. Even though this cannot be directly linked to our system, it is still an interesting study, since it is a good example of knowledge that can be interesting to incorporate with a CBR system that controls traffic. For example, in a situation where a vehicle has to automatically stop to avoid an accident, this information could be forwarded to the system, so that it can take it into account.

The NPRA has a vision of zero fatal- or sever accidents in road traffic. Since pedestrians are a key part of these accidents today (see section 2.2), it is important for the NPRA to create road systems that are safe for all traffic units. It is interesting to investigate if CBR can take good decisions in this domain, because of the small work that has been done there. In addition, cases represent experienced situations like accidents, situations that resulted in a traffic jam or similar unwanted outcomes. It can also represent desired situations, like situations where there are no traffic jams. By using solutions to problems that has already occurred, it is possible to learn from earlier mistakes, to create safer and more efficient pedestrian crossings.

Chapter 4

Methodological approach

In this chapter, we will describe the different methods that were used in conducting this project. It includes a description of the tools and frameworks that were used, why we decided to use myCBR[33] over jCOLIBRI [34] and how we gathered knowledge for the CBR system. We will also describe the methods that were used to evaluate the system.

4.1 Tools

In this section, we list the different tools and frameworks that were used in developing the system.

4.1.1 CBR tools and frameworks

myCBR 3.0 BETA is a tool to create own Case-Based reasoning applications in Java. It is open-source and developed at the DFKI[1].

myCBR 2.6 is a plugin to Protégé. This plugin includes functionality to set up the Case-Base visually in Protégé. This project can be stored as XML and used in myCBR 3.0 BETA.

Protégé 3.4.8 is an open-source ontology editor and a knowledge-based framework. That support different plugins to extend the original functionality[35].

[1]German Research Center for Artificial Intelligence

4.1.2 Bundles and software

CVIS is a system to standardize communication between vehicles and road side units, based on the OSGi framework, as mentioned in section 2.3.

OSGi is a framework to create bundles that can be integrated in CVIS described in section 2.3.

Sliding doors is an intention-based system developed by a master student at NTNU. The system was written in C++ and uses the OpenNI framework, along with other frameworks described in 4.5.1, to intepret whether humans have the intention to enter a sliding door[3].

SCANeR Driving simulator is a simulation program for testing and driving, developed by OKTAL. With the simulator, it is possible to create customized traffic facilities and establish two-way communication between the simulator and third-party applications[36]. It comes in two versions; SCANeR Studio dedicated to engineering and research and SCANeR Driving Training that is used for training and safety awareness.

4.1.3 IDE's

Eclipse is an open source IDE (Integrated Development Environment). It supports easy installation of plugins to use SVN and create OSGi bundles[37].

Microsoft Visual C++ 2010 Express is a free C++ IDE and is a part of the Visual Studio 2010 express[38].

4.2 myCBR vs. jCOLIBRI

We had three choices for frameworks to develop the CBR system; myCBR, jCOLIBRI and creating our own framework. The last choice was discarded early, because the two other frameworks both seemed promising and we were sure that we could create a framework that could provide the similar functionality in such a short time. We will now describe the frameworks myCBR and jCOLIBRI, and argue why we chose one over the other.

4.2.1 jCOLIBRI

jCOLIBRI is a framework to create CBR applications and have been in development since 2005. The framework comes in two editions; one for developers (jCOLIBRI) that want to code the application in Java, and another for designers (jCOLIBRI Studio) where the source code is created automatically after making the configurations in the jCOLIBRI Studio.

The jCOLIBRI framework provides functionality for all the steps in the CBR cycle (see section 2.4). It supports five different retrieval strategies; with seven selection mechanism, more than 30 similarity metrics, 20 adaption and maintenance components, and a lot of other features. Overall, the jCOLIBRI seems like a complete and well-functioning framework, with a diverse set of features.

When starting a development process with a large framework, it is necessary to have access to a good documentation, to avoid spending large amounts of time understanding the framework. jCOLIBRI is well documented with Javadoc and several test examples. It is also important to point out that the jCOLIBRI framework supports importing similarity functions from myCBR.

4.2.2 myCBR

myCBR is an open-source case-based reasoning tool developed at the DFKI. The tool comes in two versions; myCBR 2.6 builds on top of the Protégé ontology editor and myCBR 3.0 BETA, which works as a standalone application. Both of these systems are created as plugin projects and include similarity measures, a retrieval engine and support for creating explanation-aware CBR systems.

myCBR 2.6 is a plugin to Protégé that includes basic CBR functionality, such as similarity-based retrieval. When a CBR system is developed in Protégé, the project can be exported to XML files that contain the similarity measures and the case base. In Protégé, cases can be added manually or by importing a CSV[2] file. These cases can also be exported as XML.

myCBR 3.0 BETA makes it possible to create CBR systems in Java. The tool can be included as a library and imported into the project. myCBR 3.0 BETA lets the developer use methods for importing the case base, using similarity measures and for using the

[2]Comma separated values

retrieval engine. Since cases can be exported as XML from Protégé, building a case base is both easy and fast.

The benefits of using myCBR 3.0 BETA as a development tool to create a CBR system is that it is a standalone plugin to any type of project, and it is documented in a Javadoc and some simple examples. The obvious disadvantage is that it is still in a BETA version. Because of this, some methods have not yet been implemented, it is not bug free and the documentation is still sparse.

4.2.3 Choosing the right tool

jCOLIBRI is a framework for making it easier to create CBR systems, in all of the steps of the CBR cycle. myCBR 3.0 BETA is a more simple tool that provide less functionality. Functionality for retaining cases is not fully implemented, though it's planned. For creating less complex prototype CBR systems, myCBR seems to be the best tool, since it's easier to get familiar with, and it provides much of the basic functionality needed. jCOLIBRI seems like the better choice when creating more complex CBR systems.

jCOLIBRI is much better documented than myCBR 3.0 BETA. Both frameworks have a Javadoc, explaining the different methods implemented, but jCOLIBRI has tutorials and a lot more code examples. myCBR 3.0 BETA only got one code example, which is fairly basic, and no tutorials. The benefit is that the framework is much smaller, so it's still quite easy to understand. The developers of myCBR are currently working on better documentation, which will probably come with newer versions of myCBR 3.

The system we present in this report has to be an OSGi bundle in CVIS. The developers of myCBR 3.0 BETA has ported it to work as an OSGi bundle. For that reason it is easier to implement an application with myCBR in CVIS than in jCOLIBRI, which is not ported to OSGi. If we chose to use the jCOLIBRI framework, the project would have to be converted into an OSGi bundle, which would take extra time, because OSGi bundles require a specific structure of the code and properly adjusted settings.

myCBR supports most functionality to build a simple CBR system. Since it's also already ported to work with OSGi, was the best choice for CBR framework. Still, we believe that in creating a larger CBR system, jCOLIBRI would be a better choice and that it could be necessary to create the system in jCOLIBRI if it were to be released.

4.3 myCBR

We will now describe myCBR more thoroughly, since we chose to use it as the framework for creating the CBR system. We have used both versions of myCBR in this project. myCBR 3.0 BETA was used for developing the system, while myCBR 2.6 was used together with Protégé to create the case base and for creating similarity functions.

4.3.1 Documentation

Both of the versions have different documentation. The reason they got different documentation is because myCBR 2.6 is a plugin for Protégé, while myCBR 3.0 BETA is a development tool created for developers. myCBR 2.6 is better documented, because myCBR 3.0 is still in beta.

myCBR 2.6

Version 2.6 is documented through a tutorial, which cover all the necessary steps, from installation of the framework, to creating a complete case base. It also explains how to import and export cases to Protégé. To test the case base, the tutorial explains how to use the retrieval engine. An explanation of how to export a project from Protégé, and use the project in myCBR 3.0 BETA, is not covered by the tutorial.

myCBR 3.0 BETA

myCBR 3.0 BETA is documented through a Javadoc and a code example. The Javadoc describes the system in detail, and gives a good overview of the different methods that the framework provides. Although the system is thoroughly described in the Javadoc, it is difficult to get an overview of how all the methods can be used together. A graphical description of the architecture or more detailed examples would probably make this easier. The code example from the webpage introduces how to create a case base with myCBR 3.0 BETA. This example is short and is only of aid in the starting phase, to understand how to use the basics of the framework. myCBR 3.0 BETA isn't a large CBR tool and doesn't require a lot of documentation, but more advanced code example would easy the development process.

4.3.2 Case base

In myCBR 3.0 BETA it is only possible to create the case base by using methods in the framework, and not with a graphical tool like Protégé. This is both time-consuming and inflexible. The case base should be easy to create and edit, because changes will happen, especially in the early stages of the development process. Since Protégé, with myCBR 2.6, provides easier ways for both creating the case base, defining similarity functions and for setting the weights of the features, we used it for this purpose.

The creation of the case base can is done in the following steps; first a CSV file is created with all the cases, second the weights and similarity measures are set, and finally the project is exported to XML files. The files contain the whole project and can be loaded into myCBR 3.0. Small changes in the case base can be made directly in the XML files, although this can make the files unreadable by the XML parser in myCBR 3.0 BETA.

The following files are created from Protégé:

- {FILE_NAME}.pprj is a file used to load the project in Protégé.
- {FILE_NAME}.XML contains the case base, and is used by Protégé.
- {FILE_NAME}_CBR_CASEBASE.XML contains the case base created by myCBR 2.6
- {FILE_NAME}_CBR_EXPLANATIONS.XML contains the explanations if the system is made explanation-aware.
- {FILE_NAME}_CBR_SMF.XML is a file containing the similarity measures, and is used in myCBR 3.0 BETA to load the project.

4.4 Knowledge acquisition

Knowledge acquisition is an important part of creating a knowledge based system. In this section, we will therefore present some of the methods that were used for acquiring the knowledge needed for creating and testing the CBR system. The first section describes a test crossing that was used for getting relevant data when testing the system and for configuring domain dependent parts of the system. Section 4.4.2, explains how we

proceeded in counting pedestrians and vehicles at the test crossing. Finally, we give a short introduction to how the rest of the knowledge for creating cases was acquired.

4.4.1 Test crossing at Brattøra

As mentioned earlier, most of the traffic data available today are traffic and road data only. The CBR system is highly dependent on information about pedestrians, and as a result of this, we decided to count pedestrians in a pedestrian crossing. This made the tests of the system much more realistic, rather than sending a random number of pedestrians to the system. On recommendation from the experts at the NPRA, we decided to count pedestrians and vehicles (to make the numbers consistent) at a signal controlled pedestrian crossing at Brattøra, in Trondheim. A sketch of the crossing is shown in figure 4.1. It shows the traffic lights and the detectors (inductive loops, marked with D1-D5).One of the reasons why we selected this crossing, is that it is not controlled by any other signaling systems. In addition, the traffic around the crossing is quite high, because it is close to the center of Trondheim and it lies in an area where many people work. There are also many pedestrians that use this crossing. The speed limit at the crossing is 60 km/h.

4.4.2 Counting vehicles and pedestrians at Brattøra

We performed two sessions of counting vehicles and pedestrians at Brattøra. The first counting was at 12.30 pm on the 8th of March 2012, and lasted for one hour. The second counting was at 16.30 pm on the same day, also lasting one hour. The reason why we chose these time periods was that the first represents a time period where traffic is normal, while the second represents a time period where it is rush hour. Before the counting was conducted, a form was created to document the results (shown in figure 4.2). The time column indicates the time interval; 1 is the first minute, 2 is the second minute etc. The vehicle column represents the number of vehicles that have passed through the crossing within the current time interval, in both directions. The pedestrian column represents the number of pedestrians that have used the crossing within the current time interval. The slowest type of pedestrian column is used to document what type of pedestrian was the slowest to cross the road within the time interval, e.g. one pedestrian on a bike will be marked as fast and if one of the pedestrians is slow, it is marked as slow. A fast pedestrian, among normal pedestrians, was marked as normal.

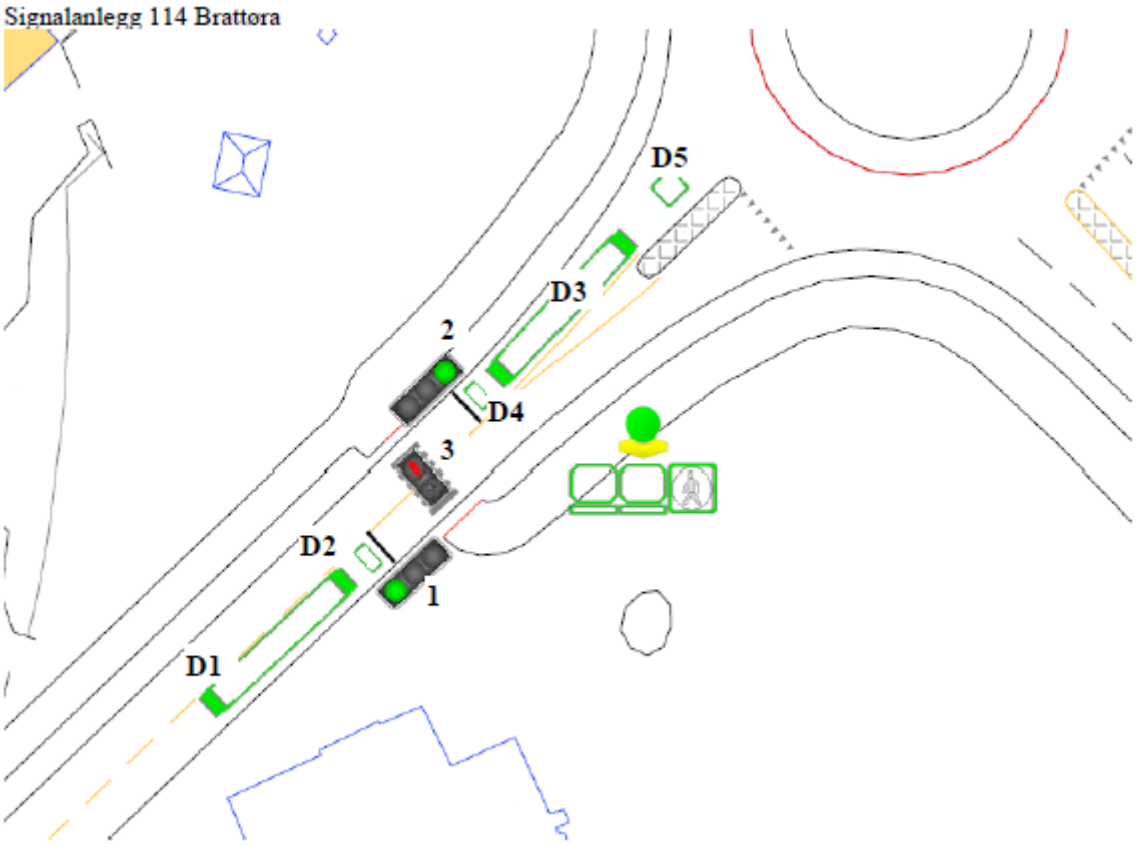

Figure 4.1: Pedestrian crossing at Brattøra

Date: Time period:

Time	Vehicles	Pedestrians	Slowest type of pedestrian
1			
2			
3			
4			

Figure 4.2: Part of the form for counting pedestrians and vehicles

4.4.3 Collecting information to create the case base

Collection of information to create the case base was done by conducting a study of the domain and by having meetings with experts on traffic control and traffic safety. A more detailed explanation of the process is described in section 5.2.

4.5 Installing frameworks

The CBR system is dependent on 3rd party frameworks to function properly. This includes installing all necessary frameworks and libraries for running the Intention-based Sliding doors framework, and installing the CVIS framework for making the system able to communicate with other traffic applications. In this section, we will describe briefly what frameworks and libraries are needed for running the CBR system.

4.5.1 Installing the Intention-based Sliding Doors framework

To measure the different features of the human body, the Intention-based Sliding doors framework use 3rd party frameworks and libraries. Running the code therefore required installation of these frameworks/libraries. OpenNI is an open source framework that provides an interface for physical devices and for middleware components. The framework provides an API for writing applications utilizing natural interaction. NITE is a middleware intended to be used together with the OpenNI framework. It provides a means for tracking humans and for getting skeletal data. OpenCV is an open-source library providing functions for computer vision, with a focus on real-time applications. The last installation was of a collection of C++ libraries called Boost. It provides some functionality that is not supported by basic C++.

4.5.2 Installing CVIS

To run the CVIS bundles, the program Knoplerfish Pro CVIS 3.2.0 is required. This program was given from the NPRA, along with an installation guide. The Eclipse Enterprise Edition was used since it's the version of Eclipse that supports creation of bundles. Knoplerfish is an OSGi service framework and was installed through the Eclipse marketplace, which made it possible to create new bundles. JAR files given by NPRA were

included in the new bundle project to provide CVIS functionality. These JAR files are what differentiate CVIS from Knoplerfish.

4.6 How to evaluate the system

In this section, we will describe the different ways the system was evaluated. In the first two subsections, we will present two methods that were used to improve and evaluate the strength of the case base, and why we chose to use these methods. Next, we describe the method that was used for evaluating how good the system would perform at the task of controlling a pedestrian crossing. Finally, we explain why the interpretation of pedestrian intention module was not a part of the simulator tests.

4.6.1 Expert validation of the system in real-time

When creating cases for the system, the best way would be to add cases that directly reflect real-world situations. Situations that are relevant to the system happen with varying frequency, so it is hard to just sit down, observe traffic, and wait for abnormal situations to happen. Since these situations has not been stored anywhere (both because there has been no need to do so, and since mixing information about pedestrians and vehicles is not regular), other than in the minds of experts, we needed a way to create these cases. This is challenging, because it is easy to leave some cases out if it is not done in a systematic manner.

When having meetings with the experts at the NPRA we found that they had problems getting an overview of the case base when presenting them with 21 cases. Imagine doubling or even tripling the amount of cases. It would make it even harder to get a good overview of the cases in the case base. To make the process of covering the complete problem domain easier, a solution could be to divide the cases into categories, to see what problems are not completely covered. The problem with this approach is that it is actually quite hard to divide cases into categories, because many cases can be put into multiple categories.

Another solution to this problem is to create a module for the system, where cases can be updated, deleted and added cases to the case base, while the system runs. This will let the expert actually observe the current situation, and let him/her evaluate the solution proposed by the system. The query case and the retrieved cases will be represented in a

systematic way, which makes the updating of the case base easier. We chose to implement this module in our system, because we believed it would make the process easier and that it would make it easier to further develop the system.

4.6.2 Leave-one-out cross-validation

To evaluate the strength of a CBR system, and the case base that it uses, it is important to use a method that can evaluate whether the system can reason beyond the examples (cases) already given to the system. If the system cannot solve new problems, its usefulness is greatly reduced. To evaluate the systems strength, a test set with cases never seen by the system has to be used. One possible way to acquire this test set is to observe a pedestrian crossing and creating the cases from actual data, or to talk to experts and create a test set together with them. The drawbacks of these methods are that they can be very time-consuming.

Because of the time constraints of this project, we decided to use leave-one-out cross-validation, instead of creating a test set. The reason we use leave-one-out cross-validation, over k-fold cross-validation, is because the case base is small. K-fold cross-validation has a tendency to waste data when the k is small[39]. Leaving out just one case is normal when evaluating case-based systems, since case bases rarely contain a large amount of cases.

Leave-one-out cross-validation removes one case from the case base, and then uses this as a test case. Every case in the case base will be used as a test case, and the result is the number of cases that were correctly classified. This method helps to find shortcomings in the solution space, by revealing what parts of the case base is not adequately covered. We also use this method in setting the weights of the features of the cases. How the cross-validation was implemented into the system is explained in section 5.6.2.

Knowing when the results of a cross-validation test are satisfying can be difficult to estimate. The results can be 100%, and the system can still perform badly over a test set, if some categories are not covered at all. Also, adding cases does not necessarily increase the number of correctly classified cases, if the case itself cannot be classified by the other cases. Still, the number of correctly classified cases does not have to be 100%, since the case is removed from the case base when performing the evaluation. Because of the reasons mentioned above, we set the limit for when the case base is complete to 75%.

The evaluation process will be executed in the following steps:

1. Find the weights to the features that gives best performance
2. If the result is not satisfactory (less than 75% correctly classified cases), add more cases in the parts of the solution space that got few cases and go back to step one.
3. Case base complete (more than 75% correctly classified cases).

4.6.3 Performance tests in the SCANeR Studio

The optimal way to test a system that performs traffic control would be to test it in real-world scenarios. Such tests would require a lot of resources and it also would require thorough tests of the system beforehand, to ensure that the system would not be a safety risk. Since this is not possible, due to the time constraints of this project, we had two other choices; create a set of test cases and give them as input to the system or compare it to a system similar to those used today in a traffic simulator. We chose the second option. The reason for this was that this would probably test the system in a more correct way, since the situations occurring in a simulator would be more realistic. It is reasonable to believe that this would give the system cases that it has not seen before.

The problem with such an approach is that it is in fact quite hard to mimic the behavior of a real signal controlled pedestrian crossing. Many of the crossings today are controlled by intersections connected to the crossing (see section 2.2 about the SPOT system), which makes it difficult to create a simulated version of the crossing. Also, the crossing should be a possible bottle-neck, with high traffic flow and many pedestrians, so that the system can be tested against a crossing that is difficult to control. There were two possible ways we could perform the simulations; with the SCANeR Studio or with a traffic simulation software called Aimsun. The SCANeR Studio was briefly described in section 4.1. Aimsun is developed by TSS-Transport Simulation Systems and according to them Aimsun is: "A traffic simulation software that allows you to model anything from a single bus lane to the whole of Manhattan"[40]. The reason we chose the SCANeR Studio is because it provides possibilities for controlling the traffic on a more detailed level. For example, the NPRA has equipment that lets you drive a simulated vehicle in the scenarios. This could enable testing specific scenarios, like if a vehicle is driving too fast. Aimsun would provide possibilities for using more exact models of the traffic in and

out of the crossing. For example, in the future it is likely that there will be models in Aimsun for simulating the traffic network in the city of Trondheim.

