Development and Evaluation of Multiple Objects Collision Mitigation by Braking Algorithms

Examensarbete utfört i Reglerteknik vid Linköpings tekniska högskola
av

Andreas Kivrikis och Johan Tjernström

LITH-ISY-EX-3570-2004

Linköping 2004
Development and Evaluation of Multiple Objects Collision Mitigation by Braking Algorithms

Examensarbete utfört i Reglerteknik vid Linköpings tekniska högskola av

Andreas Kivrikis och Johan Tjernström

LITH-ISY-EX-3570-2004

Handledare: Jonas Jansson
Examinator: Fredrik Gustafsson
Linköping 9 December 2004
Utveckling och utvärdering av CMbB-algoritmer för multipla objekt
Development and Evaluation of Multiple Objects Collision Mitigation by Braking Algorithms

Andreas Kivrikis, Johan Tjernström

Abstract
A CMbB system is a system that with the help of sensors in the front of a car detects when a collision in unavoidable. When a situation like that is detected, the brakes are activated. The decision of whether to activate the brakes or not is taken by a piece of software called a decision maker. This software continuously checks for routes that would avoid an object in front of the car and as long as a path is found nothing is done. Volvo has been investigating several different CMbB-systems, and the research done by Volvo has previously focused on decision makers that only consider one object in front of the car. By instead taking all present objects in consideration, it should be possible to detect an imminent collision earlier. Volvo has developed some prototypes but needed help evaluating their performance.

As part of this thesis a testing method was developed. The idea was to test as many cases as possible but as the objects’ possible states increase, the number of test cases quickly becomes huge. Different ways of removing irrelevant test cases were developed and when these ideas were realized in a test bench, it showed that about 98 % of the test cases could be removed.

The test results showed that there is clearly an advantage to consider many objects if the cost of increased complexity in the decision maker is not too big. However, the risk of false alarms is high with the current decision makers and several possible improvements have therefore been suggested.

Nyckelord
collision mitigation, collision avoidance, evaluation, testing, test bench
Abstract

A CMbB system is a system that with the help of sensors in the front of a car detects when a collision in unavoidable. When a situation like that is detected, the brakes are activated. The decision of whether to activate the brakes or not is taken by a piece of software called a decision maker. This software continuously checks for routes that would avoid an object in front of the car and as long as a path is found nothing is done. Volvo has been investigating several different CMbB-systems, and the research done by Volvo has previously focused on decision makers that only consider one object in front of the car. By instead taking all present objects in consideration, it should be possible to detect an imminent collision earlier. Volvo has developed some prototypes but needed help evaluating their performance.

As part of this thesis a testing method was developed. The idea was to test as many cases as possible but as the objects’ possible states increase, the number of test cases quickly becomes huge. Different ways of removing irrelevant test cases were developed and when these ideas were realized in a test bench, it showed that about 98 % of the test cases could be removed.

The test results showed that there is clearly an advantage to consider many objects if the cost of increased complexity in the decision maker is not too big. However, the risk of false alarms is high with the current decision makers and several possible improvements have therefore been suggested.
Acknowledgments

Writing this thesis at Volvo Cars has been very interesting. We have learned a great deal about car safety, especially about some of the future active safety systems that surely will save many lives. Volvo Cars is one of Sweden’s biggest employers and it has been great to get to know such a large company from the inside.

We would like to thank the people at the Chassi & Vehicle Dynamic department at Volvo Cars for giving us this opportunity, especially our supervisor Jonas Jansson who has helped us a lot. We would also like to thank Tommy Strandelin for proofreading, our opponents Stefan Hjelm and Lars Olsson, and finally our examiner Fredrik Gustafsson.

Finally we would like to state that no cars were hurt or in any way mistreated during the testing in this thesis.
# Table of Contents

## 1 Introduction

1.1 Employer ........................................................................................................... 1
1.2 Background ........................................................................................................ 2
1.3 Assignment ........................................................................................................ 3  
  1.3.1 Main Thesis Questions ........................................................................... 3
1.4 Disposition ....................................................................................................... 3
1.5 Reading Instructions ....................................................................................... 4
1.6 Notation .......................................................................................................... 5

## 2 Theory

2.1 Set Theory and Combinatorics ......................................................................... 7  
  2.1.1 Set Theory ............................................................................................. 7
  2.1.2 Combinatorics ....................................................................................... 8
2.2 Traffic Accidents ............................................................................................. 11  
  2.2.1 Stages of a Traffic Accident ................................................................ 11
  2.2.2 Statistics .............................................................................................. 13
  2.2.3 Physics ................................................................................................ 15
2.3 Collision Avoidance ......................................................................................... 16  
  2.3.1 Frontal Collision Mitigation ................................................................ 16
  2.3.2 Collision Mitigation by Braking .......................................................... 18
2.4 Statistical Design and Analysis of Experiments ............................................. 18  
  2.4.1 Factorial Design .................................................................................. 19
  2.4.2 Response Surface Methodology............................................................ 20

## 3 CMbB Systems

3.1 Composition of CMbB Systems ..................................................................... 23
3.2 Single Object vs. Multiple Objects CMbB ..................................................... 24  
  3.2.1 Single Object CMbB ............................................................................ 24
  3.2.2 Multiple Objects CMbB ...................................................................... 26
3.3 Coordinate System .......................................................................................... 26
3.4 Movement Model ............................................................................................ 26
3.5 Object Position and Dimensions ................................................................... 27
3.6 Time Frame ..................................................................................................... 28
3.7 Input ................................................................................................................ 28
3.8 Output .............................................................................................................. 29
3.9 The Decision Maker Algorithms .................................................................. 29  
  3.9.1 Single Object Constant Control ........................................................... 29
  3.9.2 Multiple Objects Constant Control ..................................................... 30
6.1.1 Choosing an Object ................................................................. 71
6.1.2 Stop Heuristic Simulation........................................................ 72
6.1.3 Optimize Search Pattern .......................................................... 73
6.1.4 Return Escape Route ............................................................... 74
6.1.5 Object States ........................................................................... 74
6.1.6 Miscellaneous ......................................................................... 76
6.2 Minimise Known Differences ...................................................... 76
6.2.1 Object Position and Dimensions .............................................. 77
6.2.2 Tunneling ............................................................................... 77
6.3 Creating the Test ......................................................................... 79
6.3.1 Defining a Search Area ............................................................ 79
6.3.2 Setting the Filter Parameters .................................................. 80
6.3.3 Creating Object States ............................................................ 83
6.4 Test Distribution ......................................................................... 84
6.5 Running the Test ......................................................................... 84

7 EVALUATION OF CMbB ALGORITHMS ........................................ 87
7.1 Movement Model ........................................................................ 87
7.2 Calculation of Escape Routes ..................................................... 88
7.3 Positioning and Extension of Objects .......................................... 89
7.4 Tunnelling .................................................................................. 91
7.5 Collision Detection ..................................................................... 93
7.6 Testing Results ........................................................................... 93
7.6.1 Alarm Counts .......................................................................... 93
7.6.2 Alarm Times .......................................................................... 95
7.7 The Algorithms ........................................................................... 96
7.7.1 SOCC ..................................................................................... 96
7.7.2 MOCC .................................................................................... 97
7.7.3 MOHC ................................................................................... 97
7.7.4 MOES ................................................................................... 97

8 EVALUATION OF TEST METHOD ................................................ 101
8.1 Number of Test Cases ............................................................... 101
8.2 Filters ....................................................................................... 102
8.3 Other Methods .......................................................................... 105
8.4 The Test Bench .......................................................................... 105
8.4.1 Requirements ....................................................................... 105
8.4.2 Shortcomings ........................................................................ 106
8.4.3 Usability ............................................................................... 107

9 CONCLUSIONS ........................................................................... 109
9.1 Conclusions .............................................................................. 109
9.2 Future Work ............................................................................. 111
9.2.1 Test Bench ................................................................................................. 111
9.2.2 Algorithms ............................................................................................. 111

REFERENCES ........................................................................................................ 113

APPENDIX A ......................................................................................................... 115
This is a thesis for the Master of Science in Computer Science and Engineering program at Linköping Institute of Technology, Linköping, Sweden. It was carried out as a research project for Volvo Car Corporation, Gothenburg, Sweden.