The signal controlled pedestrian crossing we decided to compare with is a crossing at Brattøra in Trondheim. This crossing is not controlled by other intersections. It is also located in an area with much passing traffic and many pedestrians, due to close by industrial areas. This comparison site was described more thoroughly in section 4.4.1.

The simulated crossing in the SCANeR Studio is not identical to the real crossing. For one, the traffic that goes into the crossing is not exactly the same as in the real crossing. To get the same traffic, we would need to create a much larger traffic network, to simulate the traffic that comes from both directions in the crossing. The roundabout, that is located north of the crossing, is also removed. Creating a roundabout in the simulator requires a lot of work when it comes to setting the priorities for vehicles in different directions. Also, when we configured the roundabout in the SCANeR Studio vehicles sometimes stopped, for no apparent reason. The time constraints of this project also made it difficult for us to spend much time on perfecting the surroundings of the pedestrian crossing in the simulator.

To emulate the traffic that enters the crossing, we created a signal controlled pedestrian crossing on each side of the crossing controlled by the CBR system. In this way, we could regulate the traffic to get some gaps in traffic, which is more realistic than sending a constant stream of vehicles through the crossing. These crossings were configured so that they would not have too much of an impact on the traffic, so that they would not cause any traffic jams. The settings for the two signaling systems are shown in table 4.1 (traffic light for vehicles) and table 4.2 (traffic light for pedestrians). The simulated section of road is shown in figure 4.3. The red circles mark the crossings, and the crossing in the middle is the one used for testing. The distance between the crossing and the two crossings on each side is about 700 meters. Figure 4.4 shows a more detailed screenshot of the CBR controlled traffic light in SCANeR Studio.

One of the goals of this project was to create a system that would perform better than today's system in terms of; efficiency, safety and user friendliness (see section 1.1). Since it is not possible to test whether the user friendliness has improved in a simulator, we decided to only test efficiency and safety.

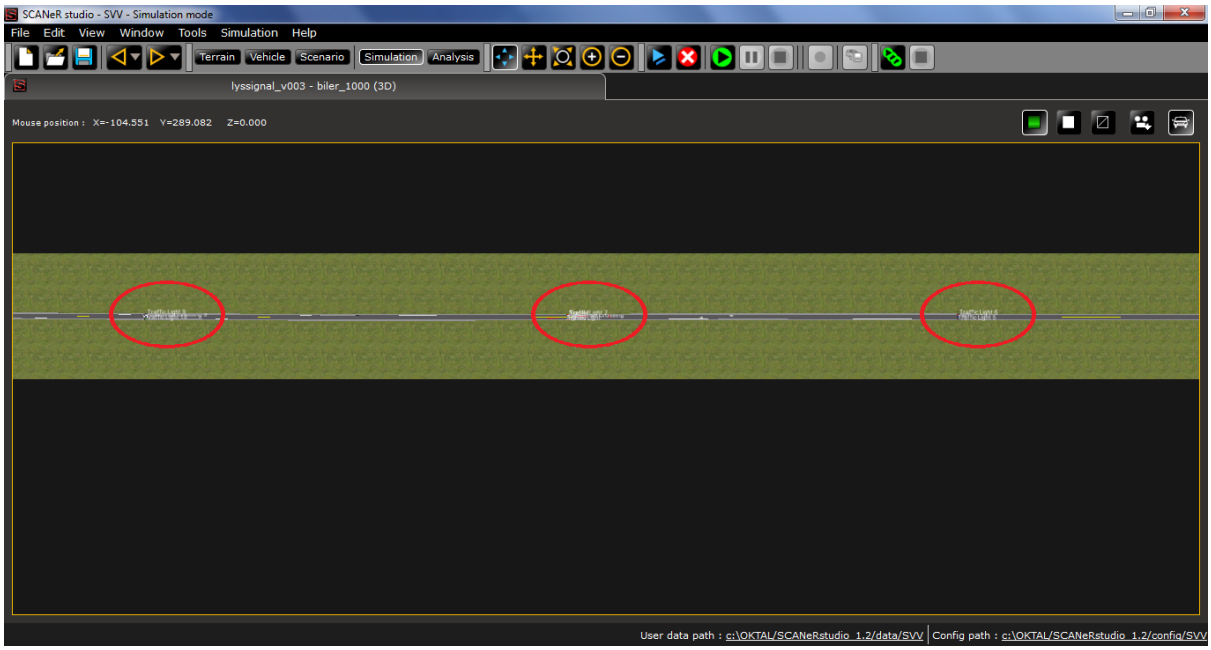

Figure 4.3: Screenshot from SCANeR Studio, showing the simulated section of road used in the test scenarios

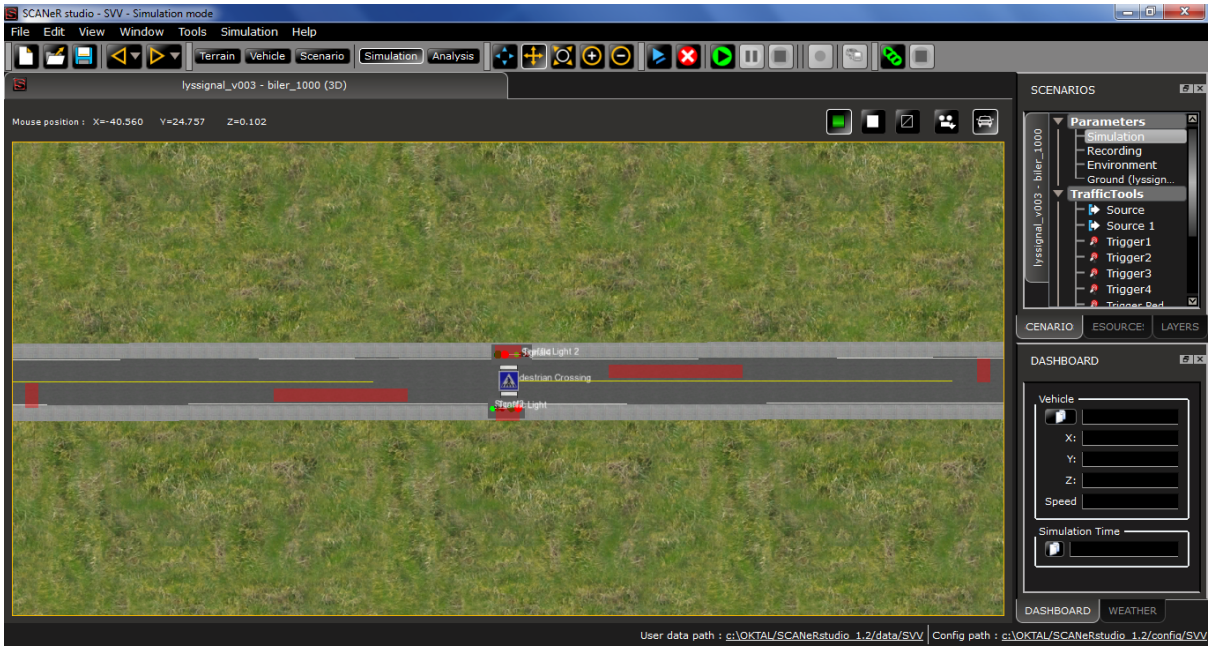

Figure 4.4: Screenshot from SCANeR Studio, which shows the traffic light that is controlled by the CBR system.

Signal color	Interval time
Green	90 seconds
Green and Amber	3 seconds
Red	8 seconds
Amber	1 seconds

Table 4.1: Settings for the traffic lights for vehicles, on each side of the CBR controlled traffic light

Signal color	Interval time
Red	94 seconds
Green	5 seconds
Green blinking	3 seconds

Table 4.2: Settings for the traffic lights for pedestrians, on each side of the CBR controlled traffic light

4.6.4 Evaluating the system with interpretation of pedestrian intention

A good evaluation of the CBR system, with interpretation of pedestrian intention, would be to test the accuracy of the interpretation and to test if the module could improve the CBR system in terms of; increased traffic flow and greater safety for pedestrians and vehicles. There are several obvious difficulties that arise when doing such an evaluation. For one, a good evaluation of such a system would require great synchronization between sending people into the range of the Kinect sensor and simulating traffic in SCANeR Studio. Second, the Kinect sensor have limitations when it comes to speed of detection and the number of people it can detect (because of 3rd party library limitations, the program often crashed when five or more human shaped objects were in range of the Kinect sensor). Also, creating scenarios with pedestrians entering the sensor, some with positive (i.e. a pedestrian intends to cross the road) and some with negative (i.e. a pedestrian that does not intend to cross the road) intention, requires a lot of effort.

Since the system presented in this thesis is an experimental system, we decided, in cooperation with our supervisors, that it was not necessary to fully test this part of the system. The purpose of the module is to show that it is possible, to some extent, to infer

the intention of pedestrians, instead of having the pedestrians signal their intention by pushing a button. Therefore, we did not see the necessity to evaluate the accuracy of the module any further than the tests done by Solem in the Intention-based Sliding Doors project[3] (these results were presented in section 2.5). Also, we decided not to test this module in terms how it would increase the efficiency and safety in a crossing, because of the difficulties mentioned above. Instead, we tested the system by running some simple scenarios, with few pedestrians and a less complex background environment. In this way, we believe that the concept of interpreting whether a pedestrian has an intention to cross the road could be proved to be doable.

Chapter 5

Implementation

In this chapter, we will describe the CBR system that has been implemented, along with the modules that were created to assist the system. Section 5.1 gives an overview of the system. A description of the case base, including the structure of the cases, how the case base was filled and a description of the similarity functions, is given in section 5.2. Section 5.3 goes through each feature, explaining how the data for each of them is obtained, and how the data is distributed in the system. Section 5.4 explains in detail the process of retrieving cases in the system. Section 5.5 gives a more thorough description of the system. Finally, we describe the modules that were created for enhancing the evaluation process and for improving the case base.

5.1 System overview

An overview of the system is given in figure 5.1. In the bottom left corner, a Kinect sensor is shown. This illustrates the sensors that are supposed to be placed on the top the traffic lights, to monitor the area around the crossing. It sends images from the sidewalks, which are analyzed by the module for interpreting the intention of pedestrians, and the not yet implemented interpretation of type of pedestrian module. Data is sent through to the CBR system by using communication standards defined by CVIS. If the CBR system receives one or more positive intentions, the CBR cycle will start.

The first step in CBR cycle is the creation of a new query case, which happens when the number of positive intentions is greater than zero. To create a new case, the system will collect the most recent data from the sensors. Vehicle- and pedestrian data are

continuously modified through the CVIS communication, so that the query case in the CBR system will be up to date. In this way, the query case represents the current traffic situation. Next, the query case is used for retrieving similar cases. If more than one case is retrieved, the most frequent solution is chosen. It will either contain the transition time for pedestrians, or suggest that the traffic light should not be changed. If the traffic light should not be changed, the CBR cycle will start from step one with the creation of a new query case. This process will continue until traffic light has been changed, or when the number of positive intentions is zero.

If a solution containing a transition time is returned by the system, it is passed on to a simulated traffic light. This is a visualization of a pedestrian traffic light, which shows the outcome of the solution given by the CBR system. To control the traffic light in SCANeR Studio, a CVIS event is published from the simulated traffic light. In this way, the simulated traffic light and the traffic light in SCANeR Studio will always be synchronized.

The gray box in the figure represents a module that is used for "sending" pedestrians into the system. It makes the evaluation process easier, since it allows the system to run without having to send pedestrians in front of the Kinect sensor. In addition, it allows the user to specify the type of pedestrian, since the module for interpreting the type of pedestrian has not been implemented. This module is described in section 5.6.3.

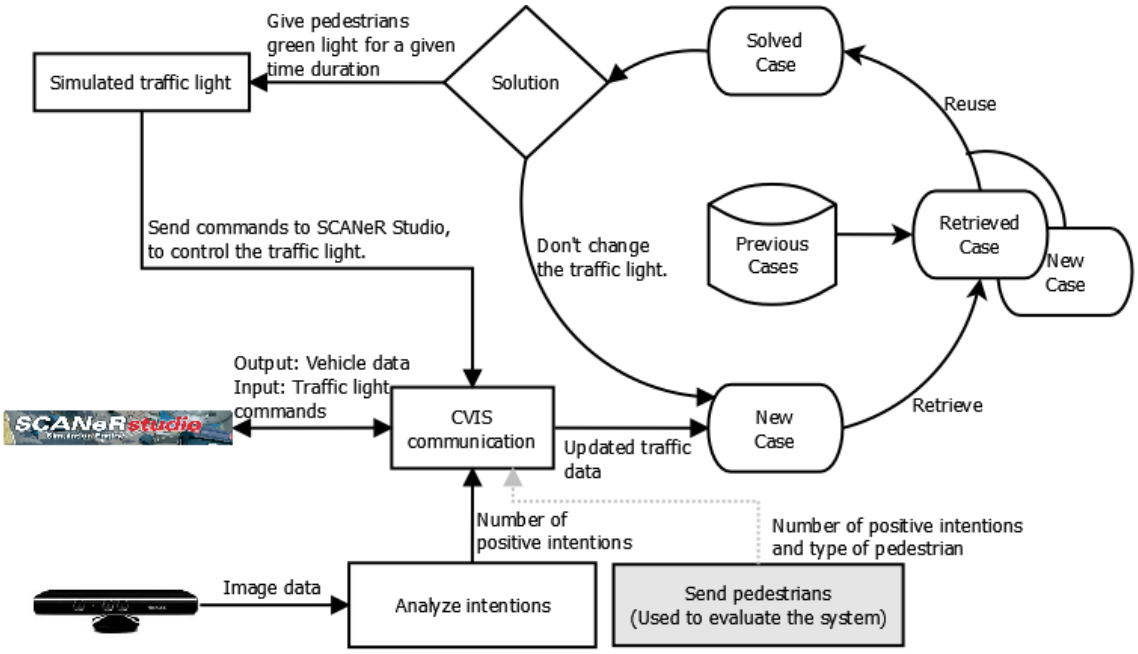

Figure 5.1: An overview of the system

5.2 Case base

In this section, we will describe how the case structure was established and how the case base was filled with constructive cases. We also describe the similarity functions that define how similarity between cases is calculated.

5.2.1 Case structure

In this section, we will describe the process of creating a case structure based on the work done in our specialization project, and information gained through meetings with experts in the relevant domains. Our focus has been on making a case structure that is rich, in the way that it can handle complex problems, but yet is simple, in that all unnecessary features are discarded. This has been an iterative process, where a case structure were produced and refactored over time, to get the best possible final result.

Knowledge acquisition

The first step in creating a good case structure is to acquire those features that are important to describe situations in the problem domain. Extracting these features can

often be a very difficult process. Experts can have different views on what features are the most important ones, and it is therefore important to do a thorough investigation. Knowledge acquisition is an essential part of creating the case structure, since it can ensure that the solutions proposed by the systems are the most satisfying solutions.

As mentioned in section 2.1, the process of studying the problem domain and the possibilities of using CBR in the domain, began in the specialization project. It included reading manuals produced by the Norwegian Public Roads Administration, reading articles on the matter and going to the ITS[1] World Congress 2011 in Orlando. From this research, we discovered that information about the situation in a pedestrian crossing can be split into two parts; vehicle information and pedestrian information. Examples of traffic related information will typically be how the traffic flows and speed of the vehicles in the area. An example of pedestrian related information is number of pedestrians. It is important to point out that information regarding vehicles is much more available than pedestrian information, since there have been a lot of focus towards this category of traffic units.

One of the decisions we had to make when creating the case structure was whether we should have one or two case structures. The reason for this is that the system has to both estimate the optimal transition time and decide when the transition should be initiated. Thus, we had to consider if we should have one case base with cases to estimate transition time and one to decide when it should change the light, or if we should have one case base with cases representing both solutions. One of the positive things of having two separate case structures, and two case bases, is that some of the features are more related to one part of the solution. For example, if a vehicle is detected or not, does not directly affect the transition time, but it greatly affects when to make the transition, since it can be useful to let a detected vehicle pass before letting the pedestrians cross the road. Furthermore, two case bases make the retrieval process faster, since the number of combinations is actually lowered. Still, we decided to create one case base. The main reason for this was that the cases for deciding when to change the light had almost all the features of the other case structure. Also, in some situations the transition time given to the pedestrians is important for when the lights should be changed. For example, if a slow group of pedestrians wants to cross the road during the rush hour, the system should

[1] Intelligent transportation systems

Feature	**Possible values**
Number of vehicles	Low, Normal, High
Gap in traffic	True, False
Vehicles per time unit	Low, Normal, High
Distance from the closest vehicle to the crossing and the crossing	Short, Medium, Long
Speed of the vehicle closest to the crossing	Slow, Normal, Fast
Number of positive intentions	Few, Normal, High
Time waited	Short, Medium, Long
Speed of the pedestrian closest to the crossing	Slow, Normal, Fast
Type of pedestrian	SlowGroup, Slow, Normal, Fast
Current date and time	Date format
Proposed transition time (solution)	null, ExtraTime, Normal, LessTime

Table 5.1: First draft of the case structure

wait as long as possible to change the lights, because the group will need an extended transition time. These situations are solved more easily if the solutions are retrieved from one case base.

The second step of acquiring the most important features was to sit down with experts on the domain and benefit of their knowledge. Our first meeting was with Kristin Kråkenes and Helge Stabursvik at the NPRA in Trondheim. Kristin Kråkenes is an expert on signal control and Helge Stabursvik is an expert on traffic safety in general. Prior to the meeting, we proposed a possible case structure, based on the work described above. Our first concern was that we would put ideas into the expert's heads and make their responses biased, if we showed them the proposed structure right up front. We therefore attempted to explain the problem to them without showing them our initial draft of a case structure. We soon discovered that acquiring the knowledge we wanted was as harder than expected. It was both hard for us to describe to them exactly what we wanted to achieve with our system, and to describe what information we wanted from them. An obvious mistake was to try and extract the features from them, without showing them our proposed solution. When showing them our proposal, it made it easier for them to understand what a case is and what a feature in a case can be. The proposed case structure is shown in table 5.1.

The meeting also strengthened our assumption that work done in traffic control is centered on vehicles and not pedestrians. The experts were much more direct in their response when talking about vehicles, while the responses concerning pedestrians were vaguer. Additionally, the features describing vehicles and traffic were more obvious, and hence the experts agreed on most the features we proposed. For example, it is quite obvious that the traffic flow affects the number of pedestrians the system can allow to cross the road. For pedestrians, they also agreed on most of our proposed features, but they pointed out some features that were more important than others. We took this into account when creating the final case structure.

The final case structure

The final case structure is shown in table 5.2. Number of vehicles and vehicles per time unit were two features in the initial draft of the case structure. Since the number of vehicles is more or less the same as vehicles per time unit, if it is a very small time unit, we merged them into one feature, namely traffic flow. Traffic flow gives an indication of how much traffic has passed through the crossing in a given time interval (e.g. five minutes). It is an important feature because e.g. a higher traffic flow can indicate that there is a greater chance for traffic jams to occur. Thus, when the traffic flow is high, the system must be more cautious to let pedestrians cross the road, while if the traffic flow is low, the system can let pedestrians cross more often.

The second feature is vehicle detected, i.e. if there is a vehicle detected in any of the lanes in the direction of the crossing. This is a very important feature, because it enables the system to detect gaps in traffic, so that it can exploit these, and let pedestrians pass at the most optimal time (because of this we could remove the "Gaps in traffic" feature). If a vehicle passes one of the detectors, the vehicle will stay detected until it has passed the crossing. The way this is done is to estimate the time the vehicle will use from the detector to the crossing, by using the speed of the detected vehicle. Since the detectors are placed close to the crossing (the standard is about 70 meters to the detector furthest away), we did not include the "Distance from closest vehicle to crossing and the crossing" feature. In the future, there is a great chance that vehicles will be equipped with GPS, so that detection can be done in a much longer range, and for a much longer period of time.

The speed of the detected vehicle is also a part of the case structure. It is important

because if a vehicle with high speed is approaching the crossing, the system might need to let the pedestrians wait, because the vehicle may not have time to stop.

The next feature is the number of positive intentions. This information is not available in today's systems, where a push button is used to request transition, because only one pedestrian pushes the button. In our system it is acquired in the module for interpreting pedestrian intention. It is important because the number of pedestrians that requests to cross the road can affect both when the pedestrians should pass, and for how long (e.g. if there are many pedestrians with intention to cross the road, the system should give them more transition time, and if there is one pedestrian that wants to cross, the system can let him/her pass if there is a small gap in traffic).

The time pedestrians have waited for a green light is also an important feature. It directly affects when the system should let pedestrians cross, since if the time waited is "Long", it should let pedestrians pass no matter what. In other situations, it is used to estimate when to let pedestrians pass (e.g. if a vehicle is detected and time passed is "Short" or "Medium", the pedestrians should not get a green light).

The last feature is the type of pedestrian. It is probably the most important feature, since it very important when it comes to giving the correct transition time. For example, if the type of pedestrian is "Slow", the pedestrian should always be given "ExtraTime" or more. This feature can also affect when the system should give a green light to pedestrians. For example, if the type is "SlowGroup" and the traffic flow is "High", then the system should wait until "Long" time has passed because the slow group needs "ExtraExtraTime" to pass.

5.2.2 Similarity functions

After establishing the structure of the cases, in terms of a problem description and a solution, the next step is to define how similarity is calculated between different values of features. In this section, we will present the similarity functions that were defined for each feature (the local similarities), while we in the next section will describe how the similarity between a query case and a case in the case base is calculated (the global similarity). The myCBR plugin in Protégé offers multiple ways of configuring similarity functions (table similarity, ordered similarity, externally scripted similarity functions in Jython, and more). We used this plugin for configuring the similarity functions. It is

Feature	**Possible values**
Traffic flow	Low, Normal, High
Vehicle detected	True, False
Speed of detected vehicle	null, Slow, Normal, Fast
Number of positive intentions	Few, Normal, High
Time waited	Short, Medium, Long
Type of pedestrian	SlowGroup, Slow, Normal, Fast
Proposed transition time (solution)	null, ExtraExtraTime, ExtraTime, Normal, LessTime, LessLessTime

Table 5.2: Case structure

important to point out that the values used in table similarity are not based on research, but on the information we got from the experts at the NPRA. A possible improvement of the system could therefore be to either make the systems learn these parameters, or have an expert set more optimal values. All values that are exact matches are given the maximum similarity value, which is 1.0. Also, all similarities are symmetric, i.e. if one feature is similar to another feature; the second feature is just as similar to the first feature.

Traffic flow similarity

The configuration of the similarity of the traffic flow feature shown in the table in figure 5.2. High values of "Normal" traffic flow can indicate that traffic is rising. Therefore the similarity between "High" and "Normal" traffic flow is set to 0.2.

Case Base Values

Reset	Normal	High	Low
Normal	1.0	0.2	0.0
High	0.2	1.0	0.0
Low	0.0	0.0	1.0

Figure 5.2: Traffic flow similarity

Speed of detected vehicle similarity

The configuration of the similarity of the speed of detected vehicle feature shown in the table in figure 5.3. Here, the similarity between "Normal" and "Slow" is set to 0.5. The

reason for this is that it is less important if a vehicle is "Slow" or "Normal", because the safety is not an issue for any of the values.

Case Base Values

Reset	Normal	null	Fast	Slow
Normal	1.0	0.0	0.0	0.5
null	0.0	1.0	0.0	0.0
Fast	0.0	0.0	1.0	0.0
Slow	0.5	0.0	0.0	1.0

Figure 5.3: Speed of detected vehicle similarity

Positive intention similarity

The similarity for number positive intentions is configured in the table shown in figure 5.4. "Normal" values are given partial credit for when the number of positive intentions is "Few" or "High".

Case Base Values

Reset	Normal	Few	High
Normal	1.0	0.2	0.2
Few	0.2	1.0	0.0
High	0.2	0.0	1.0

Figure 5.4: Positive intention similarity

Time waited similarity

The similarity for time waited is configured in the table shown in figure 5.5.

Case Base Values

Reset	Short	Medium	Long
Short	1.0	0.0	0.0
Medium	0.0	1.0	0.0
Long	0.0	0.0	1.0

Figure 5.5: Time waited similarity

Type of pedestrian similarity

The similarity for type of pedestrian is configured in the table shown in figure 5.6. For this feature, the values for "Slow" and "Slow group" are given a 0.2 similarity, because both values indicate that there are slower pedestrians that request to cross the road. Similarly, the values "Normal" and "Fast" are given a 0.2 similarity. This is because we define "Normal" pedestrians as pedestrians that can increase their speed if needed.

Case Base Values

Reset	Normal	Fast	Slow	SlowGroup
Normal	1.0	0.2	0.0	0.0
Fast	0.2	1.0	0.0	0.0
Slow	0.0	0.0	1.0	0.2
SlowGroup	0.0	0.0	0.2	1.0

Figure 5.6: Type of pedestrian similarity

5.2.3 Global similarity

The similarity between two cases is calculated by the weighted sum of all the features. The weights were learned by an evolutionary algorithm, which is explained in the next section.

5.2.4 Evolutionary algorithm

Testing all combinations of feature weights to improve the system, would be a very time-consuming, especially if it was to be done manually. Of course, some intuition could be used to set the weights according to the knowledge we have about the problem domain. Still, getting the perfect set of weights would be difficult. As a result of this, an evolutionary algorithm was implemented to easy the process. The algorithm consists of a core method, a unit, selection strategies and genetic operators. The core method initializes values and runs the evolutionary cycle. Some of the parameters in the system can be manually changed to improve the results. These parameters are the number of units, the rate at which mutation should be performed and the number of mutations that should be performed.

The first step of the cycle is to create units representing different solutions. A unit is created with a genotype, which can be translated into a phenotype. The genotype is represented by an array of size 46, were each place in the array can either be the value zero or one. It is translated into the phenotype by splitting the genotype into six parts, where each part represents one of the weights of the features. The weight is the binary value of the zeroes and ones.

There are three types of units; children, adults and parents. The first units that are created have a randomly initialized genotype. These are the adults. Selection strategies are used for choosing which of the adults should be allowed to be parents. The selection strategies are based on a fitness function, where different units get different fitness values,

based on how good the solutions are. Typically, the units with the highest fitness will be allowed to reproduce, but there is also some randomness in the selection, to be able to explore the whole solution space.

The fitness function runs the cross-validation algorithm described in section 5.6.2, and uses the results from it as fitness for the unit (e.g. if the result is 10 out of 20 correctly classified, the fitness would be 0.5).

Then a full replacement of units is used, which means that all adults are replaced by a set of children. Two parents will create two children by doing a cross-over on their genes. Mutation is performed on the genes, to explore the whole solution space. To ensure that the very best units doesn't disappear in the selection of parents, elitism is used, which involves storing a number of adults that have the highest fitness, so that they are guaranteed to reproduce.

When the fitness of the population is high enough, the cycle will stop, and the solution represented by the best unit will be returned. The weights found by the EA are shown in figure 5.3. The weights range from 0 to 12.8.

Feature	**Weight**
Traffic flow	0.5
Vehicle detected	12.4
Speed of detected vehicle	10.2
Positive intentions	1.5
Time waited	3.8
Pedestrian type	6.2

Table 5.3: Weights found by the EA

5.2.5 Building the case base

Building the case base consisted of several phases, both before and after testing the system. In the first phase, we created new cases manually in cooperation with the NPRA. The experts at the NPRA did not fully understand what a case is in our terms, and we also realized that sitting down with the experts and creating useful cases from scratch, would not be a very good approach. Because of this, we created a set of cases based on some facts they provided:

- If the traffic flow is high, pedestrians can wait longer
- If the traffic flow is high, the transition time for pedestrians should still be long enough for them to be able to cross the road before the light changes
- If no cars are detected at the inductive loops, the traffic light should change to utilize all gaps in traffic (this is especially important when traffic flow is high)
- If a slow group (e.g. a school class) wants to cross the road, it is desirable to let the whole group cross the road together and therefore give a much longer transition time

21 cases were created, where the focus was to cover the facts listed above. In the meeting with the experts at the NPRA, we wanted to check whether all cases were giving the correct solution, if some of the cases where less important and if there were any important cases that should be added.

In the next phase of building the case base, we created a set of training cases and used it to improve the case base. The training set was based on the numbers acquired when counting vehicles and pedestrians at Brattøra. These numbers were also used for finding the reference values for features in the case base (e.g. a "High" amount of pedestrians was set to seven pedestrians or more). Each case in the training set was created by looking at the number of pedestrians that crossed the road at the same time, and by looking at the number of vehicles in the past five minutes, to see what the traffic flow was. The features "Vehicle detected" and "Speed of detected vehicle", was manually inserted because we did not have the exact time for when each vehicle passed the inductive loops at the crossing. Furthermore, the type of pedestrian was set to a value other than "Normal" in some cases, because it was very few pedestrians that were slower or faster than the norm, when we counted.

The CBR system is constructed so that an expert can interrupt the system if it makes any mistakes (a more detailed description of this system can be found in section 5.6.1). Figure 5.7 shows the user interface where the expert can interrupt the system if it gives incorrect solutions. When running the system against the training data, we made sure that that the solutions given by the system were correct. If a solution was correct, the system could continue. If the system made a mistake, we could either enter the correct solution to the problem or propose a change in the weights of the attributes. In this way,

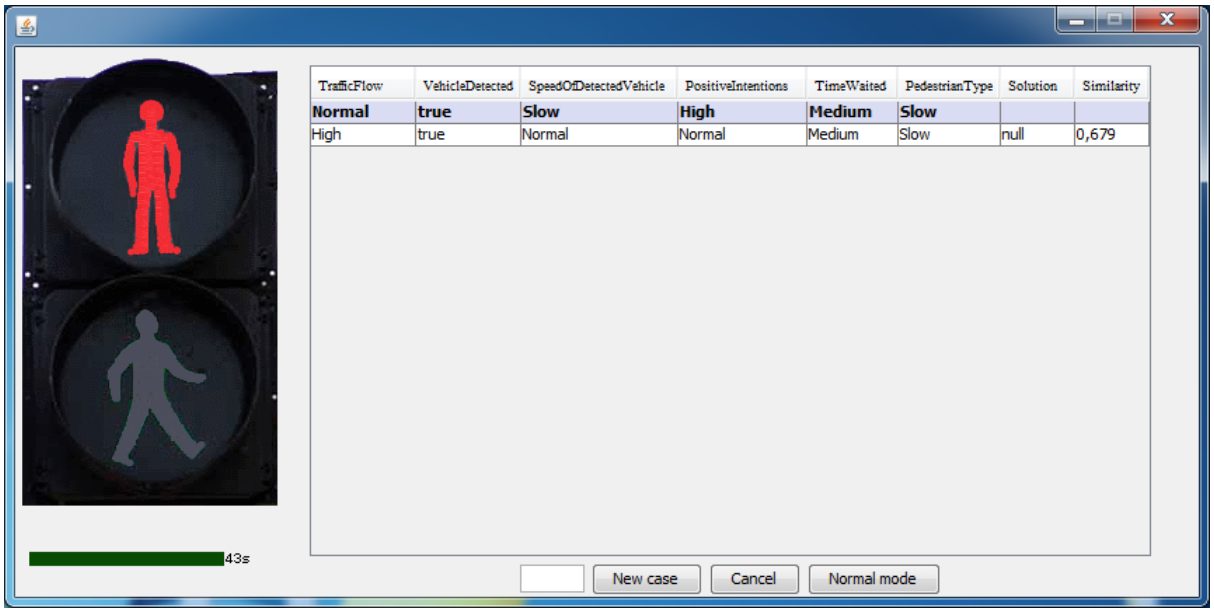

Figure 5.7: A screenshot of the user interface when the system is interrupted at a query.

the case base was improved in a systematical way, which contributed to make the cases is the case base more correct and more useful. After this process the case base contained 31 cases.

After running cross-validation tests on the system, we discovered some deficiencies in the case base. The first deficiency was that even though we had covered most of the problem domain, we did not have enough cases to fully represent the different parts of the problem domain. To get a good overview of the different parts of the case base, the cases were categorized by what types of situations their solution aims to solve. The categories are listed below:

- Normal situations
- Let pedestrians pass if there is a gap in traffic
- Pedestrians has not waited long enough
- Pedestrians has waited "Long" time
- A slow pedestrian wants to cross the road
- A slow group of pedestrians wants to cross the road
- A fast pedestrian wants to cross the road

- A fast vehicle approaches the crossing
- High traffic in the area around the crossing

By adding cases to categories that were not sufficiently covered, the system was less exposed to noise in the query cases. This also increased the accuracy in the cross-validation. After these cases were added, the case base contained 66 cases. To sum up, 21 cases were added before the meeting with the experts, 10 cases were added when using test queries generated from the data obtained at Brattøra, and 35 new cases were manually added to cover all categories.

Another thing that was discovered was that there are very strong dependencies between features in the cases. For example, the feature "Type of pedestrian" is of great importance in cases where the solution is not null (i.e. when the system allows pedestrians to cross), while if the solution is null, the feature is not that important. This makes the system more exposed to noise, because small changes in cases, can give very different solutions. In section 7.1,we propose a solution to this problem.

5.3 Features

In this section, we will describe the system acquires the knowledge for each of the features. First, we explain how the incoming data were transformed from continuous values to nominal values. Second, we describe how communication between bundles (applications in OSGi) is used for getting the data from the different sources. Section 5.3.3 describes how the Intention-based Sliding Doors system was integrated with the CBR system, to interpret the intention of pedestrians. Section 5.3.4 explains how information about the type of pedestrian was acquired by the system, before the last section is about how the system obtains the rest of the features.

5.3.1 Range of the features (Discretizing)

To make the process of retrieving similar cases easier, we discretized the features from continuous to nominal features. The range of the discrete features are shown in table 5.4. The traffic flow feature is split into low, normal and high. To define a range for these values, we used actual detector data from the crossing at Brattøra (see section 4.4.1).

The data consisted of the vehicles that passed the detectors, in both directions, from 13.02.2012 to 17.02.2012.

The range of the speed of detected vehicles feature was acquired by simply looking at the speed limit at the crossing. We found the range for the number of positive intentions by looking at the counting of pedestrians we conducted. The range of the time waited feature is based on the opinions of the experts from the NPRA. The type of pedestrian feature is a discrete value, and did not need to be translated.

The values of the solution feature was obtained by calculating the distance (7.5 meters at the crossing in Brattøra) from one side of the road to the other, and using the distance together with an estimated speed of pedestrians, to calculate the transition time. The speed for the different solutions has not been a target of research, but rather been based on intuition. For example, the normal speed is set to be 1.2 m/s, which is the speed used by the NPRA for normal pedestrian crossings. The speed for "LessLessTime" was set to 2.0 m/s because this is approximately the speed a slow runner will have.

It is important to point out that the transition time given by the solution does not correspond to the actual time the pedestrians will have to cross. This value is called the emptying time (*Norwegian:* Tømmingstid). It is the time that it takes to empty the crossing. To calculate the total transition time, the green time is calculated by dividing the emptying time by 2 and adding 2. The green blinking time is found by dividing the emptying time by 2. If the solution is to give "Normal" transition time, this would correspond to a green time of 5.125 seconds and a green blinking time of 3.125 seconds, which would result in a total transition time of 8.25 seconds.

5.3.2 Communication between bundles

In OSGi, bundles use something called a Service to export or import data from other bundles. A Service is a Java object instance, registered into the OSGi framework, with a set of properties. By publishing Service Events, bundles can let other applications use their data. This way of handling communication is one of the features of OSGi that enables the developer to create modular systems. This is very useful in the traffic domain. For example, if a bundle is created for interpreting the intention of pedestrians in an intersection, this information can be accessed by any number of bundles. In this way, useful information can be exported to several bundles in the system in a simple manner.

Feature	Range
Traffic flow (5 min. time interval)	0 - 34 = Low, 35 – 88 = Normal, 88+ = High
Vehicle detected	True, False
Speed of detected vehicle	no vehicle = null, 0 – 45 km/h = Slow, 46 – 55 km/h = Normal, 56 km/h + =Fast
Number of positive intentions	1 – 3 = Few, 4 – 7 = Normal, 8+ = High
Time waited	0 – 16s = Short, 17 – 50s = Medium, 50s+ = Long
Type of pedestrian	SlowGroup, Slow, Normal, Fast
Proposed transition time (Solution)	null = no transition, ExtraExtraTime = 12.0s, ExtraTime = 7.5s, Normal = 6.25s, LessTime = 5.0s, LessLessTime = 3.75s

Table 5.4: Range of features

5.3.3 Integrating the Intention-based Sliding Doors

This section describes how the Intention-based Sliding Doors system was integrated with our system. First, we will explain how the original system works and what changes were made to make it more suitable in the domain of traffic control. Next, we will show how we made use of the code in our system by integrating it with CVIS. Finally, we will present some of the problems that occurred during the implementation phase, along with some limitations in using the Intention-based Sliding Doors code with the Kinect device as a sensor.

Intention-based Sliding Doors overview

The framework made by Solem was originally intended to control sliding doors in a crowded environment. It consists of several classes, from recognizing and tracking users, interpreting the user's intention to controlling the sliding door. All though controlling a sliding door is quite different from controlling a pedestrian crossing, there are some similarities. In both cases, the task of the system is to only let the people that actually have an intention to enter/cross, to do so. The biggest difference is that if the system makes a mistake letting pedestrians cross the road; it can have severe consequences (both related to safety and efficiency in the crossing). Thus, the accuracy of such a system must

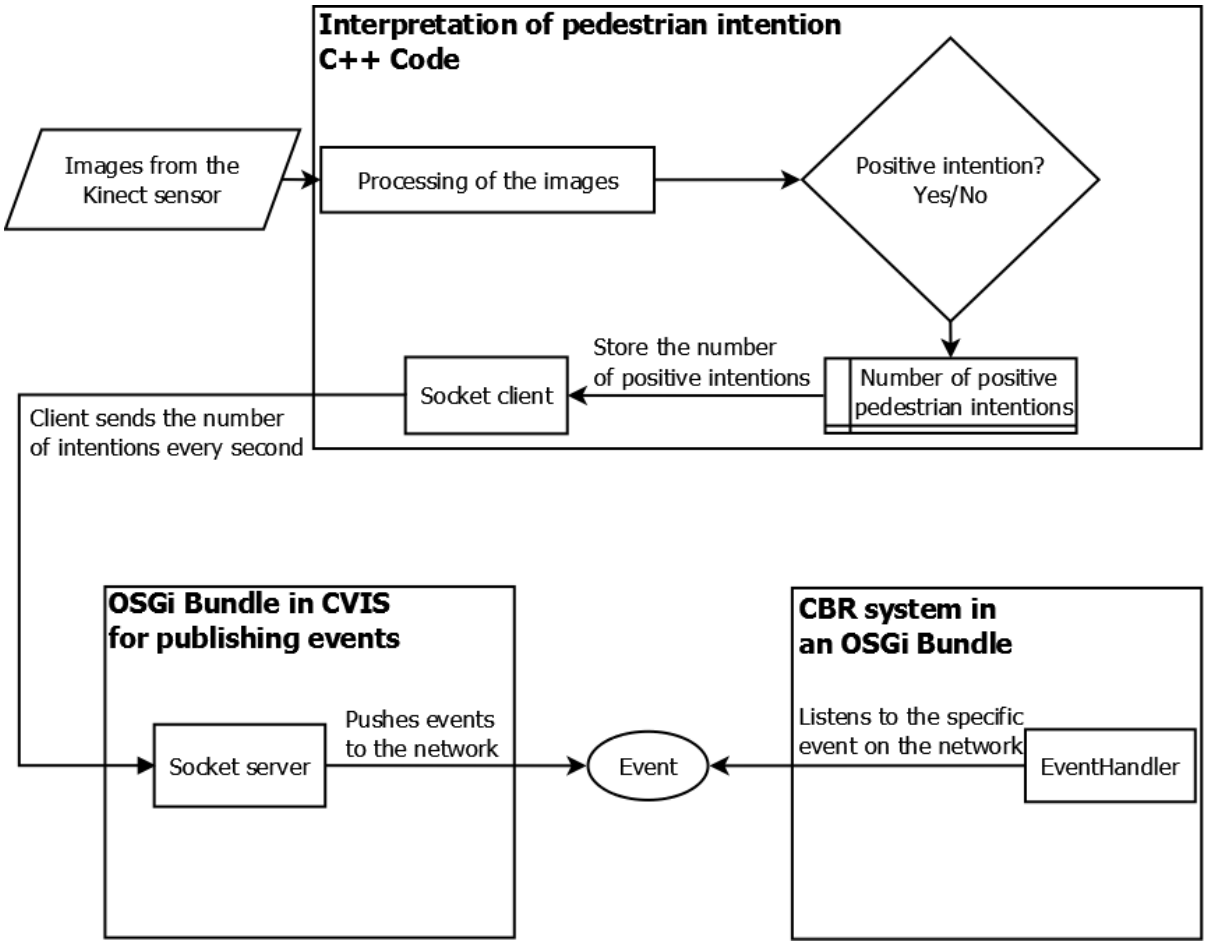

Figure 5.8: An overview of the sliding doors system, integrated with the CBR system

be close to perfect. There were also some other problems that had to be overcome:

- The Intention-based Sliding Doors is written in C++, so there was a need for communication between C++ and Java.
- An intention-based traffic light has to manage how many users have the intention to cross, while with sliding doors the system only need to know if one user has the intention to enter.
- The system for interpreting pedestrian intention has to know if there are no longer any pedestrians in sight, which is when the number of intentions to cross changes from one to zero.

There were multiple possibilities for transferring data from C++ to Java. We considered using something called JNI (Java Native Interface) where the C++ code is wrapped so that it is possible to call the C++ methods from Java. The positive side with this

approach would be that we could transfer data directly into the CBR system, because the C++ code could be a bundle on its own. The problem with the approach is that the Intention-based Sliding Doors use four large frameworks, for coding in C++ and for image processing with the Kinect device (the frameworks are described in section 4.5). Using JNI is quite complex even with simple code, and when using both more complex code and having to make it work with all four frameworks, we decided that wrapping the code was too much work, for little gain.

Instead, we decided to create a socket connection between the C++ code and the Java code. An overview of how the integration was done is shown in figure 5.8. The C++ code works as a client that always processes the incoming images, and sends all information through the socket, to an OSGi bundle. This bundle is not a part of the CBR system. The bundle publishes an event asynchronously, so that other bundles can subscribe to the stream of information. The advantage of receiving the pedestrian's intention in a separate bundle is that other bundles (i.e. other traffic applications) can use the information for other purposes. This makes it possible to use the information for purposes that might not be evident today. Our CBR system listens to the event by registering to it and creating an event handler.