This chapter gives a brief introduction to the thesis. It contains a description of the employer and some background to the thesis. It also specifies the main thesis questions and it gives some reading tips for the report.

1.1 Employer

Employer for this thesis is Volvo Car Corporation (VCC). VCC designs and produces passenger cars and is a part of Ford Motor Company, one of the largest automotive companies in the world today. VCC has for a long time been established as a leading producer of safety features, among the many features that Volvo was first to present one can find the three-point seatbelt and the laminated windshield. More information about Volvo can be found at their website www.volvocars.com and for information about Ford visit www.ford.com.
1.2 Background

One of the greatest factors in car manufacturing and ownership today is passenger safety. In Sweden alone, over 500 people are killed each year due to traffic accidents [1], and large amounts of time and effort goes into trying to lower these numbers. This is done by means of developing safety features for cars as well as for the surrounding traffic environment.

For the automotive industries there are two perspectives of passenger safety. Passive safety is all measures taken to keep the passengers safe when a collision does occur. Examples of passive safety systems are crumple zones, airbags and seatbelt pretensioners. Active safety is safety measures that are pre-emptive, meaning that the system is active even before a collision occurs. Examples of active safety systems are the anti lock braking systems (ABS), yaw control systems, lane keeping aid and even such standard equipment as break lights.

Within the last ten years the automotive industry has started producing very advanced safety features. In the cars manufactured today systems such as anti-spin and multiple airbags are more and more becoming standard. The next generation of cars will be even more advanced, car manufacturers, universities and government agencies are all developing a vast array of advanced systems to make car travelling safer. Amongst the projects published one can find automatic parallel parking, drowsy driver detection and adaptive cruise control. Many automotive companies are also researching an active safety measure called Frontal Collision Mitigation (FCM) that aims to lower the effects of frontal collisions. One way of achieving this is to use radars, lasers and/or cameras to search for imminent collisions, and when a collision is unavoidable activate a braking system that lowers the vehicle's velocity as much as possible. Such systems have been named Collision Mitigation by Braking (CMbB) systems.

Volvo is currently doing research on CMbB systems. One of the most vital parts of the CMbB system is the decision maker. This is the piece of software that tries to decide whether or not a collision is impending. In current CMbB systems the decision maker is a single object algorithm. That is, it only looks at one object at a time. The tests have given some indications to strengths and weaknesses of the algorithm but real life traffic situations often contain more than one threatening object. Volvo is therefore researching alternative CMbB decision makers that take multiple objects into consideration.
1.3 Assignment
When Volvo decided to develop several multiple objects CMbB decision makers, they realised that track testing would be unsuitable to test the behaviour and the gain of these algorithms. With several algorithms developed, extensive testing would be needed in order to compare these and to assess the benefits and drawbacks of each algorithm. To do this by driving cars towards objects would be very insufficient time wise as well as money wise.

The goal of this thesis is to develop a method for testing multiple object algorithms, and to analyse and present the results from these tests. Expected results from the assignment were a test bench and a library of interesting scenarios that show how the algorithms alter the conditions in a multiple object collision. The results could then be used by Volvo to take measures to improve or to discard their algorithms.

1.3.1 Main Thesis Questions
Questions that the thesis should answer are:

What are the benefits with multiple objects CMbB systems?

What are the strengths and weaknesses of the different CMbB decision making algorithms?

How can the algorithms be improved?

In which scenarios do strength and weaknesses show the most?

How can multiple objects CMbB algorithms be automatically tested?

1.4 Disposition
Chapter 2
Is an introduction to the theories behind this thesis. It gives a short introduction to some mathematics that has been used, and then there is a review of passenger safety, collision mitigation and their concepts.

Chapter 3
This is where the CMbB topic is introduced. This chapter discusses what a CMbB system is, what it does and how it works. Finally the CMbB algorithms that are evaluated in this are thesis presented and explained.
Chapter 4
In this chapter the evaluation method is presented and discussed. A lot of different decisions related to the testing are discussed and motivated.

Chapter 5
This chapter deals with all aspects related to the test bench that was developed. First there is a section about the design of the test bench, and then a section about the implementation follows.

Chapter 6
This chapter specifies exactly how testing was done. This includes how suitable test bench parameters were chosen and a study of some changes that needed to be done to the algorithms in order to improve test results and performance.

Chapter 7
Here all the results from the testing and different topics that affected the results are presented and discussed.

Chapter 8
In this chapter the test method and the test bench are evaluated. This includes statistics of the test bench performance and a section about its shortcomings.

Chapter 9
In the final chapter the results are concluded. There is also a section about possible future work, both regarding the algorithms and the test bench.

1.5 Reading Instructions
If the reader is only interested in the decision makers and their evaluation results it should be enough to read chapter 3, 7 and 9. But if the reader has no previous knowledge of CMbB systems it could be good thing to read chapter 2.2.1 and 2.3 first.

To understand how the evaluation was done it is necessary to read chapter 4 and 6. Chapter 4 explains how the evaluation method works and chapter 6 describes exactly how this method was applied to the testing. Chapter 8 evaluates the testing method. Chapter 5 should only be interesting to those who want to know how the test bench was implemented.

The mathematics presented in chapter 2.1 deals with set theory and combinatorics and could probably be skipped by most readers who have
taken university math courses. The part about design of experiments in chapter 2.4 can also be skipped by most readers. These methods were taken into consideration when choosing a testing method but were in the end never used.

## 1.6 Notation

Bellow follows some expressions that are commonly used in this thesis and could need some explanation.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Escape route</td>
<td>An escape route is a set of manoeuvres that a driver could use to avoid a possible accident.</td>
</tr>
<tr>
<td>Heuristic</td>
<td>The term heuristic is used in many different contexts. In this thesis a heuristic means a way to direct a search which in many cases leads to a solution being found fast. There is however no guarantees of finding a solution using this method.</td>
</tr>
<tr>
<td>Scenario</td>
<td>The input to a decision maker. In this thesis a scenario will always be a snapshot of the world model.</td>
</tr>
<tr>
<td>Test bench</td>
<td>A piece of software or hardware used to evaluate or validate other software or hardware.</td>
</tr>
<tr>
<td>Test case</td>
<td>A set of input used to test a process or system. In this thesis the process is the decision maker, and the test case will therefore always consist of a scenario.</td>
</tr>
<tr>
<td>Tunnelling</td>
<td>In science tunnelling usually represents the quantum physics phenomenon where particles move through energy barriers. In this thesis it means that objects, usually cars, move through each others during a simulation without any collision being detected.</td>
</tr>
<tr>
<td>World model</td>
<td>A world model is a system’s simplified representation of world in which it exists. In the case of this thesis, a world model represents the traffic situation in front of the car.</td>
</tr>
</tbody>
</table>
This chapter describes the theories involved in this thesis. It starts by discussing some of the mathematical basics of great importance for the thesis and then describes the nature and statistics of traffic accidents. It also defines the fields of Collision Avoidance and Frontal Collision Mitigation, and covers the theories of system validation and experiment design and analysis.