The downside of this approach is that the data has to be sent from the C++ code through to an OSGi bundle, and from this bundle, to the CBR system. If we had wrapped the code directly into and OSGi bundle we would not need multiple transfers, but as mentioned before, the gain of wrapping the code was too small.

The next change that had to be made was to keep track of the pedestrians that have the intention to cross the road. Originally, the system does not support this feature, because a sliding door does not distinguish between the number of people that want to enter. When controlling a traffic light, this is actually a very interesting feature, since the number of pedestrians affects both the time needed to cross, and when the transition should be initiated. For that reason, we had to store the number of positive intentions. The system already created a User object for each human that enters the range of sensors. To keep track of the number of positive intentions, we stored the intention of the user in the User object, and counted the number of Users with positive intention, to get the number of positive intentions. This also solved our last problem, to keep control of when the number of positive intentions moves from one to zero.

5.3.4 Type of pedestrian

The type of pedestrian feature is an important feature, because it directly affects the transition time that should be given. Extracting this data from camera sensors would be the optimal solution. By looking at the height, speed and maybe other features like angles between different body parts, it is reasonable to believe that this feature can be obtained. The problem with this approach is that it is very dependent on good accuracy in the reasoning mechanism and that the sensors are maintained in a satisfactory manner. Also, how could this reasoning mechanism for example differentiate between a blind pedestrian and a normal pedestrian? A blind pedestrian will often use either a stick or a dog as an aid to be able to move around in the environment. The problem is that "normal" pedestrians can use a stick or walk a dog, without being blind. A solution to this problem could be to equip blind pedestrians with a device (e.g. a Smartphone) that could signal to the system that a blind pedestrian is within the range of the crossing.

Because of the time constraints of this project, we did not implement any of the ideas presented above. Essentially we wanted to create a module that made a simple interpretation of the type of pedestrian. For example, it would be possible to assume that a school class (i.e. a slow group) wants to cross the road, by checking if the heights of the pedestrians are below some threshold and if the number of pedestrians is above some threshold. The problem was that the Kinect sensor gives very variable results when the height of a person is calculated. We therefore decided not to implement this.

To be able to test the strengths of the system, without creating the module for interpreting type of pedestrian, we made it possible to send different types of pedestrians manually to the system. When the system is running, the user can either press "F" (fast pedestrian), "N" (normal pedestrian), "S" (slow pedestrian) or "G" (slow group of pedestrians). This adds a pedestrian with positive intention to the system, and gives it the type corresponding to the letter. This module is described in section 5.6.3.

5.3.5 Other features

The features describing the traffic situation (Traffic flow, Vehicle detected, Speed of detected vehicle) were all obtained from SCANeR Studio. Traffic flow was calculated by counting all vehicles that pass the pedestrians crossing within a given time interval. The same detectors (described in section 4.6.3) are used for detecting vehicles and providing

the speed of the passing vehicles. This information is passed to the CBR system by publishing CVIS events in SCANeR Studio. It is done in SCANeR Studio by using the simulators own scripting language. The last feature, time waited, is simply obtained by starting a timer every time a pedestrian has the intention to cross, and is reset if there are no longer any pedestrians that have the intention to cross.

Now that we have presented the key issues regarding generating a case base with cases and acquiring the data for the features, we will in the next section present how cases were retrieved by the system.

5.4 Retrieval

In this section, we will describe the retrieval process of the CBR system. This includes creating a query case, finding a set of similar cases, choosing the best solution of these and using the solution to perform traffic control in a pedestrian crossing.

5.4.1 Create a query case

If the system gets a positive intention from a pedestrian (i.e. the number of positive intentions goes from zero to one), the system will build a query case. It does this by collecting data from different detectors (described in section 5.3). This is illustrated in figure 5.9. If the data detected is changed, the query case will be updated, so that it will always contain the latest data.

A query case has to consist of nominal data, which means that incoming continuous data has to be converted. The conversion is done by taking each continuous value and looking it up in a list of ranges, where each range indicate a nominal value. For example, if the calculated traffic flow in the last five minutes is 100 vehicles, the value is converted into “High”, because the range is from 88 to infinity (for all ranges see section 5.3). After the query case is created, it is returned to the system.

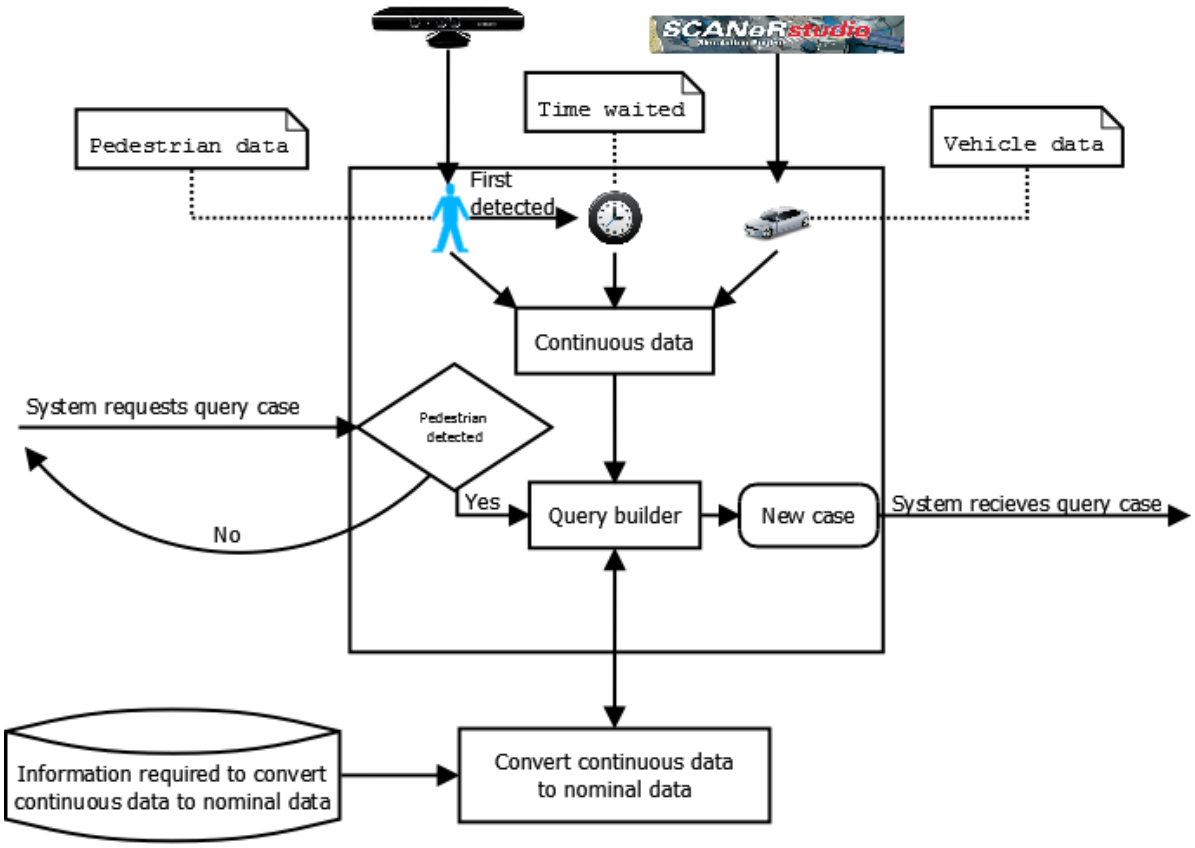

Figure 5.9: The process of building a new query case

5.4.2 Retrieving a set of cases

After the query case has been created, the system tries to find the cases that are most similar to the query case. The similarity functions were described in section 5.2. These indicate the local similarity between each feature. The similarity between two cases is then established by taking all the local similarities and combining them into a global similarity. Similarity is also affected by the importance of each feature (the weights of the features). For example, if a feature is weighted 0.5 and another feature is weighted 1.0, the second feature will count twice as much as the first feature. Once the similarities between the query case and each case in the case base have been calculated, a set of the cases with the highest similarity is formed.

5.4.3 Choosing the most similar case

If more than one case was retrieved, the best solution has to be chosen. It is done by taking the solution that is most frequent in the set of cases. The problem with this approach can e.g. occur if two cases in the set give conflicting solutions. Then the system randomly chooses one of the solutions. This increases the possibility of the system making mistakes.

A possible solution to this problem would be to always choose the solution that gives the longest transition time, if the two solutions are equally frequent. This would make the system less efficient, but it would ensure that the system would not give too short transition time for some pedestrians. The reason why we did not implement this was because these events rarely happen. If it were to happen, the reason for it would be that there are cases in the case base that have the wrong solution or that a case is missing. In a final version of the system, a safety mechanism with methods to handle these situations would be necessary.

5.4.4 Using the solution

The solution of a case is either a transition time or that a transition should not be executed. If a transition should not be executed, the process will start from the beginning with creating a new query case. If a transition time is returned, the nominal value (e.g. “Normal” or “LessTime”) is converted into seconds by looking it up in a table (the table

used in the system was shown in table 5.4 on page 56). This table can vary between crossings, since there will be a difference between the norms in different crossings (e.g. the number of vehicles that pass a crossing can be much higher in some areas). These seconds are then passed to the simulated traffic light, which will change its signal. While the traffic light gives a green light to pedestrians, the system stops the generation of a query case, but when the light is red again, the process restarts.

5.5 System description

In this section, we will give a more thorough description of the system. First, we will describe the classes of the system, and then give a short walkthrough of the system.

5.5.1 Class diagram

A class diagram of the complete system is shown in figure 5.10. The main class in the system is the Activator, where all the communication goes through. The system consists of four parts; TrafficSituationCase to store all data about a traffic situation, the TrafficLightSimulator to handle the graphics, the QueryBuilderClass to create a query case and the Cbr class to handle retrieval and reuse of cases. The classes ReadTrafficData and AddToXml will not be described, and we refer to the Javadoc for more information about the functionality they provide.

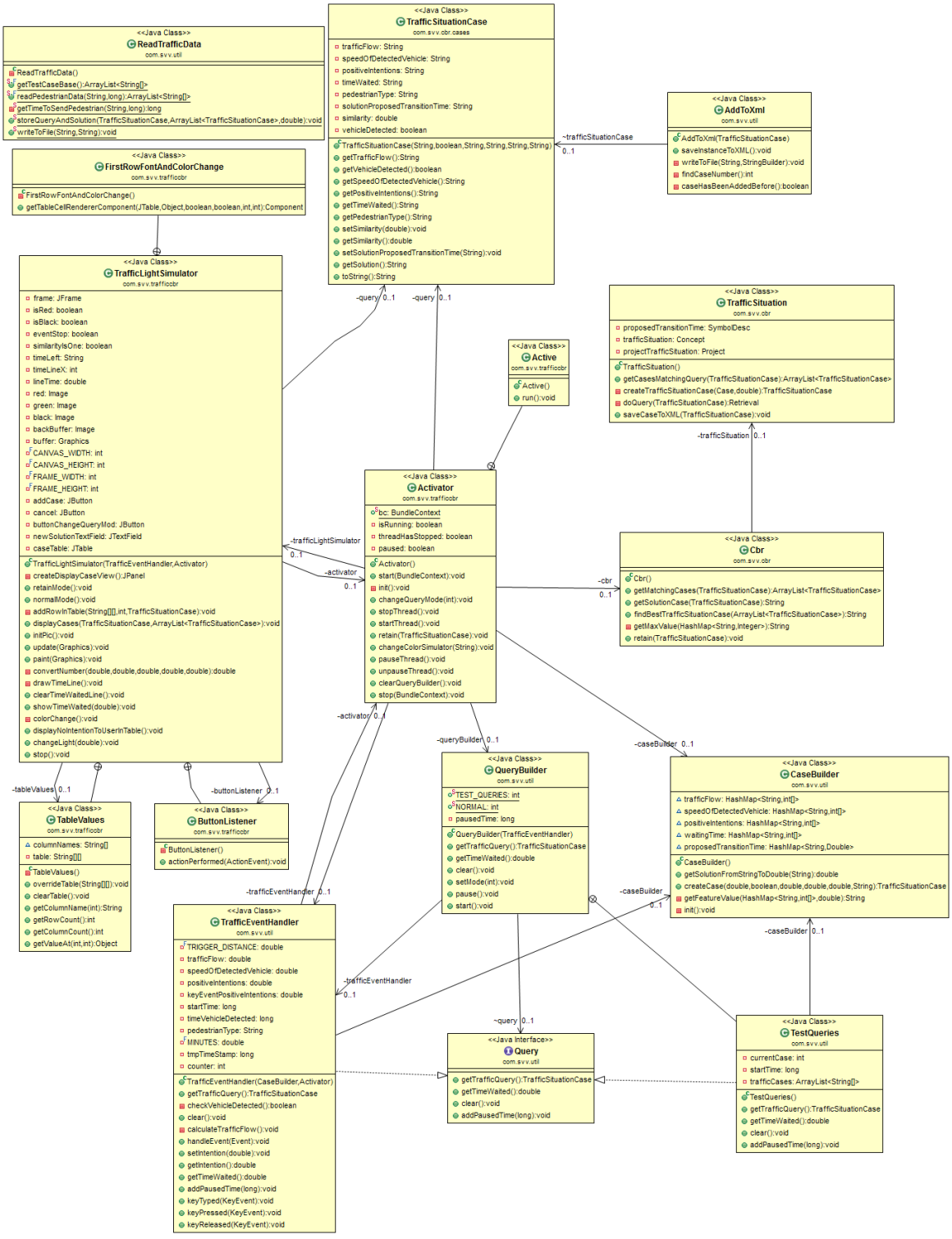

Figure 5.10: Class diagram of the CBR system

5.5.2 Description of the classes

The following sections describe some of the most important classes in the system.

Activator

Figure 5.11 shows the main class in the CBR system, the Activator class. This class is the standard main class of OSGi, where a start and a stop method are used to control what is done at startup and shutdown of the system, respectively. The class initializes two threads, one is the TrafficLightSimulator class that visualizes a traffic light and the other is the Active class that runs as long as the variable "isRunning" is true. The Active class will constantly request a new query case, but a new query case is only created if one or more pedestrians have the intention to cross the road. The Activator class also contains methods for pausing and unpausing the Active class, so that the user can halt the system if needed. When the active class has gotten a transition time, it is forwarded to the TrafficLightSimulator class running in the other thread, which uses it to change the traffic light in the simulator.

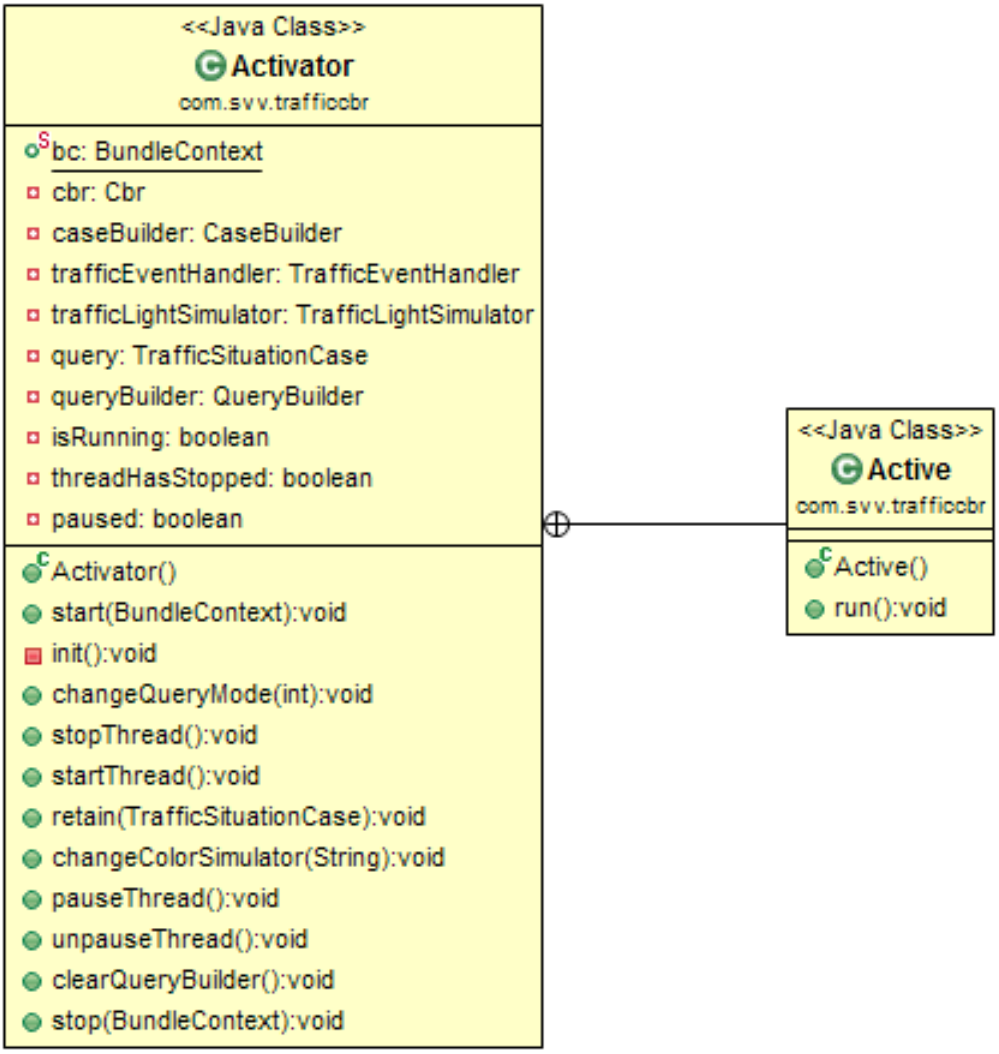

Figure 5.11: Activator class

TrafficSituationCase

The TrafficSituationCase class (see figure 5.12) represents a traffic situation case, with the features described in section 5.2. The variables similarity and solutions are optional (not initialized in the constructor), and can be used to store the similarity between a case and the query case, and a solution that is proposed by a case, respectively.

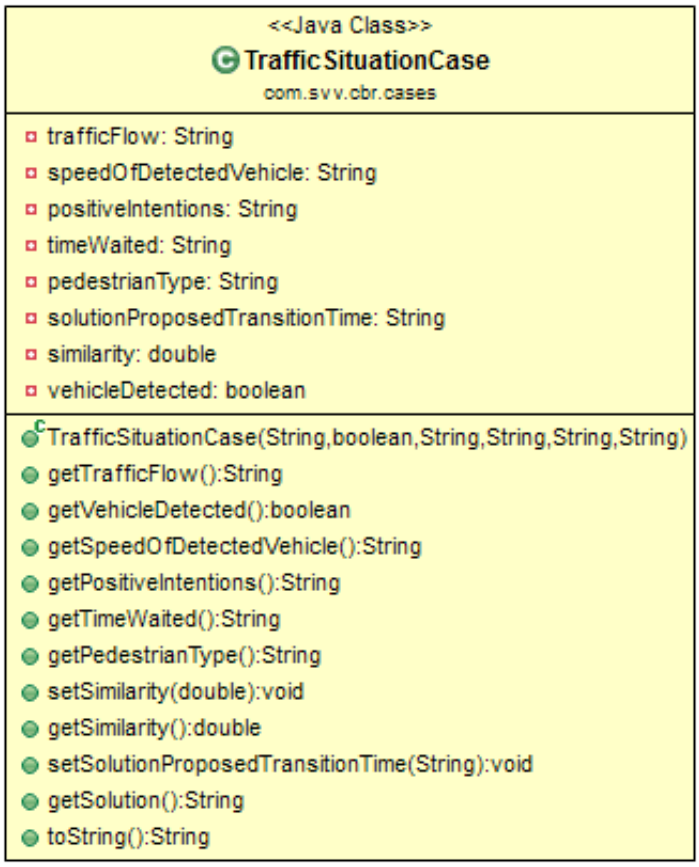

Figure 5.12: TrafficSituationCase class

QueryBuilder, TestQueries and TrafficEventHandler

Figure 5.13 shows a class diagram with the classes QueryBuilder, TestQueries and TrafficEventHandler. These are classes that are used in the generation of a new query case.

The QueryBuilder is used to generate queries, and can be run in either test mode (TestQueries class) or normal mode (TrafficEventHandler class). Both of the classes implement the Query interface, which include methods that are required by the QueryBuilder. The TestQuery class is only used in the test phase. It reads test cases from a file, and returns these cases when the getTrafficQuery method is called.

The TrafficEventHandler class receives vehicle and pedestrian data through Service events (see section 2.3). The new data that come from the events are locally stored in the class. When the method getTrafficQuery is called, the stored data is used to generate a TrafficSituationCase object, which is returned if the number of positive pedestrian intentions is greater than zero. There is a communication between the TrafficEventHandler and the Activator, even when the TEST_QUERY mode runs. This is because the KeyEventListener is inside the EventHandler, which is used to deactivate and activate the main thread. Different buttons are bound to send different pedestrians e.g. 's' = slow and 'f' = fast, and this is the reason why the KeyEventListener is inside this class.

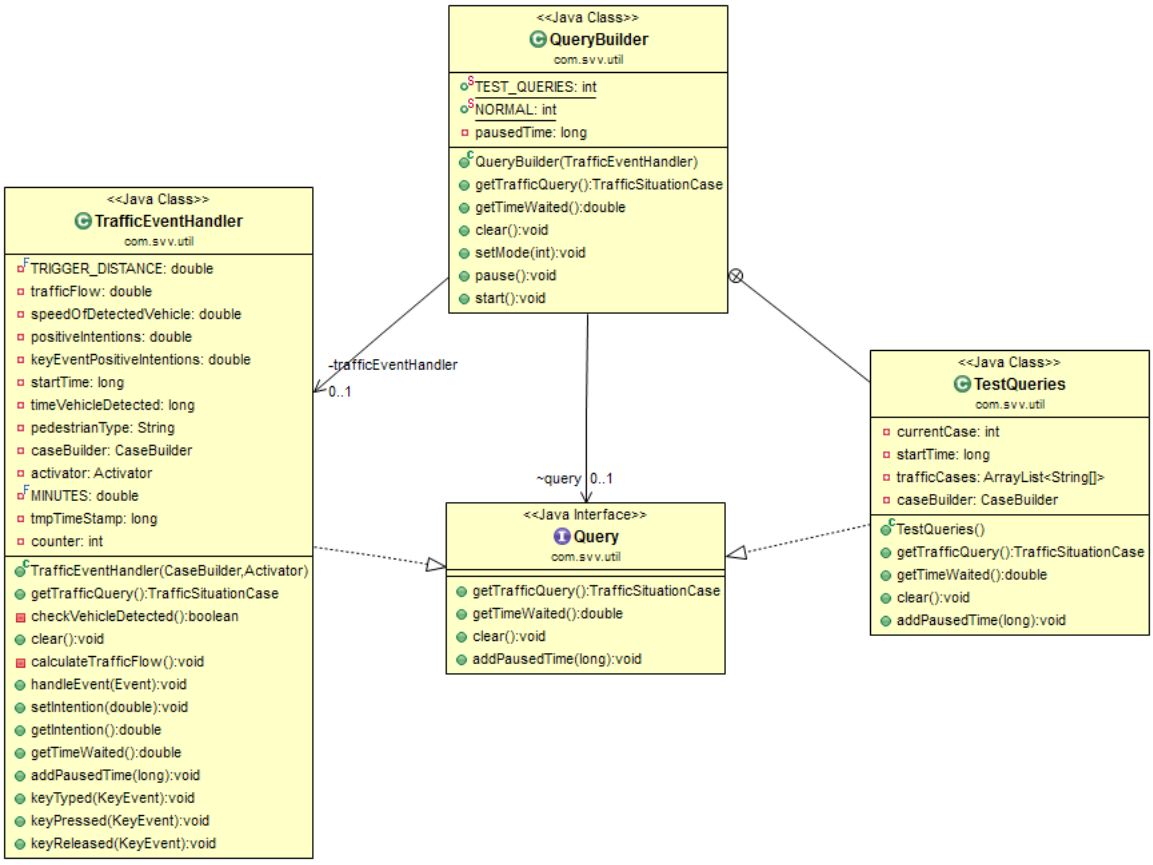

Figure 5.13: QueryBuilder, TestQueries and TrafficEventHandler class

TrafficSituation and Cbr

Figure 5.14 shows a class diagram containing the TrafficSituation class and the Cbr class. These are used for retrieving and reusing cases from the case base. The TrafficSituation class uses the myCBR tool to load the CBR project that was created in Protégé. It contains two public methods; one that retrieves cases from the case base and returns the cases with the highest similarity, and the other method to store a case to the XML file representing the case base. The Cbr class uses methods from the TrafficSituation class, to find the best case out of a set of cases that are similar to the query case.

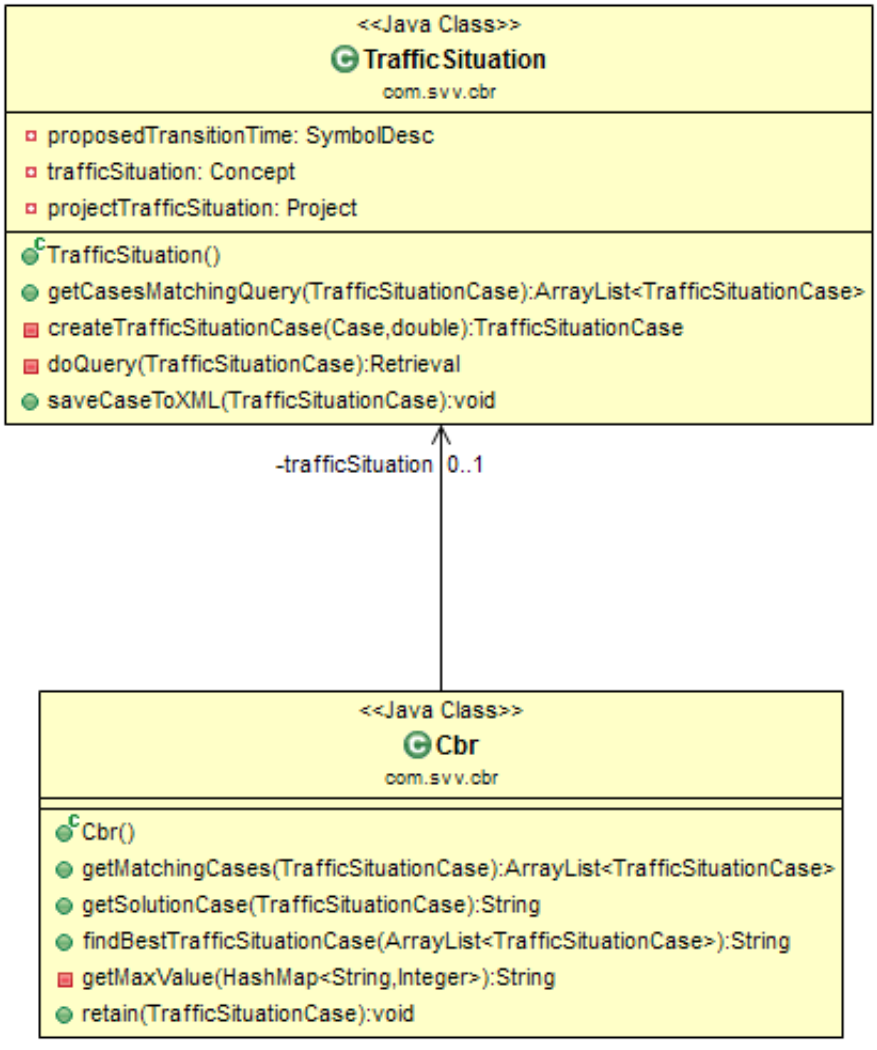

Figure 5.14: TrafficSituation and Cbr class

CaseBuilder

The CaseBuilder class, shown in figure 5.15, is used for converting data between continuous data, that is stored in a TrafficSituationCase object, to nominal data. The ranges of the nominal values are stored in HashMap's in the class. The class also contains a method for converting the solution (which is a String value) to a double value, so that it can be used by the traffic light.

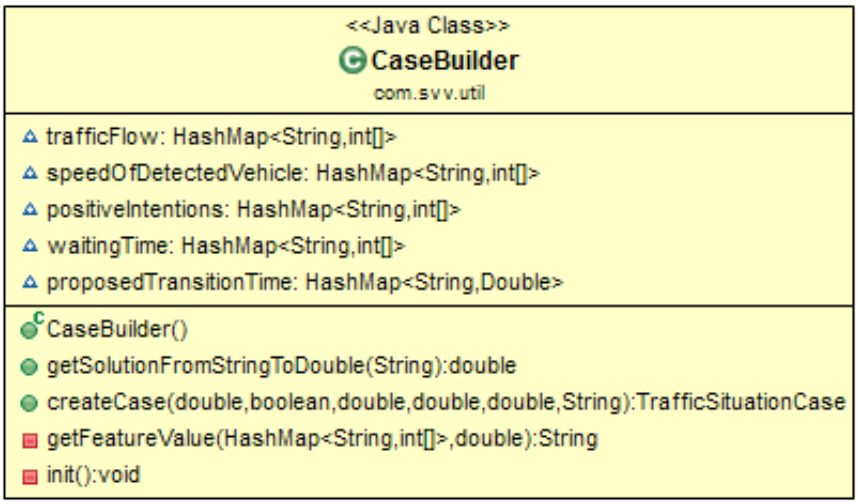

Figure 5.15: CaseBuilder class

TrafficLightSimulator

The TrafficLightSimulator class is shown in figure 5.16. It is used for creating a graphical version of a traffic light, to visualize what the pedestrians would see in reality. When the Activator class receives a solution, the changeLight method is called. This method will simulate how the traffic light changes the light, with the green time being based on the input value in the changeLight method. To change the traffic light in SCANeR Studio, the changeColorSimulator in the Activator class is used.

The class also contains the rest of the graphics like; the visualization of the current case, the matching cases and how long a pedestrian has waited. The two classes; TableValues and FirstRowFontAndColorChange, are used for updating and configuring the table, which displays the query case and the cases with highest similarity in the user interface.

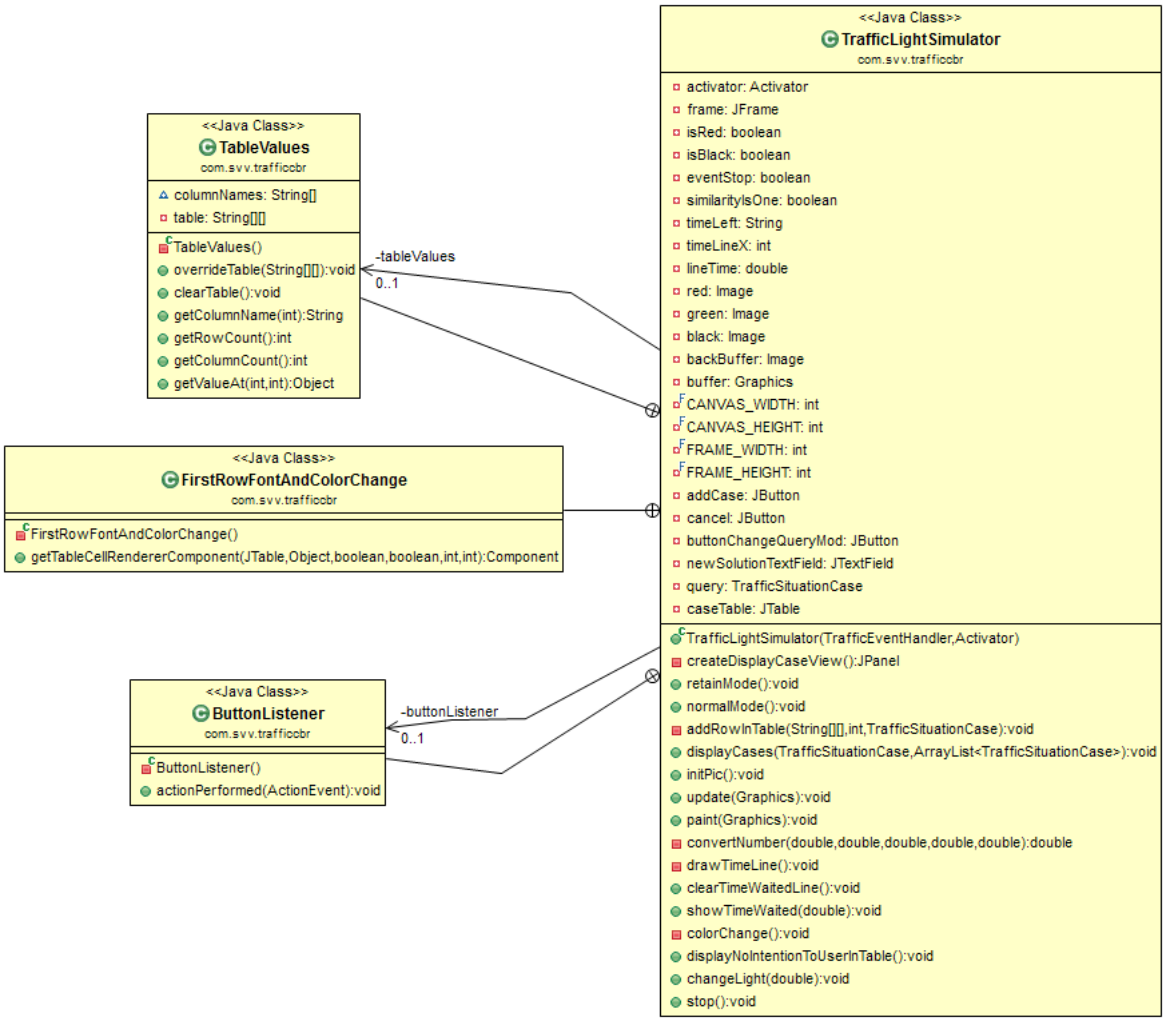

Figure 5.16: TrafficLightSimulator class

5.5.3 Class diagram for the interpretation of pedestrian intention module

In this section, we will describe the classes of the interpretation of pedestrian intention module. We have excluded most of the classes created by Solem in the Intention-Based Sliding Doors project, because most of the changes that have been done are in the class UserHandler. A class diagram for the Java code is shown in figure 5.17 and the class diagram for the C++ code is shown in figure 5.18. As mentioned before, the communication between the C++ code and the Java code goes through a simple socket connection. The method inferUsersIntentions, in the UserHandler class, has been modified to send the inferred intention of pedestrians through a socket connection. It calls the method getAllUsersIntention, which counts the number of pedestrians that have the intention to cross the road. The information is sent through the socket connection by calling the method SendInts in the Client class.

The Java code runs as an OSGi bundle. The reason for this is that it makes it easier to forward the data to the CBR system. In the Activator class, a socket server is run from the start method (the initial method running in an OSGi bundle) in a new thread. It has to be run in a separate thread, or else the bundle will never stop running the start method, and it will never get the status as "Active" (the different statuses of OSGi is explained in section 2.3). The Server class contains a run method that listens for Clients. If a Client wants to connect, the WorkerRunnable class is run in a separate thread. This enables the server to accept multiple clients connecting to it. The WorkerRunnable class handles the input and output of the client.

When a Client has connected to the server, it will immediately start sending the number of positive intentions to the Server. The method RecvInts converts the data from binary to integer, and returns the number of positive intentions. If the returned data is valid, the data is sent by registering a service in OSGi, and publishing it as an event, to make it available for other bundles.

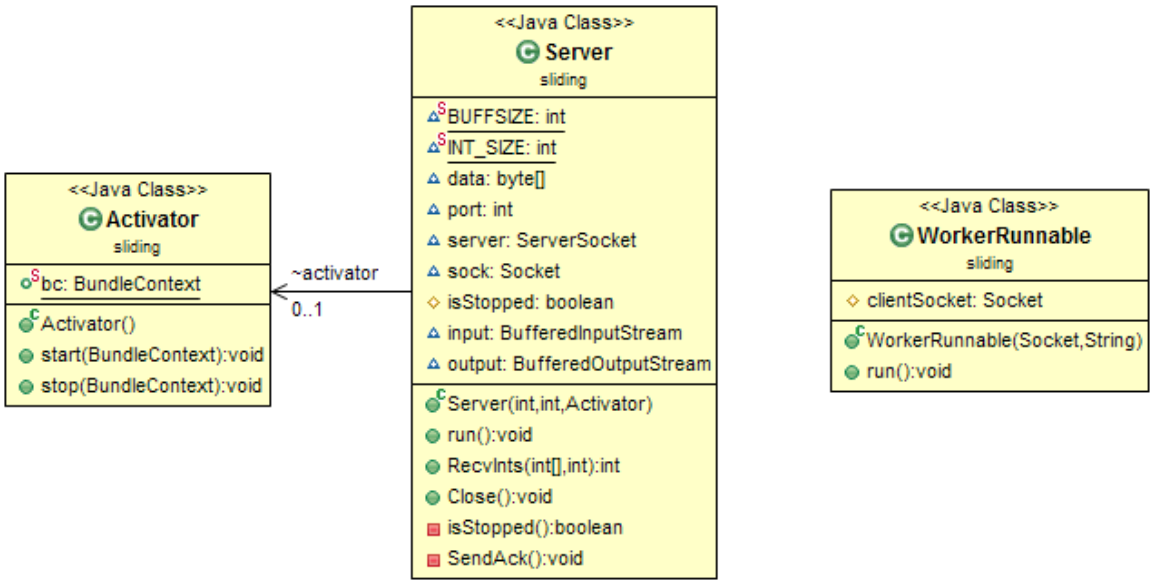

Figure 5.17: Class diagram for the interpretation of pedestrian intention module

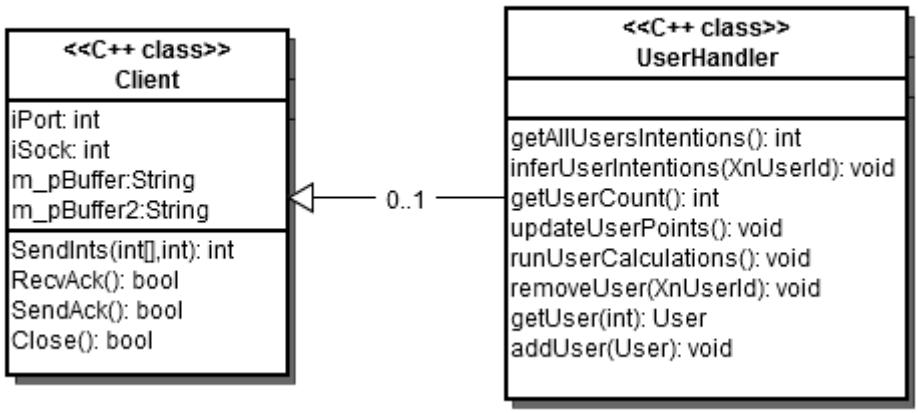

Figure 5.18: Class diagram for the C++ code

5.5.4 Walkthrough of the system

The CBR system consist of three parts, where each part runs on different computers (as shown in figure 5.19). One of the computers runs the system for interpreting pedestrian intention. It receives images data from the Kinect sensor, and sends the number of positive intentions to another computer, which runs the CBR system. The last computer runs SCANeR Studio that simulates the traffic, which got a two-way communication with the CBR system. The SCANeR Studio sends information about the vehicles and receives the status of the traffic light.

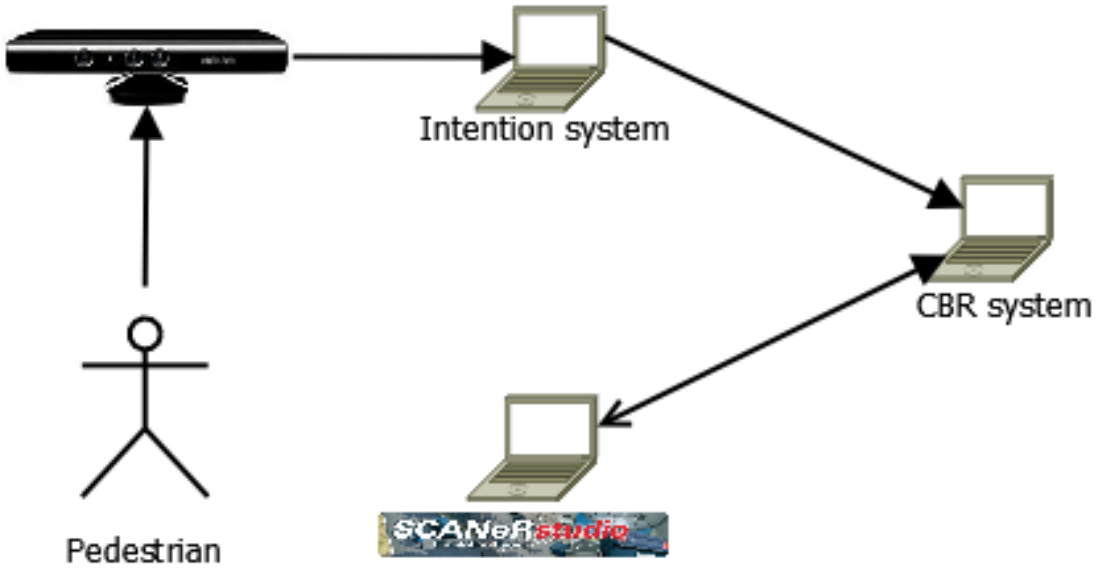

Figure 5.19: An overview of the communication between the three systems

When the CBR system starts, the user is presented with a user interface. Figure 5.20 shows the user interface, when no positive intentions from pedestrians has been detected. On the left, there is a graphical version of a traffic light, which will change according to the solutions given by the system. On the right, there is a table that will list the query case (second row) and all retrieved cases (remaining rows). In the bottom of the user interface, there are buttons for changing the mode of the system and for adding cases when the system is running. The buttons are only activated if the user pauses the system, which can be done by pressing the letter “p”. The cancel button will re-activate the system. The “Test mode” button activates a system that loads pre-defined queries from a file into the system.