This chapter will cover theories that in some aspects apply to this thesis. All theories that are described have in some way influenced the development of this thesis, even though some fields in the end have been dismissed. Even so, the theories need to be accounted for in order for the reader to gain a wider perspective of the choices made during the development of the test bench and analysing tools.

2.1 Set Theory and Combinatorics
This thesis will in large parts deal with sets and combinations of the members of sets, so as a remainder, a short presentation of the fields of set theory and combinatorics will be made. This theory is gathered from [2].

2.1.1 Set Theory
Set theory deals with the theories on how to group objects and how groups of objects can interact and relate to other groups. The mathematical
The definition of a set is a collection of objects. Each object in the set is called an element. The exact meaning of an element does not need to be known, an element could represent a person, a symbol or a number, this does not affect the theories.

A set can be finite or infinite, depending on if it is possible to count the number of elements in the set or not. An example of an infinite set is the set of natural numbers. There can also be sets without any elements in it. This is called the empty set and is in many ways the set theory's version of the number zero.

The typical notation for a set is \{ \}. In this particular case, the set was empty, and the empty set has a separate symbol, \( \emptyset \). \( \mathcal{U} \) is also a special set. It is the set that contains all possible elements within a domain. For example, when considering the domain of all natural numbers \( \mathcal{U} = \mathbb{N} \).

The size of a set is denoted \(|A|\). Within a set the internal ordering of the elements does not matter, this means that \{1, 2, 3\} and \{1, 3, 2\} is considered to be the same set. An element can only exist once in the same set and when an element is part of a set, it is said to be a member of that set. When the order of the elements matter, and when elements can occur several times, it is no longer recognised as a set but as an ordered list, or just a list.

There are many possible interactions that can be made between two or more sets. The union of two sets, \( A \) and \( B \), is a set that contain all elements that are either in \( A \) or \( B \). The notation for this is \( A \cup B \). The intersection of two sets is a set of all elements that is in both \( A \) and \( B \), and is written \( A \cap B \).

One can also negate sets, denoted \( \neg A \). Negating \( A \) result in all elements in \( \mathcal{U} \) that is not in \( A \).

A set can also be part of one or more other sets. This means that all elements in a set \( A \) are also a member of another set \( B \). \( A \) is said to be a subset of \( B \) in such case, and \( B \) can also be said to be a superset of \( A \). The notation is \( A \subseteq B \). The definition reads

\[
\text{if } A \cap B = A \text{ then } A \text{ is a subset of } B \text{ and } B \text{ is a superset of } A.
\] (2.1)

2.1.2 Combinatorics

Combinatorics is the mathematical theories on how to combine elements in a set. Typical questions answered with combinatorics are: How many possible Poker hands can be made with a regular deck of cards? If a car is
made with the choice between three different motors and five different colours, how many possible different versions can be made?

In the first question there are only one set, the set of the 52 playing cards. The task is to pick five members from this set, which means that the set will change during the picking of cards. In the other question however, there are two constant sets. The first set contains three elements, the different motors. The second set contains five elements, the five colours. The problem now is instead how to choose one element from the first set, and one element from the second set.

An important law in combinatorics is the multiplication principle, stating that

\[ \text{If } A_1, A_2, \ldots, A_n \text{ is finite sets, then} \]
\[ |A_1 \times A_2 \times \ldots \times A_n| = |A_1| \times |A_2| \times \ldots \times |A_n| \]  \hspace{1cm} (2.2)

This is the basis in combinatorics and it can be used to answer both questions above.

Several methods and rules on how to choose elements from sets have become so important that they have got their own name.

Permutation is one of those. A \textit{k-permutation} of a set \( A \) is a list with \( k \) of the elements in \( A \). As an example Figure 2.1 show the 1-permutation, 2-permutation and 3-permutation calculations from the set \{1,2,3,4\}. The number of k-permutations for a set with \( n \) elements is denoted \( P(n,k) \) and the mathematical definition, based on equation (2.2), is

\[ P(n,k) = n \times (n-1) \times \ldots \times (n-(k-1)) = \frac{n!}{(n-k)!} \]  \hspace{1cm} (2.3)
Combination is a very similar concept to permutation, but where permutation returns a list, combination instead returns a set. A \( k \)-combination of a set \( A \) is a subset of \( A \) containing \( k \) elements. Figure 2.2 show the 1-combination, 2-combination and 3-combination output from the set \{A,B,C,D\}. The number of possible \( k \)-combinations of a set with \( n \) elements is denoted \( C(n,k) \) and the mathematical definition is

\[
C(n,k) = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-(k-1))}{k \cdot (k-1) \cdot \ldots \cdot 1} = \frac{n!}{k!(n-k)!}.
\]  

(2.4)

One can see that \( P(n,k) \) and \( C(n,k) \) are very similar and that in fact

\[
C(n,k) = \frac{P(n,k)}{k!}.
\]  

(2.5)

This is because permutation returns lists and when transferring these to sets each set will appear \( k! \) times.
2.2 Traffic Accidents

The goal of this thesis was to develop evaluation tools for an automotive safety feature. The safety feature aimed to mitigate or avoid collisions and a small presentation of the statistics and physics of collisions will therefore be made. A small section will also discuss the different stages of a collision and the aids that might be available in each stage.

2.2.1 Stages of a Traffic Accident

Traffic accidents can generally be divided into five different stages [3]. Each stage is defined by the relation between the host vehicle and the object of the collision. The transfer from one stage to the next is made as the relations change. The stages are depicted in Figure 2.3.

The first stage is normal driving. In this stage there very low or no risk for a collision. Driving aids as adaptive cruise control and yaw control can be active. If an object suddenly becomes threatening, due to heavy braking, change in direction or because a new object has been discovered the second stage will be entered.

Figure 2.2
The 1, 2 and 3 k-combination operations on the set {A,B,C,D}.
In this stage there is a high risk for a collision but the collision can still be avoided. If the collision is to be avoided counter measures need to be taken during this stage. If the host is equipped with a collision avoidance system it can execute an escape maneuver. Another counter measure is to warn the driver through audible, visual or tactile displays, in hope that the driver will execute an evasive maneuver. If none of this is done the third stage will be entered.

In the third stage a collision is unavoidable, the risk is not only high, the collision is definite. The host is no longer able to avoid the collision. Instead the goal is to mitigate the collision as much as possible. Typical systems that need to launch their counter measures during this stage are seatbelt pretensioners, airbag preinflations and preemptive whiplash protection systems.

The fourth stage is the actual collision. This stage is entered as the host and the object make physical contact. Up to this stage all counter measures have been considered active safety features. The systems that launch during the fourth stage are passive safety features. But there are some strong relations between the active safety features of the third stage and the passive safety features of the fourth stage. Typical passive safety systems are seatbelts, airbags and whiplash protection systems, so it is clear that safety features in stage three can be used to prepare and improve the safety features of stage four.
2.2 TRAFFIC ACCIDENTS

The fifth stage is post collision and in this stage analysis of the systems behaviors can be made so that improvements can be made to upcoming versions.

2.2.2 Statistics

When studying traffic accidents the most useful and reliable available data is statistics of real accidents. The statistics are often gathered for national transport institutes, insurance companies and car manufacturers, and the contents of the reports are heavily dependent on the publisher. Using statistics to find typical traffic situations is very hard, in the anonymity of large numbers the traffic situations is often bundled into large groups. Finding data on scenarios where multiple objects are considered have proven hard, this is often not of primary interest of the organisations gathering the statistics.