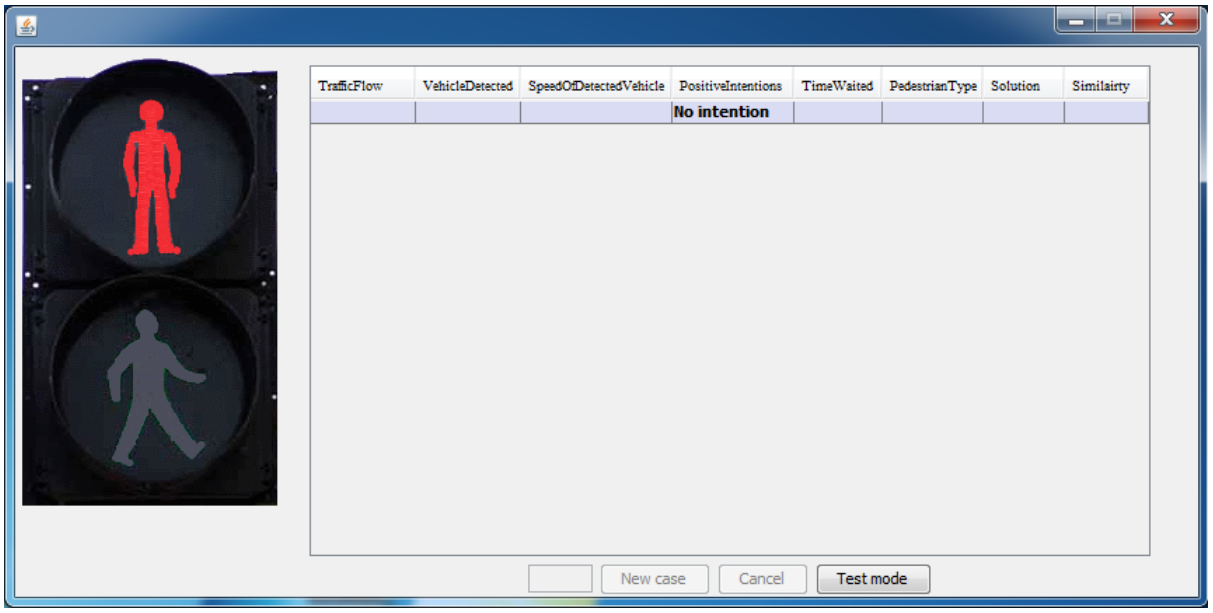

Figure 5.20: The CBR system indicate that no intention is detected

To trigger the traffic light, a positive intention is required, which is interpreted by the intention system. A positive intention is a pedestrian that is facing directly or heading against the Kinect sensor. Figure 5.21 shows the Kinect sensor installed on top of the traffic light. The traffic light is controlled in parallel with the simulated traffic light, and the traffic light in SCANeR Studio.

Figure 5.21: The Kinect sensor installed on top of the traffic light

Figure 5.22 shows a screenshot from the intention system (user interface is created by Solem [3]). In the top left corner, the number of pedestrians detected is shown (users). The rest of the screen shows detected pedestrians with a grid representing important points on the pedestrian's body, which are used for calculating the intention of pedestrians. The number of positive intentions is passed to the CBR system, and used as a feature in the query case. If a pedestrian with positive intention disappears from the sensors range, that intention is removed from the number of positive intentions.

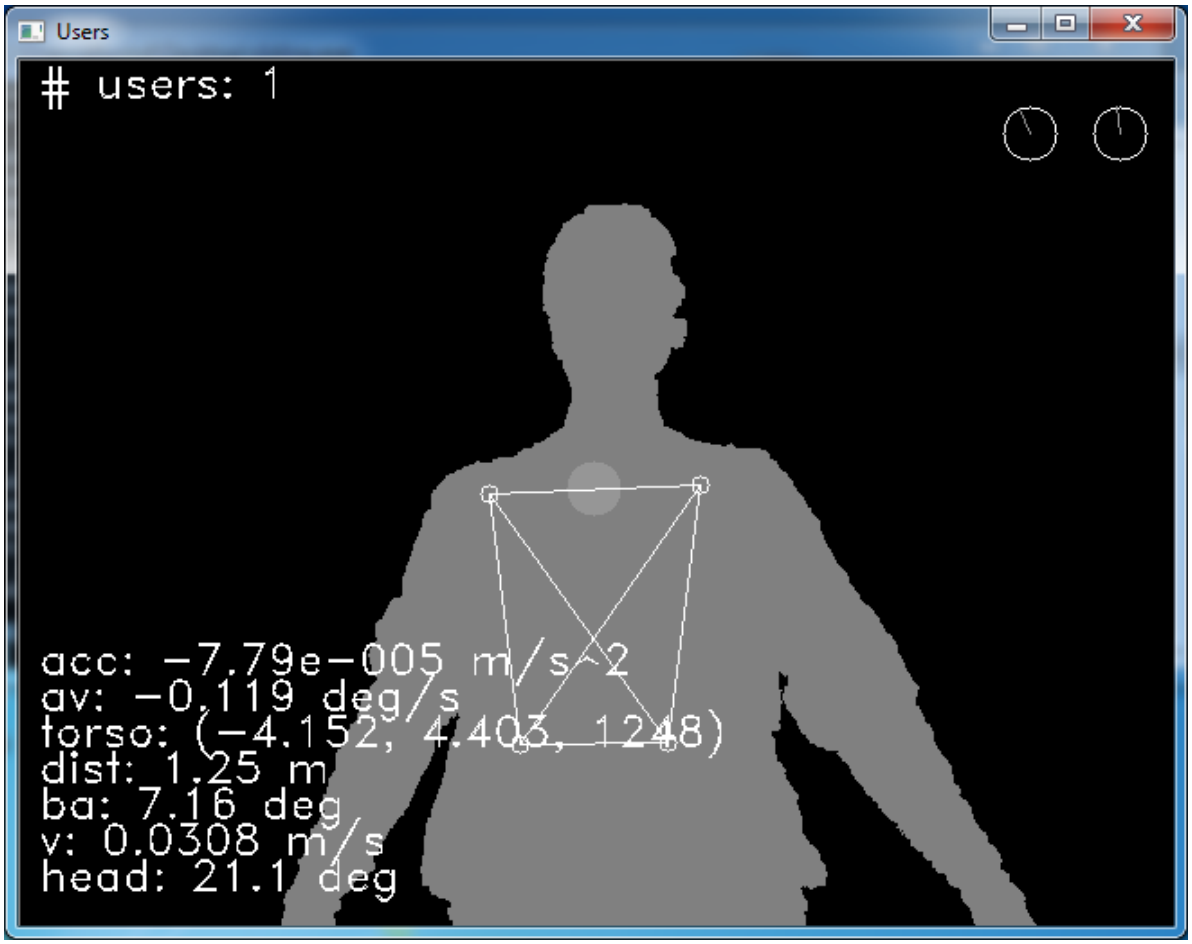

Figure 5.22: A screenshot from the intention-based system

When a positive intention comes from the intention system, the CBR system creates a timestamp for when the detection was made. This time is used to calculate the waiting time, as shown in the bottom left corner of figure 5.23. "3s" means that the first pedestrian has waited 3 seconds and a green progress bar illustrates the time waited.

In figure 5.23, a positive intention has been detected, and is sent to the CBR system and stored as the feature “PositiveIntentions”. If it is the first positive intention, the system will set the timestamp for the “TimeWaited” feature to be the current time. If more pedestrians walk in front of the Kinect sensor, the number of positive intentions will increase, but the timestamp will remain unchanged.

If the detectors in SCANeR Studio are triggered by a vehicle, the speed of the detected vehicle is sent to the system. This speed is used directly to set the feature “SpeedOfDetectedVehicle” and a timestamp for when the vehicle was detected is set. The timestamp is used for calculating when the vehicle has passed the crossing, by using the distance

from the detector to the crossing and the speed of the vehicle. When the time calculated time has ended, the "VehicleDetected" feature is set to "null", if no other vehicle has been detected in the meantime. Also, every vehicle that passes the detectors in SCANeR Studio are counted and used to calculate the number of vehicles the last five minutes. This value is used by the feature "TrafficFlow". As mentioned before, the module for interpreting the type of the pedestrian has not been created, for that reason the feature "PedestrianType" is predefined to normal when a pedestrian is detected by the Kinect. All these features are converted to continuous data, and represented as the query case shown in figure 5.24 in the second row of the table.

The system gets the query case, which is used to retrieve the cases with the highest similarity. The third row of the table on figure 5.23, displays a retrieved case, with similarity 1.0. This similarity is calculated by the weighted sum of all the cases features, and is 1.0 because all the features are equal with the query case. Since the retrieved case has the solution "null", it will not change the traffic light, and therefore continue the reasoning process. This query case will now contain the latest data, because the sensor data stored locally in the CBR system is always updated.

In figure 5.25, it can be seen that the query case has changed. The traffic flow is now calculated to "Low" and no vehicle is detected. Now the system retrieves two new cases from the case base, shown in the third and the fourth row of the figure. Both cases were retrieved because they got an equal similarity of 0.986. Only the traffic flow is different between the query case and the retrieved cases. As can be seen in table 5.2 on page 48, the similarity between low and normal traffic flow, and between high and normal traffic flow, is zero. Therefore, both cases get equal similarity. The reason why the similarity is as high as 0.986, even if one feature is not equal, is because the weight for the "TrafficFlow" feature is only 0.5 (see table 5.3 on page 51).

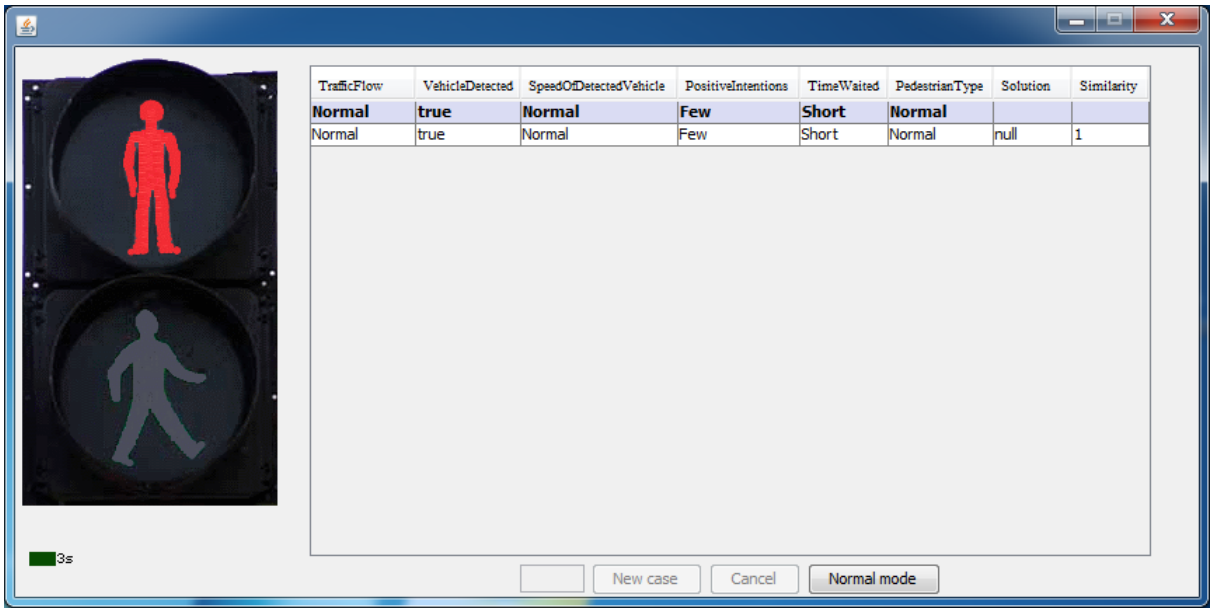

Figure 5.23: A positive intention has been detected, and pedestrians are waiting

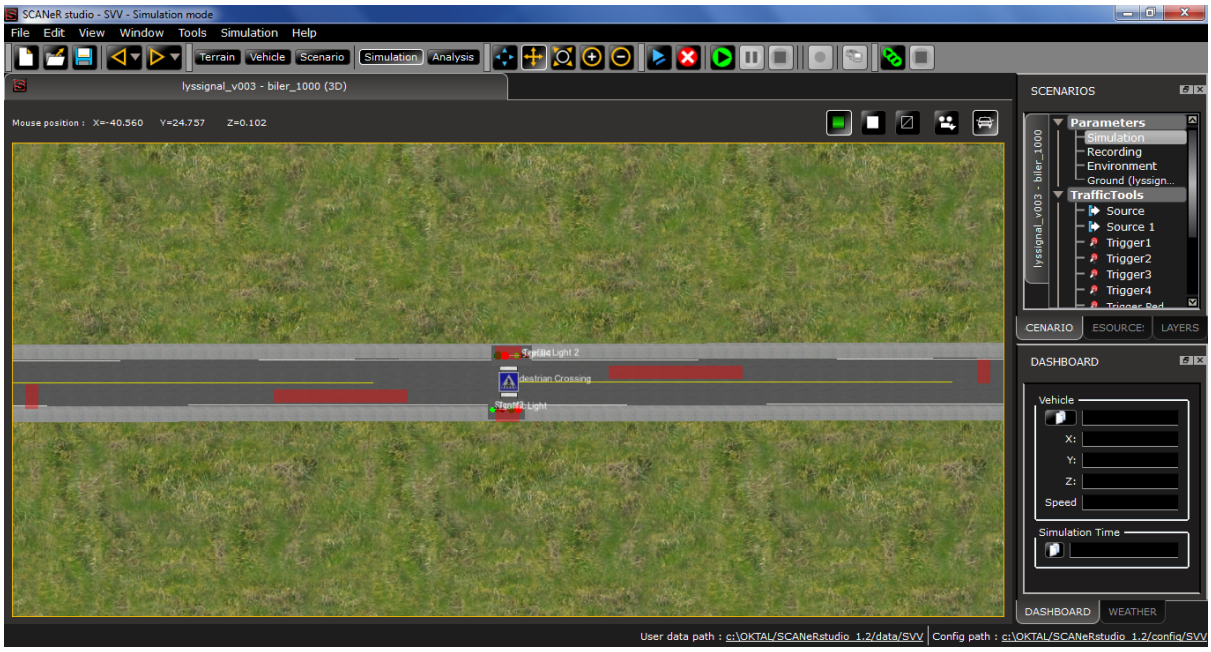

Figure 5.24: A screenshot from the SCANeR Studio, which shows the traffic light that is controlled by the CBR system.

Since both solutions give "LessTime" as transition time, this solution is chosen. This number is used by the simulated traffic light to give pedestrians a green light. The number below the simulated traffic light in the figure now indicates how long the traffic light has

left of the green time. The system also publishes an OSGi event, which is listened to by a script created in SCANeR Studio. This event tells the simulator to change the traffic light. The vehicles in SCANeR Studio will stop when the traffic light is red or amber, and start again when the traffic light turns green. Then the CBR system will continue to retrieve cases to solve new situations.

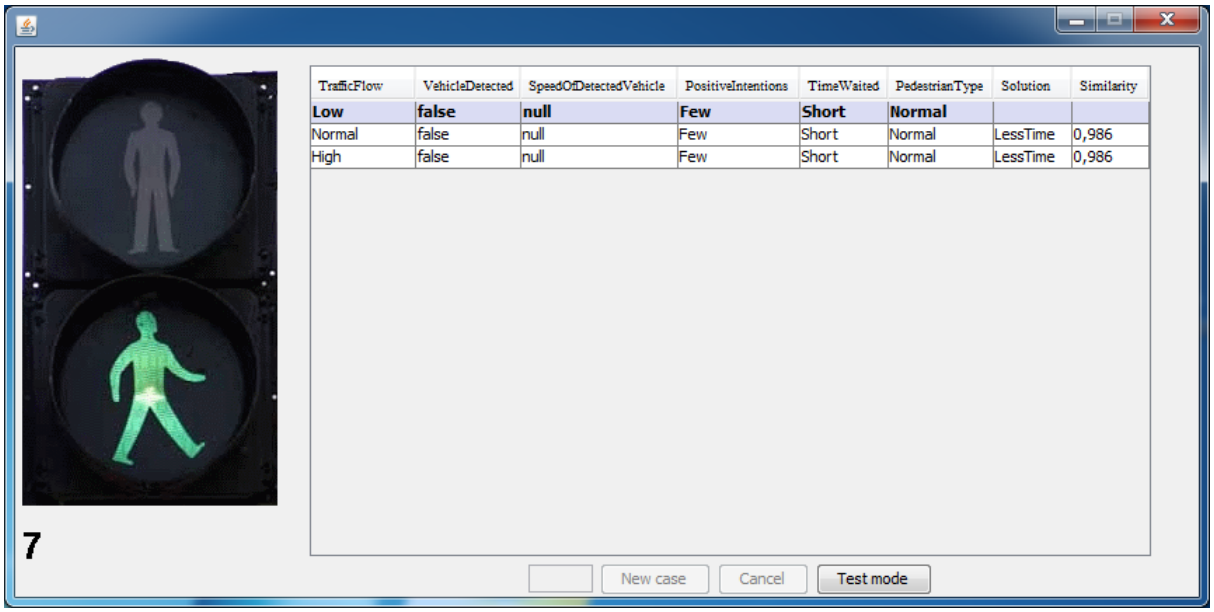

Figure 5.25: The CBR system gives a green light to pedestrians

5.6 Modules created for evaluation purposes

In this section, we describe the modules that were created for evaluation purposes only.

5.6.1 Adding cases real-time

To make the process of building the case base more systematic, we created a module for editing the case base. An optimal way would be to be able to add, edit and remove cases from the case base in real-time. It would also be useful to be able to change the weights of the features, without having to manually edit the XML files representing the case base. The problem was that myCBR 3.0 BETA does not support changing the XML files. The methods are included in their Javadoc, but they are not yet implemented. As a result of this, we had to create our own XML parsers for dealing with this issue. Editing and removing cases requires searching through the XML files, and removing or editing the correct parts. Since we already had problems with the XML files being corrupt when creating them in Protégé, we chose not to make it possible to edit or remove cases through the program.

A module for adding cases to the case base was created. When the system runs, the expert (or other users) can deactivate the system if it makes a mistake (i.e. retrieves a case incorrectly or if no case is retrieved at all). Deactivating the system also gives the expert time to evaluate the retrieved cases, and stores the query case if he or she thinks it is of value to the system. It is done by providing a solution to the query, and then re-activating the system. The system edits the XML files to include the new case.

5.6.2 Leave-one-out cross-validation

A separate system was created for cross-validating the system. Instead of building a query case from vehicle- and pedestrian data, the cases in the case base are used as query cases. Since we used leave-one-out cross-validation, the query case was simply chosen by taking the first case in the case base, removing it, and using it as a query to the system (without the solution). The solution of the query case was used as a reference to validate the performance. After validating the performance of the single query case, the case is put back into the case base, and the next case is used as a query to the system. This continues until all cases have been used as a query case

We defined two evaluation parameters, accuracy and partial accuracy. Accuracy is directly derived from the number of correctly classified cases. Partial accuracy is the number of correctly classified cases, plus the number of partially matching cases. Here, we give partial credit to solutions that give one step longer transition time than the correct solution (e.g "ExtraExtraTime" is given partial credit if "ExtraTime" is the correct solution). It is reasonable to believe that this parameter can give a more correct indication of how well the system would perform. The results of the cross validation can be found in section 6.1.1

5.6.3 Module for sending pedestrians into the CBR system

In the performance test with SCANeR Studio, described in section 4.6.3, pedestrians are sent to the CBR system through a module that reads pedestrians from a file. The file consists of two columns; the time at which the pedestrian should enter the traffic light (i.e. 00:04:00 will send a pedestrian after four minutes) and the type of the pedestrian, which is either 'N' for normal pedestrian, 'S' for slow pedestrian, 'SG' for slow group or 'F' for fast pedestrian. Each line of the file is one pedestrian.

The module runs in a separate bundle that sends the pedestrians, and the slowest type of pedestrian, to the CBR system. It runs in an infinite loop that checks the time passed against the time for when the next pedestrian should be sent. If the time has passed, the number of pedestrians are increased, and sent to the CBR system.

When the CBR system gives transition time, a value with the traffic light color is sent to SCANeR Studio through CVIS. This value is also readable by the send pedestrian module, which uses this value to clear the number of pedestrians that are waiting. It is cleared when the traffic light is no longer green (i.e. when it turns amber for vehicles). This makes the scenarios more realistic, since pedestrians that enter the scene when the light is green (for pedestrians) will cross the road.

Chapter 6

Evaluating the system

The CBR system was evaluated both in terms of usefullness in the domain and in terms of how accurate it is. Section 6.1 present the results of the cross-validation tests and the performance tests in SCANeR Studio. In section 6.2 we discuss the results in terms of how good the system performed and if the results are valid.

6.1 Results

In this section, we present the results from the cross-validation tests and the performance tests using SCANeR Studio.

6.1.1 Cross-validation

Cross-validation was used to evaluate the performance of the case base. The initial case base, on which we performed cross-validation tests, contained 31 cases (see section 5.2.5). The performance from the cross-validation test showed an accuracy of 45.2% correctly classified cases. This wasn't a satisfying result, and it indicated that the case base did not have enough cases to cover the whole problem domain. To improve the accuracy, more cases were added to the parts of the solution space that had few cases. When new cases were added, cross-validation was used on the case base. How the different case bases preformed are shown in the diagram in figure 6.1. The accuracy rapidly increased when the case base was small, but slowed down when the size of the case base increased. The final case base, with 66 cases, has an accuracy of 75.8%. The reason we stopped at this percentage was explained in section 4.6.2. Average similarity among the retrieved cases

was 0.9642, which indicates most of the retrieved cases were close to the query case. The resulting case base was used in the simulator tests, which we will now present.

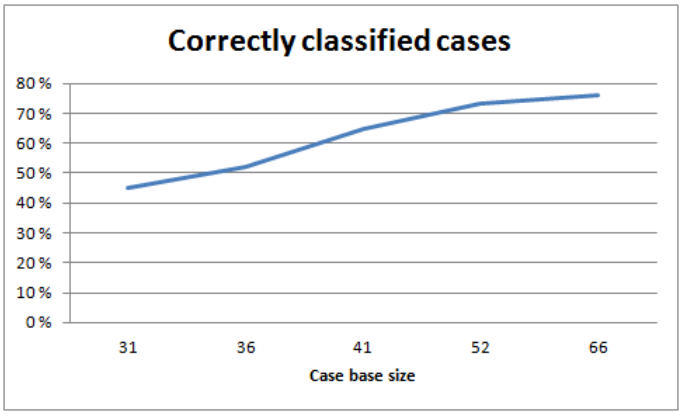

Figure 6.1: Number of correctly classified cases, relative to the case base size

6.1.2 Simulator tests

In the simulator tests, our CBR controlled traffic light and a normal time controlled traffic light were tested against each other in different test scenarios. Both of the systems were run in the same environment, with no random variables, so that the results could be evaluated against each other. The scenarios can be split into three main groups; with "Low", "Normal" or "High" traffic flow. A total of 7 test scenarios were conducted to evaluate the system. The properties of the normal traffic light is presented in table 6.1.

Interval	**Normal traffic light**
Green time	8.25 seconds
Amber time	3 seconds
Change time	30 seconds

Table 6.1: Properties of the normal traffic light that is used as comparison to the CBR controlled traffic light

Scenarios 1 to 5 were tested with "High" traffic flow, test number 6 with "Normal" traffic flow and test number 7 with "Low" traffic flow. The reason why we got five test scenarios with "High" traffic flow is that we wanted to see when one of the traffic lights would start to cause traffic congestions.

To evaluate whether the system performs better, we defined several criteria which can be seen as important for evaluating the safety and efficiency of the crossing. The criteria, along with explanations of why we chose them and what they test, are shown in table 6.2 and 6.3. "Two or less vehicles delayed by traffic light" and "No vehicles delayed by traffic" are only used when number of vehicles on the road is low or medium. These are not included in the test scenarios with "High" traffic flow, simply because there are very few gaps when the traffic is high.

Criterion	**Explanation**
Vehicles passed	Total number of vehicles passed the traffic light during the time period. If the pedestrian crossing creates queue or reduced speed, fewer vehicles will pass the crossing, and the result will be a less efficient traffic system.
Average pedestrian waiting time	This is the average waiting time among all the pedestrians that used the traffic light. This measure how efficient the crossing is for pedestrians.
Numbers of transitions executed	This value indicates the total number of times the traffic light gave pedestrians transition time. Vehicles efficiency may be affected by too many transitions and pedestrians are affected by too few.
Two or less vehicles delayed by traffic light	This value is number of times one or two vehicles are stopped by the traffic light. It gives an indication of situations where a gap was close to be exploited, but instead interrupted the traffic.
No vehicles delayed by traffic light	This value is number of times a traffic light changed and no vehicles had to stop. This is a perfect transition where the traffic isn't interrupted.
Less time to fast pedestrians	When all the pedestrians waiting are fast, less time is given. This will increase the efficiency because the vehicles have to wait a shorter time period.
Less time to normal pedestrians	In some situations normal pedestrians are given less time to increase the efficiency.

Table 6.2: Criteria for evaluating efficiency in a pedestrian crossing

Criterion	Explanation
Extra time to slow pedestrians	When a slow pedestrian is detected among the pedestrians, extra time is given. The safety of the slow pedestrian will increase, because of a more suitable crossing time.
Extra time to slow groups	A slow group will typically need a very long transition time, so that the group does not have to cross the road in several batches. The extra time should be enough to send the whole group in one cycle, which will increase safety.
Average waiting time less than 16 seconds	This is the number of times the average waiting time of all the pedestrian that shall cross is less than 16 seconds. Short waiting time will increase safety, because it reduces the possibility that pedestrians will jaywalk.
Pedestrians waited more than 50 seconds	A long waiting time will result in reduced safety, because impatient pedestrians are most likely to attempt to jaywalk. It also reduces efficiency for pedestrians.

Table 6.3: Criteria for evaluating safety in a pedestrian crossing

Test scenario 1

The first scenario is a basic test scenario with high traffic flow. The traffic flow and number of pedestrians are based on the counting performed at Brattøra (see section 4.4). The type of pedestrians does not directly correspond to the pedestrians that were counted at Brattøra. The reason for this is that only a very small fraction of the pedestrians were "abnormal". This is probably because the crossing is not located in an area were slower pedestrians travel through. Also, since the counting was performed in March, the weather in Trondheim affected the number of "Fast" pedestrians (cyclists etc.). To be able to test the strength of the system, we added both "Slow" and "Fast" pedestrians to the scenario. The properties of test scenario 1 are shown in table 6.4.

Parameter	Value
Time period	30 min
Traffic flow	High
Vehicles per hour	1700
Pedestrians	69
Slow pedestrians	6 (8.7%)
Fast pedestrians	2 (2.9%)
Slow group	0 (0%)

Table 6.4: Description of test scenario 1

Criterion	CBR controlled traffic light	Normal traffic light
Efficiency:		
Vehicles passed	828	829
Average pedestrian waiting time	25.0	27.4
Numbers of transitions executed	21	19
Less time to fast pedestrians	2/2	
Less time to normal pedestrians	11/14	
Safety:		
Extra time to slow pedestrians	5/5	
Average waiting time less than 16 seconds	4/21	1/19
Pedestrians waited more than 50 sec.	9/21	

Table 6.5: Results from test scenario 1

The results of test scenario 1 are shown in table 6.5. Some of the criteria in the tables contain only results for the CBR controlled traffic light (e.g. "Less time to fast pedestrians"). This is because these are situations that never happens with the normal traffic light. In this scenario, the performance is almost the same between the CBR controlled traffic light and the normal traffic light. The average number of pedestrians waiting is lower for the CBR controlled traffic light, which is a result of the system being able to let pedestrians pass at all times.

Test scenario 2

Since both systems performed well on test scenario 1, some of the properties (shown in table 6.6) have been changed to make the task harder. The number of pedestrians has been increased with 20%. The number of slow and fast pedestrians has been increased to correspond to the change in the number of pedestrians.

Parameter	**Value**
Time period	30 min
Traffic flow	High
Vehicles per hour	1700
Pedestrians	83 (+20%)
Slow pedestrians	8 (9.6%)
Fast pedestrians	8 (9.6%)
Slow group	0 (0%)

Table 6.6: Description of test scenario 2

Criterion	**CBR controlled traffic light**	**Normal traffic light**
Efficiency:		
Vehicles passed	829	829
Average pedestrian waiting time	25.4	26.4
Numbers of transitions executed	25	27
Less time to fast pedestrians	3/3	
Less time to normal pedestrians	12/16	
Safety:		
Extra time to slow pedestrians	5/5	
Average waiting time less than 16 seconds	5/25	3/27
Pedestrians waited more than 50 sec.	8/25	

Table 6.7: Results from test scenario 2

Table 6.7 shows the results from the second test scenario. Increasing the number of pedestrians did not make the task of controlling the crossing too hard for any of the traffic

lights. The CBR controlled traffic light is performing better than the normal traffic light, in terms of letting pedestrians wait for a shorter time period in average, but the differences in the results are small.

Test scenario 3

Table 6.8 shows the properties of the third test scenario. Now the number of vehicles per hour has been set to 1954 (+15% compared to the counting), since the results from the second test scenario showed no sign of traffic congestions.

Parameter	**Value**
Time period	30 min
Traffic flow	High
Vehicles per hour	1954 (+15%)
Pedestrians	83 (+20%)
Slow pedestrians	8 (9.6%)
Fast pedestrians	8 (9.6%)
Slow group	0 (0%)

Table 6.8: Description of test scenario 3

Criterion	**CBR controlled traffic light**	**Normal traffic light**
Efficiency:		
Vehicles passed	979	956
Average pedestrian waiting time	30.9	26.4
Numbers of transitions executed	23	27
Less time to fast pedestrians	3/3	
Less time to normal pedestrians	11/12	
Safety:		
Extra time to slow pedestrians	5/5	
Average waiting time less than 16 seconds	2/23	3/27
Pedestrians waited more than 50 sec.	17/23	

Table 6.9: Results from test scenario 3

The results of the third test scenario are shown in table 6.9. As can be seen, the CBR controlled traffic light is adjusting to the increase in number of vehicles, by letting the pedestrians wait longer before they can go across. In 17 of 23 transitions, the pedestrians will have to wait the maximum number of time (50 seconds). This leads to an increase in the number of vehicles that can pass the crossing (979 compared to 956). Still, none of the systems are experiencing any traffic congestions.

Test scenario 4

For test scenario 4, we increased the number of vehicles by 25% relative to the numbers from the counting. This is because the there were no traffic congestions in test scenario 3. The properties of test scenario 4 are shown in table 6.10.

Parameter	**Value**
Time period	30 min
Traffic flow	High
Vehicles per hour	2124 (+25%)
Pedestrians	83 (+20%)
Slow pedestrians	8 (9.6%)
Fast pedestrians	8 (9.6%)
Slow group	0 (0%)

Table 6.10: Description of test scenario 4

Criterion	**CBR controlled traffic light**	**Normal traffic light**
Efficiency:		
Vehicles passed	1055	986
Average pedestrian waiting time	30.6	26.4
Numbers of transitions executed	20	27
Less time to fast pedestrians	3/3	
Less time to normal pedestrians	9/11	
Safety:		
Extra time to slow pedestrians	6/6	
Average waiting time less than 16 seconds	4/20	3/27
Pedestrians waited more than 50 sec.	16/20	

Table 6.11: Results from test scenario 4

Table 6.11 shows the results of the fourth test scenario. The normal traffic light did not create queues, but the traffic moved slower, which resulted in that fewer vehicles passed the traffic light. Although the CBR controlled traffic light let more vehicles pass, the average waiting time for pedestrians did not increase, compared to test scenario 3.

Test scenario 5

The fifth test scenario consists of 3 tests. This is because the normal traffic light with a 30 second waiting time pedestrians had problems with handling the large amount of vehicles per hour. We therefore performed a test with a normal traffic light with 50 second waiting time, which is consistent with the maximum transition time given by the CBR system. The properties of the scenario are shown in table 6.12.

Parameter	Value
Time period	30 min
Traffic flow	High
Vehicles per hour	2294 (+35%)
Pedestrians	83 (+20%)
Slow pedestrians	8 (9.6%)
Fast pedestrians	8 (9.6%)
Slow group	0 (0%)

Table 6.12: Description of test scenario 5

Criterion	**CBR controlled traffic light**	**Normal traffic light 30sec**	**Normal traffic light 50sec**
Efficiency:			
Vehicles passed	1105	974	1080
Average pedestrian waiting time	36.94	26.4	36.1
Numbers of transitions executed	20	27	19
Less time to fast pedestrians	3/3		
Less time to normal pedestrians	10/11		
Safety			
Extra time to slow pedestrians	6/6		
Average waiting time less than 16 seconds	1/20	3/27	1/19
Pedestrians waited more than 50 sec.	19/20		

Table 6.13: Results from test scenario 5

Table 6.13 shows the result from test scenario 5, for the three runs that were carried out. As mentioned, the normal traffic light with a 30 second waiting time was not able to prevent queues. It is also important to note that the CBR controlled traffic light performed better than the normal traffic light with a 50 second waiting time, since it let 25 more vehicles pass, even when the normal traffic light let all pedestrians cross the road after the maximum time (50 seconds) and the CBR controlled traffic light gave extra transition time to slow pedestrians.

Test scenario 6

Vehicles per hour are reduced to match the numbers from the counting at Brattøra from 12.30 to 13.30, when the traffic flow was normal. As in the counting from 15.30 to 16.30, the pedestrians that crossed the road at Brattøra were mostly normal. Consequently, we decided to add pedestrians in both the "Fast" and the "Slow" category. The properties of the scenario are shown in table 6.14. This scenario is not run to test how the system can handle larger amounts of traffic, but to test its ability to take advantage of gaps in traffic to and give an adjusted transition time to pedestrians.

Parameter	**Value**
Time period	30 min
Traffic flow	Normal
Vehicles per hour	1000
Pedestrians	38
Slow pedestrians	4 (9.5%)
Fast pedestrians	4 (9.5%)
Slow group	1 (2.6%)

Table 6.14: Description of test scenario 6

Criterion	**CBR controlled traffic light**	**Normal traffic light**
Efficiency:		
Vehicles passed	480	480
Average pedestrian waiting time	9.1	28.2
Numbers of transitions executed	28	22
Two or less vehicles delayed by traffic light	14/28	3/22
No vehicles delayed by traffic light	4/28	0/22
Less time to fast pedestrians	4/4	
Less time to normal pedestrians	18/19	
Safety:		
Extra time to slow pedestrians	4/4	
Extra time to slow groups	1/1	
Average waiting time less than 16 seconds	23/28	3/22
Pedestrians waited more than 50 sec.	0/28	

Table 6.15: Results from test scenario 6

The results from test scenario 6 are shown in table 6.15. To evaluate the system's ability to exploit gaps in traffic, we have added two new evaluation criteria; "No vehicles delayed by traffic light", and "Two or less vehicles delayed by traffic light". The first criterion is met if the transition for pedestrians is executed, with no vehicles having to wait for a green light. This means that the system has exploited a gap in traffic, letting pedestrians pass when there are no vehicles. Since the total transition time can be from approximately 6 to 14 seconds, situations can occur were one or two vehicles can arrive at the crossing at the end of the transition time. Because of this, the criterion "Two or less vehicles delayed by traffic" was added.

The numbers show that the times where no vehicles were stopped were 4 by the CBR controlled traffic light and 0 by the normal traffic light. Stopping two or less vehicles occurred in 14 out of the 28 transitions with the CBR controlled traffic light. In section 6.2, we will discuss why this happened, and how we could improve the system so that the number of times when no vehicles were stopped is increased.

It is also notable that no pedestrians had to wait the maximum time of 50 seconds,

and that all pedestrians got the correct transition time. Also, the average waiting time for pedestrians was much lower with the CBR controlled traffic light, compared to the normal traffic light (9.1 seconds over 28.2 seconds).

Test scenario 7

The last test scenario was conducted with "Low" traffic flow. Table 6.16 shows the properties of the seventh test scenario.

Parameter	**Value**
Time period	30 min
Traffic flow	Normal
Vehicles per hour	360
Pedestrians	38
Slow pedestrians	4 (9.5%)
Fast pedestrians	4 (9.5%)
Slow group	1 (2.6%)

Table 6.16: Description of test scenario 7

Criterion	**CBR controlled traffic light**	**Normal traffic light**
Efficiency:		
Vehicles passed	173	172
Average pedestrian waiting time	5	28.2
Numbers of transitions executed	28	22
Two or less vehicles delayed by traffic light	6/28	16/22
No vehicles delayed by traffic light	19/28	5/22
Less time to fast pedestrians	4/4	
Less time to normal pedestrians	20/20	
Safety:		
Extra time to slow pedestrians	4/4	
Extra time to slow groups	1/1	
Average waiting time less than 16 seconds	27/28	3/22
Pedestrians waited more than 50 sec.	0/28	

Table 6.17: Results from test scenario 7

Table 6.17 shows the results from the seventh test scenario. The results for the normal traffic light did not change from the previous test scenario, except that the vehicles passed has increased. This is not unexpected, since the traffic light is controlled in a static manner, which makes it unable to adapt to the situation.

The CBR controlled traffic light got changes in the results related to pedestrians. The results showed that the average waiting time for pedestrians was reduced to 5 seconds. This shows that the CBR system finds a more appropriate waiting time, based on the traffic situation that is detected.

The CBR controlled traffic light exploited gaps in traffic in 19 of the 28 transitions, while the normal traffic light only exploited gaps 5 times.

6.2 Discussion

In this section, we will discuss the results presented in the previous section and attempt to point out the strengths and weaknesses of the system. The test scenarios were used to evaluate the strength of the system in controlling a signal controlled pedestrian crossing, while the results from the cross-validation evaluated the strength of the system in terms of how good it is at retrieving cases and reusing solutions.

The first five scenarios that were run in the simulator were focused on testing whether the system could adjust to high amounts of traffic. This is important for testing the efficiency of the system, because the number of vehicles the system can send through the crossing directly reflects the efficiency of the crossing. Test scenarios 2 to 5 are the most interesting, because test scenario 1 had fewer pedestrians.

The most obvious difference between the CBR controlled crossing and the normal crossing, is that the CBR controlled crossing adjusts the intervals between transitions relative to the amount of traffic. Figure 6.2 shows the average waiting time for pedestrians for the different amounts of traffic. It is important to point out that the scenarios with 360 and 1000 vehicles were carried out with fewer pedestrians. As can be seen, the number increases along with the traffic flow (the normal traffic lights have fixed time intervals, thus the only change in average waiting time comes from the different amount of pedestrians). This is a negative aspect in terms of safety, since long waiting time can make pedestrians impatient, which may lead to pedestrians jaywalking. It also makes the crossing less efficient for pedestrians, since they use more time to get from one place to another. However, according to the experts at the NPRA, it is generally more accepted by pedestrians that they need to wait longer when traffic is high. Thus, it can be seen as positive that the waiting time increases, since it may alleviate some of the pressure from the crossing. Also, the average waiting time for the normal traffic light with a 50 second waiting time is almost the same as the highest value for the CBR controlled traffic light. The system also performs much better when the traffic flow is “Normal” or “Low”, with 9.1 seconds and 5.0 seconds waiting time, respectively.

A second interesting result is the number of vehicles that passed the crossing. In test scenario 2 (see table 6.7 on page 89), both systems allowed 829 vehicles to pass. This is not the exact half of the vehicles per hour, because the vehicles are sent into the system about 1 km from the crossing, on each side of the road. It is still the maximal number

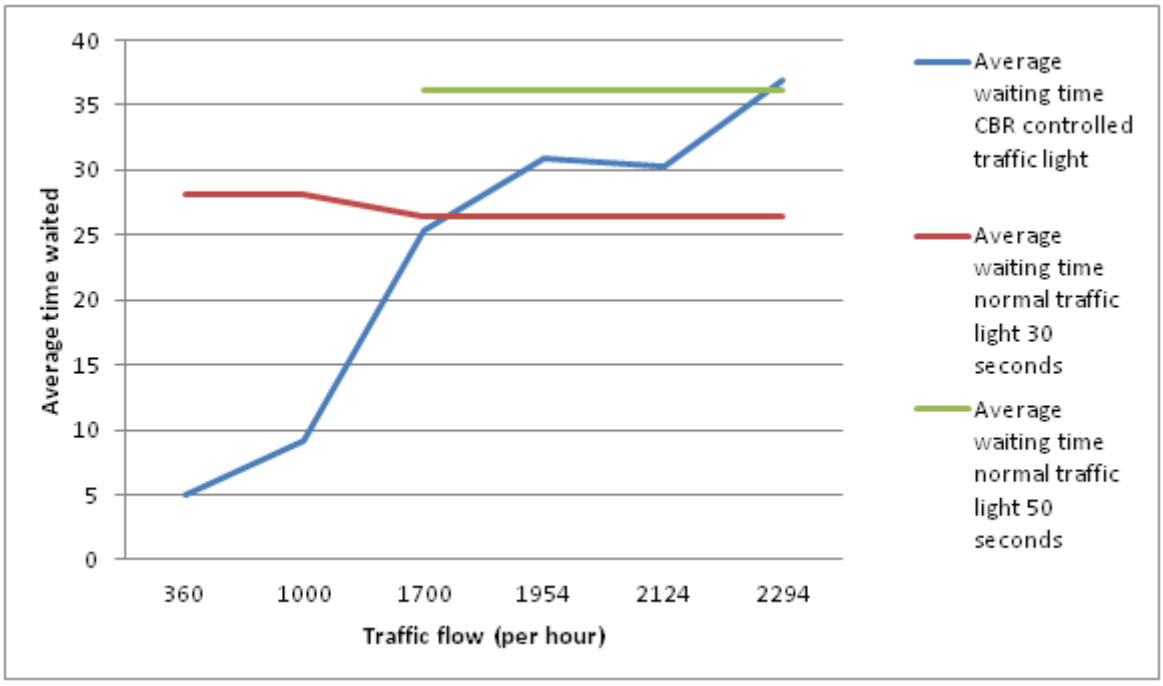

Figure 6.2: Average waiting time for pedestrians with different amounts of traffic

of vehicles that could be sent through the crossing. In test scenario 3 and 4, the normal traffic light created some congestion. When the number of vehicles was increased for the fifth test scenario, the traffic stopped with the normal traffic light, only letting 974 vehicles pass, compared to 1105 by the CBR controlled traffic light (see table 6.13 on page 93). Since the normal traffic light changed the light after 30 seconds, and the maximum time for the CBR controlled traffic light was 50 seconds, we changed the configuration of the normal traffic light to 50 seconds. Table 6.13 on page 93 showed that the CBR controlled traffic light still handled the large amount of vehicles better (although not significantly). This happened even though the CBR system gave an increased transition time in 6 out of 6 times when slow pedestrians were detected (3 times the system gave less transition time to fast pedestrians). It is reasonable to believe that this slight difference comes from the CBR systems ability to time the transitions better, and that it gives a shorter transition time to some of the normal pedestrians.

It is important to point out that the test environment is far from perfect. There exist much more complex systems that are used to control crossings today, than the traffic lights that used as comparison to the CBR controlled traffic light. Also, formations of traffic congestions are less likely to occur in the simulator than in the real world. The

reason for this is that the crossing is not a part of a larger traffic network, thus the congestion will have to be formed in this crossing alone. In a larger traffic network, the crossings are dependent of each other, so that small congestions in one crossing can create larger congestions over time in the complete network. Thus, it would be interesting to test the system in a larger traffic network, by for example using Aimsun (described in section 4.6.3). Another interesting test would be to use multiple connected versions of the CBR controlled traffic light, to see if the systems would in some way work together (for a further discussion, see section 7.1). Still, the results seem promising, since the system dynamically adjusts to changes in the traffic situation.