When looking at annual statistics of traffic accidents in Sweden [4] some trends become clear none the less. During the time period of 1960 – 2001 the number of traffic accidents that cause injuries have varied between 14,959 and 19,338 and showing only a marginal decrease over the long run. The number of people killed or injured gives the same picture, the number lies between 19,338 and 26,243, and there is no real change in average over the 42 year period.

When looking at the number of fatalities a different trend is clear though. Until 1979 the number of fatalities was consistently above 1,000 while during the time period of 1994 to 2001 the number was consistently below 600. When averaging, one can see that the number of fatalities actually was cut in half in the last 25 years.

During the same period the number of registered vehicles grew from about 2.5 million to 4 million cars but despite this one fact remains, the number of casualties is lowered at a greater rate than the number of collisions resulting in injuries. This could be seen as an indication that most developed safety features have been passive and that many injuries can be avoided with active safety measures.
Figure 2.4
The number of traffic accidents reported to the police in Sweden during the years 1960 – 2001.

Figure 2.5
The number of killed or injured in police reported accidents in Sweden during the years 1960 – 2001.
2.2 Traffic Accidents

Traffic statistics also show that there is a great correlation between velocity and the risk for injuries. This is indicated by the law of kinetic energy

$$E_k = \frac{mv^2}{2}. \quad (2.6)$$

This law states that the energy of a car have a quadratic relation to the car’s velocity. Creating a system that lowers collision velocities can be greatly rewarded in form of lesser energies involved in the crashes, which in turn would lower the chance of injuries.

This assumption is confirmed by accident research when studying the relation between the change in velocity at the moment of impact and the probability of fatality. For example, crashing into a fixed obstacle at 100 km/h has probability of mortal outcome of 0.59. But if the speed is reduced to 85 km/h the probability drops to 0.31. More information and complete statistics can be found in [3], [5] and [6].
2.3 Collision Avoidance

Collision avoidance (CA) is a very broad term, covering absolutely every situation where objects may collide. It is most commonly used in avionics, seafaring and automobiles, but the term is also often used for industrial robots, autonomous vehicles and for non-physical objects in computer games and simulations as well.

Collision avoidance is a domain independent in the sense that one can not deduce any general information about preconditions, post conditions or time factors for CA systems. A CA function could be applied to just about any moving object, no matter the domain the object act in. For instance, one can compare TCAS\(^1\), a CA system used in avionics, with a CA system for a character in a computer game. TCAS relies on radar to detect objects, works in a 3-D environment, only gives suggestions on countermeasures and must never fail. A computer character on the other hand might have perfect world knowledge, work in 2-D, actively controls the character and might accept to bump into objects every now and then. The only actual similarity between the two systems is the ultimate goal of avoiding collisions between objects.

Looking at the stages of a collision in chapter 2.2.1 CA is described as a counter measure for stage two. There is a high risk for collision but it can still be avoided. The system can avoid the collision through a series of acceleration, deceleration and/or steering manoeuvres without consulting the driver. There are also CA systems for cars that work by producing warnings to the driver when the risk for a collision is alarmingly high. These systems are often specified as collision warning (CW) systems since they rely on the driver to execute an evasive manoeuvre.

2.3.1 Frontal Collision Mitigation

Frontal collision mitigation (FCM) is the automotive industries name for systems that try to mitigate collisions between a car and any object in front of it. The object can be other vehicles but also pedestrians, animals or stationary objects such as road signs. FCM can in some sense be called a CA system since the optimal mitigation of a collision is a complete avoidance, but in most cases FCM will not be able to avoid a collision completely. Another difference between FCM and CA is that FCM can take several measures that a general CA system could not. There are several

\(^1\) Traffic Alert & Collision Avoidance System, used in airplanes to detect and avoid other airplanes while airborn.
possible actions apart from braking and steering that a FCM system could make when an imminent collision is detected. As an example it could lower the chassis of the car so that the car better can absorb the energies in the crash, other actions is to activate seat belt pretensioners, preinflate the airbags and to activate similar systems.

It is important to differentiate CA and FCM systems. It is the similarity between avoiding and mitigating a collision that at first glance makes the systems seem identical, but when studying the systems more carefully one can realise that they certainly are not.

First of all the systems are active during different stages of a collision as described in chapter 2.2.1. CA systems must launch their counter measures in stage two, while a FCM system should launch only in stage three. The job of a FCM system is therefore to decide when the third stage is reached, i.e. when a collision become unavoidable. FCM should also strive to detect the transfer to the third stage as early as possible since a larger degree of mitigation then can be reached.

Another difference is the possible post conditions. A CA system only tries to avoid a collision, and in that sense any collision will be a failure. For a FCM system this statement does not hold. When the goal is to mitigate a collision different counter measures can be taken. The counter measure mentioned earlier, the activating of seat belt pretensioners, is a good example of this. Pretensioners will in no way be able to avoid crashes, but it will in nearly every case mitigate them.

One more reason for the automotive industry to differentiate FCM from CA is the fact that they both often are used as intervening systems. Intervening systems will seize control of the vehicle and thus remove the control from the human driver, obviously a very controversial action. No driver will accept that they involuntary may lose control of their vehicle, even if it would be to an advanced computer system. That the system will perform better or as good as any driver is not condition enough to give it control of the vehicle, but the driver must also accept that it does. The result is that the system may not intervene if the driver might be in control and can thus only intervene when a collision truly is unavoidable. At any time up till that point, the system must assume that the driver is able to correct the situation without help.

This means that FCM by definition cannot be a CA system. Since FCM launches its counter measures when the collision is already unavoidable the collision simply cannot be avoided, but in best case mitigated into a simple fender bender.
FCM systems typically use radar, cameras, sonar or laser to sense objects in front of the car. These devices are chosen based on field of view, reliability and precision, and as a general rule, objects will be seen and their positions will be accurate enough for the FCM to make proper decisions.

2.3.2 Collision Mitigation by Braking
Collision Mitigation by Braking (CMbB) is one possible subclass of FCM. This system uses a braking intervention to reduce the collision speed in an imminent collision. The aim is to avoid the collision all together but this is often not possible since it in many cases would be more efficient to veer from the object. This means that the collision could be avoided by steering, or by a combination of braking and steering from by driver, but not by the CMbB system. If the driver fails to do so, the collision will be unavoidable, and only then will CMbB brake the car. This is to avoid situations where the CMbB system reacts to easily and brake in situations where the driver is in full control of the traffic situation.

2.4 Statistical Design and Analysis of Experiments
Statistical design and analysis of experiments is a field of mathematical statistics and product design and development. As the name suggests it focuses on experiments, and how these should be planned and conducted in order to make the testing as efficient as possible.

Large parts of statistical experiment cover the use of statistical tools such as variance and mean values. This will not be covered since it does not apply to this thesis. The reason for this is that the tests are deterministic, so any repetition of tests will result in the exact same output.

The parts that do apply to the thesis are two: factorial design and response surface methodology [7]. These are both powerful tools to find extremes in system or process behaviour, with a low number of experiments.
2.4 Statistical Design and Analysis of Experiments

2.4.1 Factorial Design

The traditional way of testing is to keep all but one factor constant. In this manner one get direct knowledge on how the varying factor affect the output, but this result in that many tests need to be made and correlations between factors might pass unnoticed. With factorial design, one does not test each factor separately, but all k-combinations are created and tests are made where these factors are varied. As the number of controllable factors grows this method will span the test domain with much less tests than what would be needed in an exhaustive test. This also allows for the factors to correlate to each other, and this will show in the output.

As an example, imagine the theoretical testing on a car's crumple zone. The crumple zone is interesting since it must be strong enough to withstand small collisions, while not to strong, it must also be weak enough to be able to absorb the energy in a serious accident. Say that the controllable factors of interest are number of crumple beams, crumple beam size and crumple beam material strength. With traditional design, one might test one of these factors at a time. The results would probably be that there should be a medium number of beams, the beam size should be medium and the beam material strength should be medium. However, this might not generate the best possible crumple zone. There are correlations between the factors that can not be measured with this method. For instance, it is fairly obvious to say that the more beams there are, the lesser the material strength are needed. It is also safe to say that large beams also result in lesser strength is

\[\text{Inputs} \rightarrow \text{Process} \rightarrow \text{Output}\]

\[x_1, x_2, x_p, z_1, z_2, z_q\]

\[\text{Controllable factors}\]

\[\text{Uncontrollable factors}\]

\textbf{Figure 2.7}

The figure shows a general model of a process or system. As well as input and output there may be several factors affecting the process. They may be controllable or uncontrollable.
needed. One could also argue that it would be beneficial to lower the
number of beams while raising the material strength. This leads to
contradictions, should the strength be high or low? Where will the results
intersect into an optimal crumple zone?

A possible alternative is to conduct factorial experiments. Given the set of
controllable factors \{number, size, strength\}, all possible 2-combinations of
this set is created. By varying the variables in these sets while keeping the
others (in this case the last one) constant, information about the
correlations between the variables can be gathered. This means that the sets
\{size, strength\}, \{size, number\}, and \{number, strength\} should be tested.
These tests would result in a three-dimensional result domain, where the
optimal crumple zone can be located.

The strength in factorial testing is that one is able to span the result domain
with a low number of tests. To span the result domain one could also
conduct all-factor testing. This would consist of conducting experiments
where all factors change at the same time. This will often be possible when
there are only three controllable factors, but as this number rises the
number of tests that need to be made quickly become unmanageable.

### 2.4.2 Response Surface Methodology

Response surface methodology (RSM) uses multiple variable analysis to
calculate gradients for multi-dimensional surfaces. It also uses statistics to
determine whether or not correlations exist among factors.

The basis of RSM is to choose a small test area from the test domain and to
conduct tests within this small area. Given the results one can calculate a
gradient for the results. This gives an indication to the direction in which
the optimum lies. The next step is to move the test area in the direction of
the gradient, and this can then be iterated until the optimum is trapped
within the test area.

Among the important design aspects of RSM testing is how to choose
starting points, the geometry of test areas and how to move the test area in
accordance to the gradient. RSM can be a very useful tool, but it requires in
depth knowledge in the system by the experiment designer. One needs to
have an outline of the result domain in order to know how far to move the
test domain, as well as to be able to analyse test coverage.

Several problems can occur during the experiments. For example there lies
great risk in missing saddle points and ridges in the results, there are also
the problems of local and global extremes and how to guarantee the finding
of such. There is also the possibility that if the test domain is not chosen or
moved wisely, the number of needed iterations can become large and in worst case even infinite.
3 CMbB Systems

This chapter describes the workings of a CMbB system. It describes the CMbB algorithms to be evaluated and discusses the way that the algorithms detect collisions, how they search for possible evasions and some common properties of interest.

This thesis focuses on the development and evaluation of algorithms within CMbB systems. In order to understand the problems and solutions within the thesis it is therefore crucial that the definition and subparts of a CMbB system is unambiguously described.

3.1 Composition of CMbB Systems

When discussing CMbB, it is important to define the parts that a CMbB system consists of as well as the tasks of these. When looking at the hardware, a CMbB system generally consists of radar, laser or vision sensors for input, a computer for analysing and a braking system for counter measures. The hardware is used to find threatening objects and assess their widths and lengths, the computer to process the data and produce a decision and the braking system to carry out counter measures.

When looking at software, there are two crucial parts, the tracking algorithm and the decision maker. The tracking algorithm is the software that takes in the data from the input hardware and converts this into a world model. Typical tasks are to minimise errors in state estimates and to
update the world model when new data is available. This is mainly a signal processing task and is not covered by this thesis.

The focus of the thesis is instead the decision maker. The decision maker is the software that uses the world model to calculate the probability of collision. General problems for the decision maker are to choose a dynamical model, to predict the movement of the objects and to search for possible escape routes. Throughout this thesis, when discussing the algorithms or decision makers it will refer to decision making algorithms if not clearly stated otherwise.

3.2 Single Object vs. Multiple Objects CMbB

CMbB systems are generally divided into two groups, single object CMbB and multiple objects CMbB. These two approaches to CMbB give very different conditions for the development of suitable tracking and decision making algorithms. In this section a small introduction to the two alternatives will be made.

3.2.1 Single Object CMbB

A single object CMbB system works by designating one object as the most threatening and then looking for ways to avoid this object. This can be done in many manners, and there are several important design factors. For one, there are various ways to define which object is the most threatening. One way is to take the object closest to the host vehicle, another is to choose the object with the lowest time to collision (TTC).

Another important design factor is where to put the extraction of the most threatening object. In the example with the TTC, the task can easily be assigned to the tracking algorithm. When this is the case the CMbB system is said to have a single object tracking algorithm. Even though it actually is tracking several objects, it only passes one object to the world model of the decision maker. There are several benefits with this solution. In such a CMbB system the decision maker always know that there will only be one object to take into account, which allows for the decision maker to be very fast and simple. The drawback is that it has to take the decisions on a simplified world model and the decisions might therefore not be correct.
The extraction of the most dangerous object could instead be assigned to the decision maker. In that case the tracking algorithm must be a multiple objects tracking algorithm, in the sense that it must put all discovered objects into the world model. The decision maker can then be said to be a single object decision maker since it, even though it has several objects in the world model, extracts one and only tries to avoid that one.

One benefit with this approach is that the decision maker can apply its escape route searches on all objects and designate the object with the fewest escape routes as the most dangerous. This gives a higher assurance that the most threatening object actually is chosen, as long as the tracking algorithm approximate the objects state estimate well enough.

**Figure 3.1**

Three examples of how the single object and multiple objects properties change the conditions for the tracker and the decision maker. In example 1 both the tracker and the decision maker is single object. This means that the tracker will only allow one object in the world model. Knowing this, the decision maker needs extremely few calculations to make a decision. In example 2 the multiple object tracker put all objects in the world model and the single object decision maker is more likely to chose the best object to avoid. It may still collide with the objects ignored though. In example 3 all objects are considered throughout the process, returning a decision that avoids all objects if possible.

<table>
<thead>
<tr>
<th>Example</th>
<th>Real world</th>
<th>Tracking</th>
<th>World model</th>
<th>Decision Making</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image" alt="Single object" /></td>
<td>Single object</td>
<td><img src="image" alt="Single object" /></td>
<td>Single object</td>
<td><img src="image" alt="Decision" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image" alt="Multiple objects" /></td>
<td>Multiple objects</td>
<td><img src="image" alt="Single object" /></td>
<td>Single object</td>
<td><img src="image" alt="Decision" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image" alt="Multiple objects" /></td>
<td>Multiple objects</td>
<td><img src="image" alt="Multiple objects" /></td>
<td>Multiple objects</td>
<td><img src="image" alt="Decision" /></td>
</tr>
</tbody>
</table>
3.2.2 Multiple Objects CMbB

The gain with single object CMbB is that it becomes very cheap in term of time and memory to search for escape routes and thus make a decision. The drawback is that it can become somewhat pessimistic in situations where more than one object is present. This is why Volvo decided to research multiple objects CMbB. Since multiple objects CMbB is a much more complex problem, there were no obvious solution and Volvo decided to develop several different decision makers.