In the last two test scenarios (see table 6.15 on page 95 and table 6.17 on page 97), the traffic flow was lowered, to test other parts of the case base. Here, the criteria; "Two or less vehicles delayed by traffic light" and "No vehicles delayed by the traffic light", were added. These are important because when there are fewer vehicles on the road, it is essential to find a balance between giving a green light to vehicles and to pedestrians. If the times when no vehicles are approaching the crossing can be used to let pedestrians cross the road, the efficiency of the signaling system is increased. The reason we added the criterion, "Two or less vehicles delayed by traffic light", is because if this occurs, the system is often closer to executing an optimal transition. In many of these situations, the vehicles stop at the end of the transition time, this makes it close to not delaying any vehicles.

Test scenario 6 had a "Normal" traffic flow with 1000 vehicles per hour. Compared to the earlier scenarios, the average waiting time has greatly decreased, from about 25 to 34 seconds, to 9.1 seconds, as mentioned earlier. This is caused by the system's ability to exploit gaps in traffic and because the reduced traffic flow makes the system less strict on letting pedestrians pass. In 14 of the 28 transitions that were executed, there were two or less vehicles that had to wait for a green light. In 4 out of 28 times no vehicles were delayed at all. It seems reasonable that the first number is so much higher, because there will be gaps in traffic when the traffic flow is "Normal", but they are not large enough to be perfectly utilized. Still, the system uses gaps to a much greater extent than the normal traffic light, which only hit gaps randomly.

For the last test scenario, the traffic flow was set to "Low". As expected, the average waiting time for pedestrians were further lowered, to 5 seconds. In 19 out of 28 transitions

(67.9%), no vehicles were delayed by the traffic light, compared to only 5 out of 22 times (22.7%) for the normal traffic light. This is to some extent affected by the transition time given by the CBR controlled traffic light. When the number of pedestrians are "Few", i.e. 1 to 3 pedestrians, then "Normal" pedestrians are given less time to cross the road. For this reason, the CBR controlled traffic light can more easily exploit gaps in traffic, since the gaps can be smaller, while the static transition time of the normal traffic light can be too long. In section 2.2, we described a type of signal controlled pedestrian crossing called PUFFIN. This type of pedestrian crossing can work somewhat similar as the CBR controlled crossing, because the PUFFIN crossing uses camera sensors to change the traffic light when the pedestrian has walked across. The difference is that the CBR controlled traffic light uses the number of pedestrians that has an intention to cross the road. This can be used to predict the increase or decrease in transition time, instead of looking back and changing the transition time. Furthermore, a PUFFIN crossing is best used in areas where there are few pedestrians crossing the road and in areas where it is less likely that traffic congestions will occur, because it will always prioritize pedestrians. With our system, the crossing will only prioritize pedestrians in periods where the traffic is low, thus it is reasonable to believe that it can function in busier areas.

There are some problems with the CBR system, as it is today, that we now wish to present. This is both caused by the lack available traffic data and caused by the fact that some of the important modules have not been implemented yet. First, there is a problem with the detectors that were used in the test scenarios. These are very similar to the detectors that are available today. The main problem is that they only detect when a vehicle is just above the detector. To fully detect gaps in traffic, it would be useful to follow the closest vehicle for a longer distance (e.g. 300 meters). The detectors that are furthest away from the crossing is placed 70 meters from the crossing. If the speed of a vehicle is 60 km/h, the vehicle will use approximately 4.2 seconds from the detector to the crossing (given a constant speed). The transition times given by the CBR system vary from about 6 seconds to 14 seconds. Thus, it would be more practical to know if there are vehicles 6 to 14 seconds away. In section 7.1, we will discuss this in more detail.

Another important thing to be aware of is that the module for interpreting the type of pedestrians has not yet been implemented. There is no guarantee that the interpretation can be done successfully. The way the system is implemented today, it relies on the fact

that it will always get a good interpretation of what type the pedestrians are. Still, it is reasonable to believe that looking at different features of the human body can give an indication of what type of pedestrian is approaching the crossing. In section 7.1, we present some ideas on how this interpretation could be accomplished.

In addition to the information retrieved from the camera sensors, some pedestrians can be equipped with devices, to let the system know that an "abnormal" pedestrian is approaching. An example is blind pedestrians, because it can be hard to detect blind pedestrians by looking at the images from the camera sensor (discussed in section 5.3.4). Having to equip some pedestrians with devices are of course a second choice, since devices can be less reliable (problems with battery, Bluetooth/WIFI connection etc.).

Overall, the experimental results of the test scenarios give promising indications. The CBR system does not make any obvious mistakes in the retrieval phase. For that reason, the system always gives extra time to slower pedestrians, and a shortened transition time for faster pedestrians. If this can be introduced in a real crossing, it could improve the safety in the crossing, since slower pedestrians can cross the road without having to worry about not getting there in time. The system also performs well at adjusting to the different situations that occur in traffic, by changing the waiting time for pedestrians and exploiting gaps in traffic. By prioritizing vehicles in times of the day when there are high amounts of traffic, and prioritizing pedestrians when the amounts are lower, it is reasonable to believe that the systems overall efficiency can be improved. The high average similarity from the cross-validation tests (see section 6.1.1) also indicate that the case base is properly filled. Still, there is much work left on many components of the system. Some of the possibilities will be discussed in the next chapter.

Chapter 7

Conclusion and further work

This chapter proposes some possible future work that can be done on the system, and give some concluding remarks on the work done in this project.

7.1 Further work

There are many aspects that should be considered when further developing the system presented in this report. Many components of the system can be enhanced, and many components can be added to improve the system. In this section, we will present some ideas that can be useful for those who may further develop the system.

Development of the system can be split into several parts; case base maintenance, improving the steps of the CBR cycle, adding or removing features from the case structure, and changing how the query case is built. For the first part, case base maintenance, with the amount of features that is used in the cases today, the cases that are in the case base seems satisfactory. The reason for this is that the system has not made any mistakes in the test scenarios. Still, it could be useful to have experts go through the cases once more; to verify that all the cases in the case base give correct solutions. Some cases are actually quite hard to give a solution, because they typically lie between two solutions. Also, if new important features were to be added, the case base would have to be updated.

The steps of the CBR cycle can be improved in many ways. For one, the table similarity functions that were used in the evaluation contain values that were manually configured, based on indications given by the experts at the NPRA. The retrieval process could therefore be improved by fine tuning these values, by either creating a learning

algorithm for finding the optimal values, or by putting more effort in setting the values manually.

Another problem with the system is that some of the features in the cases are very dependent of each other. For example, the type of pedestrian feature is very important in cases where the solution is to give a transition time, while it can be of very little importance in cases where the solution is to not execute a transition. This increases the number of cases that is needed, because the case base has to be filled with very similar cases, where only the type of pedestrian feature and the solution is changed. Since the number of features is quite small, it doesn't cause many problems for the system. But if some features were to be added, it could involve having to add a very large amount of cases, which can make the retrieval process slower than what is desirable.

A possible solution to this problem would be to change the way cases are retrieved. Instead of retrieving cases from the whole case base, the case base could be divided into categories, where the retrieval would only be performed on one or some of the categories. The categories of the query could be decided by a simple set of rules. For example, if the type of pedestrian is "Slow" or "SlowGroup", the retrieval algorithm could search parts of the case base with only these types of situations. Such an approach would make the search smaller, since the system would not have to search through the whole case base. This is similar to work done by Smyth et al.[41]. There the case base was split into different competence groups, where cases in a competence group share competence (i.e. they solve the same types of problems) with one or more of the cases in the group. By representing each group with one reference case, the retrieval process is carried out by searching in the set of reference cases, and then search in the competence group of the retrieved reference case. Using reference cases for each category could be an alternative to having a set of rules for deciding the categories of a case.

There is also work that needs to be carried out to interpret what type a pedestrians is, as mentioned in earlier chapters. By looking at different features on the human body such as; height (to differentiate children from adults), how the body is shaped (e.g. an old man might walk more crooked), or by looking at whether the pedestrian is using a walking aid (walking stick, crutches, wheel chair etc.), it could be possible to interpret the type of pedestrian. Another feature could be the speed of the pedestrian. Alone, this could not be used to interpret the type of pedestrian, because all pedestrians can

walk slowly, even if they are fully capable of walking faster. It is not the purpose of the system to give normal pedestrians that walk slowly, an extended transition time. The speed could rather be used to verify the solution of the interpretation. For example, if a young man walks slowly towards the crossing, if the interpretation of the images from the camera sensors tells the system that the young man is a normal pedestrian, and then the system will re-evaluate the camera data to see if it might have made a mistake. On the other hand, if the pedestrian is interpreted to be slow, and the speed is slow, then it is probable that the pedestrian does in fact need an extended transition time.

Before the system can be used in the real world, the accuracy of the interpretation of pedestrian intention has to be improved. As mentioned before, Intention-based Sliding Door created by Solem had an accuracy of 86%. To use the system in the real world, the accuracy has to be close to perfect. The first obvious limitation of this system is that it uses a Kinect sensor. This sensor is not developed for monitoring a possibly large crowd of people. Thus, there is a need for using a more appropriate sensor. Another problem with interpreting the intention of a pedestrian is that abnormal pedestrians, such as people carrying an umbrella or other large objects, may occlude the sensor images, making it very hard to do a correct interpretation. Thus, it might be necessary to use multiple interpretation mechanisms. In chapter 3, we described work done by Hogg et al.[29], where they trained an artificial neural network, on long image sequences, to predict the trajectories of pedestrians. It is an interesting approach, since knowing the path of which the pedestrian is likely to follow, could be used to tell if the pedestrian indents to cross the road. This is an example of a method that could be used together with the system created by Solem. A possible approach would then be to create a CBR system or some other similar mechanism, to switch between the different methods for interpreting the intention of pedestrians.

In chapter 3, we described a system created by Li and Zhao, where CBR was used for urban intersection control. If the system could not retrieve any cases for a given situation, actuated control was used to control the intersection. The system used a detecting and surveillance system to see, over a period of time; if the solution was successful (i.e. traffic congestion was less serious). If it was, the solution would be permanently stored as a case in the case base. This is a very interesting approach, because making the system learn, could improve its performance over time. For example, if the system uses the solution of

a case several times to let pedestrians pass, but every time the pedestrians are given a too long transition time, then the system could update the case to give a shorter transition time. Similarly, if no case were to be retrieved, then the system could have an underlying default system that could take control of the crossing, and give a default transition time. The system could then see if this solution was correct, and if it was, store the query case with the default solution in the case base. However, integrating learning in a system that is not controlled by humans, and is used in real-time, can be risky. For example, if the detectors used are not working correctly (e.g. caused by pollution), the system can start learning incorrect cases, which may impair the systems performance. Thus, it is important to only learn cases that the system can guarantee are correct.

The detecting and surveillance system mentioned above, could also be used to cancel the green time, if the system has made a mistake (e.g. a pedestrian did not have the intention to cross after all) or if the green time was to long (i.e. the pedestrians that had intention to cross has already crossed the road). This is used in PUFFIN crossings (see section 2.2).

As mentioned in section 6.2, the detectors for detecting vehicles that are used in the test scenarios, and that are available in the real world today, have some limitations. In the future, it is probable that vehicles will be equipped with a GPS sensor, so that vehicles in fact can be monitored for greater distances. This would enable the system to be more prepared and to make more accurate decisions than it could today. For example, the system could use the distance from the closest vehicle to assess whether it could let some pedestrians cross the road, given the transition time they need (e.g. if the vehicle is 7 seconds away, but a slow pedestrian needs 9 seconds to cross the road, the system would not execute the transition).

To reveal the real strengths of the system, it is also necessary to test how well the system would work in a larger traffic network, with multiple CBR controlled traffic lights. It would be interesting to see if the traffic lights would dynamically adjust to each other, or if a centralized system would have been used to control the crossings. A possible way to do it would be to create a multi-agent system where each CBR controlled traffic light would be agents, interacting with each other through a centralized unit. This is similar to the approach used in the system created by Schutter et al., which was described in chapter 3. There the traffic network was split into smaller sub networks, where each sub network

had its own case base. By representing signal controlled crossings and intersections as agents, each having its own case base, it might be possible to control large traffic network. The case bases could be similar in similar areas (e.g. two intersections with about the same amounts of vehicles and pedestrians, could have the same case bases initially), but learning mechanisms could make the case bases different over time.

A last suggestion on how the system could be improved is to add more knowledge to the system. An example is to add a feature for differentiating between different types of vehicles that approach the crossing. For example, knowing if an emergency vehicle is approaching or if a bus is close by, can give these vehicles a higher priority. Also, adding knowledge about the area around the crossing (or having the system learn it) could improve the reasoning process. For example, if a school is located near the crossing, it can indicate that large amounts of pedestrians will use the crossing in the hours when the school children go to school, and when they go home. This could be used for predicting the type of pedestrian feature, since most of the pedestrians that cross the road at these times will be school children. It could also enable the system to predict at what times the amount of pedestrians will be high, so that it can be prepared for these situations.

7.2 Conclusion

In this thesis, we have presented a prototype system that uses Case-based reasoning for controlling a pedestrian crossing. It uses solutions to earlier experienced situations, to solve new problems. Situations are described by different features, related to both vehicles and pedestrians. Pedestrians are monitored by a Kinect sensor, and a system created by a previous MSc student in our group (Solem), was integrated to interpret the intention of pedestrians. In this way, the system can determine if someone wants to cross the road, without having the pedestrians make the signal themselves. Vehicle information was obtained through the SCANeR Studio, which is a software tool for simulating traffic. The information from pedestrians and vehicles was sent to the system using a framework called CVIS.

The system has been programmed in Java, and uses myCBR as a tool for making the process of creating a CBR system easier. This project has been conducted in cooperation with the Norwegian Public Roads Administration.

The goals of this project was to study whether CBR could be proved useful in controlling a single pedestrian crossing, and if such a system would be better than today's systems in terms of; efficiency, safety and user-friendliness. The prototype system that has been implemented shows promising results. Using past experience to solve new problems seems to work in this domain, since system is able to dynamically adjust to changes in the traffic situation. It improves efficiency in that gaps in traffic can be exploited and that it is able to balance between prioritizing vehicles and pedestrians. Safety is improved in that some types of pedestrians are given an extended transition time. It is also reasonable to believe that automatically detecting whether pedestrians intend to cross the road can improve the user-friendliness, since the pedestrians does not have to perform the signaling themselves.

The prototype system created in this project is just a foundation for what we hope will be a larger and more comprehensive system in the future. The systems strength is that it adapts to the traffic situation, so that more optimal transitions can be executed. Also, the interpretation of pedestrian's intentions enables the system to take faster decisions, because the system does not need to wait for the pedestrians to signal. The system also has some weaknesses that need to be improved. The most apparent weakness is the interpretation of intention, since its accuracy needs to be is too low. This mostly comes from the fact that the Kinect sensor was not developed for monitoring pedestrians, making it less useful for this matter. Using a more appropriate sensor, with a more accurate reasoning mechanism, would improve the accuracy, and therefore the entire system. Also, the cases could incorporate more knowledge, like type of vehicle and distance to closest vehicle, to better describe the traffic situation. In addition, there is a need for interpreting the type of pedestrians, since this module has not been implemented. Still, we believe that a full implementation of the system could improve the way pedestrian crossings are controlled.

Bibliography

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

[2] Keith L. Downing. Introduction to evolutionary algorithms, 2010. Lecture note from the course IT3708 - Subsymbolic Methods in AI at IDI.

[3] John Sverre Solem. Intention-based sliding doors. Master's thesis, Norwegian University of Science and Technology, 2011.

[4] Billie Louise Bentzen and Lee S. Tabor. Accessible pedestrian signals. Technical report, Report presented to the U.S. Access Board, 1998.

[5] Norwegian Public Roads Administration. Vegtrafikkulykker. Technical report, Norwegian Public Roads Administration, 2009.

[6] World Health Organization. Global status report on road safety. Technical report, WHO - Department of Violence & Injury Prevention & Disability (VIP), 2008.

[7] Norwegian Ministry of Transportation and Communication. National transport plan 2010-2019. Technical report.

[8] Statens vegvesen. Håndbok 142, 2007.

[9] Kyrre Gran. Utprøving av puffin-konseptet, 2005.

[10] SWARCO. Spot / utopia. `URL:http://www.swarco.no/default.asp?menu=35`, 2010. [Online; accessed 22-March-2012].

[11] ERTICO. About cvis. `URL:http://cvisproject.org/en/about_cvis/`, 2011. [Online; accessed 19-November-2011].

[12] OSGi Alliance. About osgi. URL:http://www.osgi.org/About/HomePage, 2011. [Online; accessed 20-October-2011].

[13] Katrin Bilstrup, Annette Böhm, Kristoffer Lidström, Magnus Jonsson, Tony Larsson, Lars Strandén, and Hossein Zakizadeh. Vehicle alert system. 2007.

[14] ERTICO. The new cooperative era. URL:http://www.cvisproject.org/download/ERT_CVIS_FinalProject_Bro_06_WEB.pdf, 2009. [Online; accessed 21-February-2012].

[15] Chen-Fu Liao. Mobile accessible pedestrian signals (maps) for people who are blind. In *18th ITS World Congress, Orlando, Florida*, 2011.

[16] Roger Schank. *Reconstructive Memory: A Computer Model.* Cambridge University Press, 1983.

[17] Janet L. Kolodner. Dynamic memory; a theory of reminding and learning in computers and people. 1982.

[18] J.A. Wentworth. Expert systems in transportation. In *AAAI Technical Report WS-93-04*, 1993.

[19] Kaidong Li and Nigel Waters. Transportation networks, case-based reasoning and traffic collision analysis: A methodology for the 21st century. In Aura Reggiani and Laurie A. Schintler, editors, *Methods and Models in Transport and Telecommunications*, pages 63–92. Springer Berlin Heidelberg, 2005.

[20] Adel W. Sadek, Michael J. Demetsky, and Brian L. Smith. Case-based reasoning for real-time traffic flow management. *Computer-Aided Civil and Infrastructure Engineering*, 14(5):347–356, 1999.

[21] Neil Prosser and Stephen Ritchie. A real-time expert system approach to freeway incident management. In *Transportation Research Record, Issue 1320*, pages 7–16, 1991.

[22] A. Gaupta, V.J Maslanka, and Spring G.S. Development of prototype knowledge-based expert system for managing congestion on massachusetts turnpike. In *Transportation Research Record No. 1358*, pages 60–66, 1992.

[23] A.C Boury-Brisset and N. Tourigny. Knowledge capitalisation through case bases and knowledge engineering. In *Knowledge-Based systems 13*, pages 297–305, 2000.

[24] Fred Lin, Tarek Sayed, and Paul Deleur. Estimating safety benefits of road improvements: Case based approach. In *Journal of Transportation Engineering*, pages 385–391, 2003.

[25] Asad Khattak and Adib Kanafan. Case-based reasoning: A planning tool for intelligent transportation systems. In *Transpn Res.-C*, pages 267–288, 1996.

[26] L. Wang, C.C. Hayes, and R.R. Penner. Automated phase design and timing adjustment for signal phase design. In *Applied Intelligence 15*, pages 41–55, 2001.

[27] B. De Schutter, S.P. Hoogendoorn, H. Schuurman, and S. Stramigioli. A multi-agent case-based traffic control scenario evaluation system. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, pages 678 – 683 vol.1, oct. 2003.

[28] Zhenlong Li and Xiaohua Zhao. A case-based reasoning approach to urban intersection control. In *Proceedings of the 7th World Congress on Intelligent Control and Automation*, pages 7113–7118, 2008.

[29] N. Johnson and D.C. Hogg. Learning the distribution of object trajectories for event recognition. In *Image and Vision Computing*, pages 609–615, August 1996.

[30] D. Gavrila. Pedestrian detection from a moving vehicle. In *Computer Vision — ECCV 2000*, volume 1843 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin / Heidelberg, 2000.

[31] L. Zhao and C.E. Thorpe. Stereo- and neural network-based pedestrian detection. *Intelligent Transportation Systems, IEEE Transactions on*, 1(3):148 –154, sep 2000.

[32] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *International Journal of Computer Vision*, pages 153–161, 2005.

[33] DFKI GmbH. mycbr. `URL:http://mycbr-project.net/index.html`. [Online; accessed 18-March-2012].

[34] Complutense University of Madrid. jcolibri. URL:http://gaia.fdi.ucm.es/research/colibri/jcolibri. [Online; accessed 24-May-2012].

[35] The National Center for Biomedical Ontology. Protègè. URL:http://protege.stanford.edu/. [Online; accessed 18-March-2012].

[36] OKTAL. Scaner. URL:http://www.scanersimulation.com/. [Online; accessed 18-March-2012].

[37] Eclipse Foundation. Eclipse. URL:http://www.eclipse.org/. [Online; accessed 18-March-2012].

[38] Microsoft. Microsoft visual c++ 2010 express. URL:http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express. [Online; accessed 18-March-2012].

[39] Andrew W. Moore. Cross-validation for detecting and preventing overfitting. URL:http://www.autonlab.org/tutorials/overfit10.pdf. [Online; accessed 05-June-2012].

[40] TSS-Transport Simulation Systems. About aimsun. URL:http://www.aimsun.com/wp/?page_id=21, 2012. [Online; accessed 24-April-2012].

[41] Barry Smyth and Elizabeth McKenna. Footprint-based retrieval. In *Case-Based Reasoning Research and Development*, volume 1650 of *Lecture Notes in Computer Science*, pages 719–719. Springer Berlin / Heidelberg, 1999.