The drawbacks of single object CMbB come from the fact that the set of escape routes for each object is considered separately. The result is that the escape route from one object can lead into a collision with another object. The multiple objects CMbB works by considering all objects when searching for escape routes. This means that the algorithm always works against a joint set of escape routes. This may require more advanced manoeuvres from the host. To avoid a single object, there are really only three factors to consider. How hard must one brake, how much one must steer to pass the object on the left and how much must one steer to pass the object on the right. With multiple objects this is not sufficient, for example the host vehicle might need to brake in order to avoid one object and steer in order to avoid another.

3.3 Coordinate System

All algorithms use an earth fixed Cartesian coordinate system. All scenarios start with the host front centred on origin, facing the positive $\hat{x}$ direction. The following notation will be used throughout this document:

- $p_x$ - Position in the $\hat{x}$ direction
- $p_y$ - Position in the $\hat{y}$ direction
- $v_x$ - Velocity in the $\hat{x}$ direction
- $v_y$ - Velocity in the $\hat{y}$ direction
- $a_x$ - Acceleration in the $\hat{x}$ direction
- $a_y$ - Acceleration in the $\hat{y}$ direction.

3.4 Movement Model

All algorithms use the constant acceleration model to estimate the movement of objects. This is a rather simple model that is common in
tracking systems. It can also easily be modified into a constant velocity
model simply by disregarding the acceleration. The model is defined by the
following equations:

\[
x = \begin{pmatrix} p_x & p_y & v_x & v_y & a_x & a_y \end{pmatrix}^T
\]

(3.1)

\[
x_{t+1} = \begin{pmatrix} 1 & 0 & T & 0 & \frac{T^2}{2} & 0 \\
0 & 1 & 0 & T & 0 & \frac{T^2}{2} \\
0 & 0 & 1 & 0 & T & 0 \\
0 & 0 & 0 & 1 & 0 & T \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} x_t.
\]

(3.2)

There is no noise in the model. The reason for this is that the model is
internal for the decision maker. Since the decision maker only uses the data
in the world model no new errors will be introduced during the decision
making. More important, since tracking will only be simulated during
testing, it can also be decided that no errors should be introduced in the
tracking phase. In a complete CMbB system the situation will be different.
In the complete system, the input sensors will contain erroneous state
estimates that will be transferred into the world model by the tracking
algorithm

3.5 Object Position and Dimensions

In the object states there is a position in the form of \(x\) and \(y\) coordinates. It
does not give any information of the object length, width or facing direction.
In a final system, these states would be given by the tracking algorithm as
well but at this point all objects are assigned a uniform size and an
extension relative the position and velocity. A prerequisite for evaluation is
that all algorithms do this uniformly, otherwise they will interpret scenarios
differently and the test results might be inconsistent. When this prerequisite
is not fulfilled the decision makers will be fed the same objects but create
different scenarios based on that data. The test case would then prove
useless since it is not possible to compare the output from the decision
makers.
During the testing all algorithms will assume that objects are rectangular, they will be 1.8 meters wide and have a length of 0.2 meters. They will also assume that the rectangle will extend 0.9 meters in each direction of the $y$-axis and 0.2 meters in the direction of the $x$-axis relative the state position. This means that the objects will always extend away from the host car and will always be aligned to the earth fixed coordinate system, no matter the direction of velocity or acceleration vectors. The reason that objects are assigned this size will be discussed in chapter 6.2.1.

3.6 Time Frame
All algorithms simulate the movement of the objects and the host in order to find escape routes. In order to keep the number of simulation steps low all algorithms have a time horizon of two seconds. This means that it is assumed that no collision can be found more than two seconds before the actual event, so the algorithms should not bother with trying to.

3.7 Input
The decision maker takes the world model as input. The world model consists of the position, velocity and acceleration of all the objects. The decision maker does not care about how the input is gathered, and from the decision makers point of view input will always be available, and it will always fulfil the requirements on the system. The job of gathering and evaluating the data belongs to the tracking algorithm and if there is any reason to believe that the input is not trustworthy, the decision maker should be by-passed or ignored. This means the decision maker can be seen as constantly working with a good enough world model, it will always contain erroneous state estimates, but they will be acceptable. In other case the tracking algorithm, or a control process, would detect this and bypass the decision maker.

The position of the objects is represented by a point, but of course any object in real life objects will have width and depth. In a CMbB system, this data will be given in the object's state as well, but for the evaluation of the algorithms, all objects were assigned fixed dimensions. This was done since the decision maker algorithms did not use object states that defined the size of the object. It also lowered the number of tests that needed to be made, since there were no variations in size to account for.
3.8 Output

During the development end evaluation of the algorithms the output will only be a Boolean value signalling whether a collision is imminent or not. In a final product the decision maker will probably not do this, it is more likely to result in a Fuzzybool\(^1\) representing the probability of collision. The reason for returning a Boolean is the absence of noise and erroneous state estimates during testing.

Even though the only return value is a Boolean, there are still some interesting data that is useful during testing of the algorithms. The most important one is the escape route. If an algorithm does not alarm it has found an escape route. It is vital for the system that this escape route is correct, meaning that it must not cause a collision and it must be possible to carry out the escape manoeuvres. Studying escape routes will be the primary method of finding unforeseen behaviours of the decision makers.

It is important to be aware of the fact that the given escape route is only a theoretical escape route. The system will not attempt to steer the car on this route, but one can rather see it as the CMbB system saying: "If the driver would take this route there will be no collision, so I should not activate braking procedures". The CMbB system has no means of controlling the vehicle other than to apply braking, and this will be done when the CMbB system fails to find an escape route, or in other words, when a collision is unavoidable.

3.9 The Decision Maker Algorithms

In order to gain a better picture of the testing an introduction to the algorithms, and how they search for escape routes, are needed. The decision maker algorithms are as follows.

3.9.1 Single Object Constant Control

The Single Object Constant Control (SOCC) algorithm was the first to be implemented. SOCC works by calculating the lateral or longitudinal acceleration needed to avoid each object. It assumes that the same acceleration will be applied during the entire escape manoeuvre, hence the name constant control.

---

\(^1\) A Fuzzybool is a continuous Boolean variable that can be assigned any value between 0 and 1. A Fuzzybool of 0.5 indicate that there is a 50% chance that the statement is true.
SOCC calculates the time to collision for each object and suggests an escape route that avoids the object with the lowest TTC and that is on a collision course with the host. The single object attribute come from the fact that the algorithm looks at one object at a time and if any object lacks an escape route the algorithm will alarm.

As secondary output the algorithm suggests an escape route when it does not alarm. The escape route will be chosen with the lowest acceleration needed to avoid the object, often resulting in a path that brings the host very close to the object without ever making contact.

3.9.2 Multiple Objects Constant Control
Multiple Object Constant Control (MOCC) works very much like SOCC, but instead of simply iterating over each object, it intersects the set of escape routes for each object. If the total set is empty, this means that there is no escape route that avoids all objects, and the algorithm alarms. The difference towards SOCC is that in this manner the escape routes for each object is compared to the escape route of the other objects and only the escape routes that escape all objects are kept.
As with SOCC the proposed escape path will be the lowest possible acceleration that avoids all objects. This means that the host will try to make as small counter manoeuvre as possible, often resulting in a near collision.

### 3.9.3 Multiple Objects Heuristic Control

Multiple Objects Heuristic Control (MOHC) in turn is an extension of MOCC. MOHC uses the same method as MOCC to search for escape routes, but utilizes the two seconds time line better. It does this by simulating the world model and predicts if an escape route will appear in the close future. MOHC assumes that the driver of the host will steer towards the most open space if an escape routes exist and that the driver would brake when an escape route does not exist.

Figure 3.3

*MOCC calculates the lateral and longitudinal acceleration needed to avoid each object and join these. If there is no escape route that avoids all objects the algorithm alarms.*
This gives the algorithm the freedom that constant control counter measures must not be applied during the two seconds but rather that the counter measure can be revaluated several times. The result is that the host can break for an imminent collision, only to after some time find an escape route that was not there a split second ago.

MOHC uses the same means of finding escape routes as SOCC and MOCC, but since it revaluates several times it wants a bigger marginal then the previous algorithms. It does this by always choosing the centre acceleration of the biggest continuous interval of possible accelerations. The result will be that the host will steer into the largest available gap between the objects.

3.9.4 Multiple Objects Exhaustive Search

Multiple Objects Exhaustive Search (MOES) is the most advanced of the algorithms. It calculates a set of routes that a human driver would be able to take, given a simple model of human driver behaviour. It then searches through the routes after a route that does not collide with any of the objects. If none is found, there is no escape route and the algorithm alarms.

MOES searches through the tree depth first and stops as soon as an escape route is found. The first tested counter manoeuvre is to steer full left and the result is therefore that the escape routes will include an unproportional amount of left turns.
3.10 Properties of the Algorithms

There are several factors that were of interest for the development of a test bench. These are factors that might not initially been thought of when the algorithms were developed, but they are there and can be exploited for the testing none the less.

3.10.1 Symmetry

One of the most powerful properties of all algorithms is that they are symmetrical. For almost every traffic situation there exists another one that is a longitudinal reflection of the first, and which will always result in the same output. This means that there is an ability to cut testing in half without losing test coverage. The reason that not exactly every scenario has an opposite is that objects can be placed on the symmetry axle, and thus are symmetric to itself.

3.10.2 Derivability

Another property of all algorithms is the derivability of test results from tests with fewer objects. That is, if a set of object states generates an alarm from an algorithm, then all supersets of this set will also cause an alarm. This means that an alarm for a scenario with one object can be transferred to several scenarios with two objects and many more scenarios with three objects. Given that there are $n$ different objects and that there are $k$ objects then the number of scenario results that can be derived are $C(n, k)$.

3.10.3 Complexity

None of the algorithms has any significant usage of memory, they are all implemented in a fashion that a very low number of variables need to be saved.

When looking at time complexity there are some differences though. SOCC and MOCC stand out by far, by being very quick. They only have to calculate a small number of variables which means that only a low number of operations are needed in order to make a decision. MOHC also has a small number of operations, but needs to repeat them many times since it is able to change control manoeuvres. It is slower than the previous two decision makers, but it is still within acceptable limits. MOES however, shows great problems with time complexity, even with a quite simple model of human driving behaviour. This is because the search tree grows exponentially, and when the branching factor is the number of possible manoeuvres by the driver the tree grows large rapidly.
This chapter presents the plan of development. It will discuss the methods decided upon for testing and evaluation of the decision makers. It also describes some of the problems that were encountered as well as the solutions to these.

When evaluating algorithms there are many aspects that can be taken into consideration, which one chooses to look at heavily depends on the state of development. In this thesis more emphasis was put on what the algorithms can do, then on how they would do it. This means that there were no defined demands on time complexity or memory complexity. The goal of the evaluations was instead focused on verifying the behaviour of the algorithms, both in terms of correctness as for how they performed in term to each other.

4.1 Plan of Development

Some collision avoidance systems can be proven correct with formal methods, see [8] for an example of how the avionics system TCAS II was validated. The CMbB algorithms correctness can however not be proven mathematically or logically. Even if they could, one would still need extensive testing to be sure that the implementation was correct. The goal of the evaluation was not only to prove correctness, but also to evaluate output from different algorithms against each other. When the literature study failed to provide any suitable method for conducting the evaluation, it
was decided that the algorithms would be evaluated by some form of exhaustive search, by creating a large number of test cases and run the algorithms on these.

This would result in a large matrix of alarms from the decision makers which could be processed in order to find conflicting results as well as beneficial or unfavorable behaviors. By doing exhaustive testing it can also be asserted that large amounts of diverse scenarios will be tested, which in turn will lead to wide coverage of the decision maker's behavior. It will also be advantageous for software validation, since it should require all parts of the code to execute.

An exhaustive testing presents many problems. How many test cases can be made, how many tests can be performed and how would the tests be made? A method for how results should be gathered must also be defined and most important, how can it be shown that the results are valid? These were important questions that needed to be addressed before designing and implementing.

Statistical design and analysis of experiments is an area that in many respects can seem suitable or even preferable within the scope of this thesis. In many aspects it surely does, but there are some strong factors that ultimately led to the fact that factorial design as well as response surface methodology was disregarded as possible testing methods.

First of all they both have a basis in statistics. They are both meant to be used in experiments where there are anomalies from each test occasion. This is not the case with evaluation of CMbB decision makers though, given a scenario a decision maker will always produce the same decision so there is no use in repeating tests. Nevertheless, this is not enough by itself to dismiss the methods. They could both still be used in order to find object states that result in great threats to the host. Statistical design contain many methods for finding extremes in systems and processes but for the evaluation of the decision makers the extremes are not what is interesting, it is the borderline scenarios. Borderline scenarios are scenarios in which all objects by themselves might be harmless, but when combined produce a scenario where either only a very small number of escape routes exist, or where no escape routes exist but only a small variation of the scenario would produce escape routes.

There are also other ways to reduce the number of test cases. One popular method is called Combinatorial Design and can be read about in [9] and [10]. This method is very powerful when little is known of how variable correlates, but since it is relatively easy to identify interesting test cases in the case of CMbB, this reduction of test cases can be done in a better way.
4.2 Defining the Test Domain

To start with a domain from which the test cases could be gathered needed to be defined. This meant that a world model suitable for the decision makers had to be defined. A time complexity analysis was also necessary, in order to get an estimate of how many tests that was possible to conduct.

Defining the test domain was a rather simple task. It is common to use radar and laser to find objects in the path of the car. Both radars and lasers give a width, height and distance as output. But with the devices commonly used the height dimension is so small, about 5°, that the height dimension can be neglected.

4.2.1 Degrees of Freedom

Defined by the object states needed for the algorithms the degrees of freedom was set to position, velocity and acceleration. In order to make a general test bench it was decided that the number of possible positions, velocities and accelerations would not be fixed, but instead be generated from input arguments to the test bench. This would enable users to easily edit the size of the tests depending on the goal of the testing.

Object dimensions such as width and length was intentionally left out, and all objects were instead fixed uniform dimensions. The main reason for this was that it would keep the number of test cases down while the decision makers' behaviours would still be challenged.

4.2.2 Search Area

When defining the search area the goal was to make it as small as possible in order to allow for high resolution of the starting positions. However, the search area must not be too small, otherwise interesting scenarios could be excluded from testing. The nature of the FCM systems allow for every point not in front of the host to be eliminated, a frontal collision with objects behind the car is highly unlikely. For the area in front of the host, no restrictions should be made though. The reason is versatility, in case larger tests should be made in the future the search area need to be able to adapt.

It was decided that two possible ways to define the search area should be available. The first area was a rectangular shape in front of the car. The reason for choosing the rectangular shape was to simulate the layout of a road strip. The second shape was a circle segment. This was based on the workings of radar and lasers, which often search in an angular pattern creating a circle segment.
Generate Object States

All objects are represented with states. A state is a set of information about the object that the decision makers need in order to produce valid output. Given that there is a uniform model for how the objects behave only the initial state is required, since all subsequent states can be calculated from the initial one. Since the movement model of the algorithms are a constant acceleration model the object states for objects consist of position, velocity and acceleration.

The generation of object states has the single most impact on the number of tests that will be made. This part is therefore completely dependent on input arguments in order to make the test bench more flexible. The input arguments needed are used to create the starting positions and to define possible velocities and accelerations for the objects.

All variables are specified with a set of tuples. In the case of positions the tuples represent points, for the other variables the tuples represents vectors. The points and vectors can be given in Cartesian or polar coordinate systems, this makes easy to specify a rectangular or circle segment search area. The input parameters that specify the objects states are thoroughly described in 5.3.1.

\[\text{Figure 4.1}\]

The search area can be defined as either a rectangle or as a circle segment.
This very flexible way of specifying object states also makes possible to use different lengthwise spacing. The spacing can be varied so that the distances are either constant or grow logarithmically. The logarithmical alternative will generate more states close to the car, where there is a greater likelihood that they will be a threat.

It is also possible to add noise to the starting positions so that test can be repeated with the same input but will generate different scenarios. With this function an area can be well covered without the need of generating a vast amount of stating positions. It also removes the element of organisation that is introduced when objects always are placed at perfectly aligned positions.

4.3 Time Complexity Analysis

Given the search area and the degrees of freedom, it was possible to make a time complexity analysis. The test would consist of $k$ number of cars, $p$ possible starting positions, $v$ velocities and $a$ accelerations. This means that the total number of possible test scenarios $S$ would be

$$S = (p \cdot v \cdot a)^k.$$  \hspace{1cm} (4.1)
All $S$ scenarios do not need to be tested though. In order to span the set it is only necessary to test all combinations $C$ that exist in the set. The number of test cases $C$ would then be

$$C = \frac{(p*v*a)}{k!(p*v*a-k)!}.$$ \hspace{1cm} (4.2)

This is surely a much more manageable number, but it was still not good enough to enable a test of a sufficient set of scenarios. A good example for this is a test with 50 positions, 5 velocities, 5 accelerations and 3 objects. 3 objects are absolutely necessary in order to adequately test the multiple objects aspects of the algorithms, while the other values are chosen quite low. This would mean that the numbers of scenarios that need to be tested are

$$C = \frac{(50*5*5)}{3!(50*5*5-3)!} = 324740000$$

There are 604 800 seconds in a week so under the assumption that it is possible to test 10 scenarios per second, this would still require the test bench to run non stop for over a year.

Obviously the number of tests needs to be lowered substantially in order for the test method to be applicable. This can be done by defining strict requirements for how a scenario should be composed in order to be test worthy. One could then create filters that all scenarios must pass and eliminate the scenarios that do not meet the requirements of the filters. Two rules that the scenarios had to fulfil in order to be tested were specified.

- All scenarios must be possible to recreate in real life, with the same result.
- All objects in the scenario must be a threat to the decision maker.

Based on the intentionally vague rules several filters that put then into effect could be created. The filters were also created with input arguments that specify how hard the rules where to be enforced.

### 4.4 Harmless States Elimination

When generating the object states, all combinations of position, velocity and acceleration will be created, but many of the object states will not cause the object to threaten the host.
For instance, there will be objects that start far away from the car, and have a direction and a velocity that moves the object away from the car. These states are not of interest and should be removed. This is done before the test bench starts generating scenarios. The filter simulates the movement of the host and the object for each state for two seconds. If the object does not get within a certain distance of the host, the state is considered harmless and is removed from the set of object states. The distance that the object must pass within is an input argument to the filter. Running the filter on the set of object states will ensure that all objects at some point will pass the host at a distance close enough to be considered threatening.

4.5 Scenario Creation

A scenario is a possible event that the CMbB algorithms can be faced with. In the test domain a scenario is therefore represented as a set of objects or more accurately, the object states that represent objects. To create a scenario it is only to choose the object states that should be included in the scenario.

The aim of the testing is to do such an exhaustive test as possible. Since the scenarios were limited to containing no more than three objects the objective was to test all possible combinations with 3 object states. When conducting the search however, it was actually not necessary to perform the test on each scenario, many test results could be derived from already tested scenarios. Also, many tests would not be useful or even worse,
misrepresenting for the test. One example of misrepresentation is situations where several objects share the same starting position, even though they have different object states. In a real traffic situation, this would mean that an object suddenly would seem to divide into two or more separate objects. Such scenarios should be removed even before testing begins, in order to save time and allow for more interesting scenarios to be tested.

4.6 Eliminate Harmless Scenarios

Several measures are taken to eliminate scenarios that will not be of interest for the evaluation of the algorithms. The previous mentioned filter is one example of this, but this filter only looks at the object states. Several other measures not to generate erroneous scenarios can also be taken. With the object states the filtering only considers the relation between an object and the host. When scenarios are filtered the complexity grows in many ways. Primarily the vast number of scenarios presents problems. In a large test there is a possibility that a hundred million scenarios can be generated, but only some percents are found interesting. Therefore scenarios are filtered as soon as they are generated. This allows for much memory to be saved.

There is also a growing complexity of the type of behaviours that need to be located. In a scenario there might be three objects and the relations between the objects must be considered as well as their relation to the host. For this reason several filters are needed.

4.6.1 Symmetric Scenario Elimination

Since all four algorithms are symmetric only about half of the scenarios need to be tested in order to cover the defined search area. This can be done by dividing the object states into three sets. The first set contains all object states on the left side of the search area. The second set contains all object states that are placed on the centre line, straight in front of the host. The third set contains all object states on the right side of the search area. These three disjoint sets will be called Left, Center and Right. When the search area is divided in an even number of angles Center will be empty, and exactly half of the scenarios need to be tested.
4.6 Eliminate Harmless Scenarios

Dividing the object states into these sets gives a lot of control when constructing the scenarios. For scenarios with one object for example, it is only necessary to test the object states in the sets \textit{Left} and \textit{Center}. All scenarios in \textit{Left} will automatically give the results for the object states in \textit{Right}, so testing these will not result in any new information. This is because of the symmetric properties of all the algorithms. Similar rules on how to manage the sets can easily be defined to create scenarios with two or three objects.

\subsection*{4.6.2 Permutation Elimination}

One of the most powerful ways of lowering the number of scenarios is to eliminate permutations at an early stage. An efficient rule for this is to assign all object states unique identifiers and to assert that all sets is generated with the elements in ascending order, and that no element occurs more than once in each scenario. To create the k-combinations of a set is a rather simple task, and in most programming languages one can find functions that does this for you. These functions could not be used though, since the object states was divided into the three separate sets \textit{Left}, \textit{Right} and \textit{Center}, in order to avoid symmetries. The solution was to introduce a relation between the sets \textit{Left} and \textit{Right}, and then write a new k-combination function.

\textit{Figure 4.4}

\textit{In this scenario the rightmost object could be moved to the outlined position without any principle influence on the decision makers.}
Method

The sets will contain the same number of elements and each element in \textit{Left} will be symmetric to an element in \textit{Right}. When the elements in the two sets are sorted it is possible to deduce much information of the element by looking at its position in the set. It was therefore decided that element \textit{i} in \textit{Left} would always be symmetric with element \textit{i} in \textit{Right}. The sets would also be sorted so that the elements would start at the biggest possible width and the lowest possible length. They where then primarily to increase in length and when no position was left to decrease in width. How the states are sorted can be seen in Figure 4.5.

4.6.3 Elimination of Scenarios where Objects Collide

In a previous example the possibility that two or three objects can start at the same position, even if they have different object states, was discussed. This can happen since the objects can be assigned different velocities or accelerations, even though they share the same starting position. The result of this would be that an object would appear to divide into several objects. This is not a realistic traffic situation, so these scenarios are not to be included in testing.

This is not the only way scenarios can become unrealistic, it can happen in several other situations as well. One example of this is when two objects have different starting positions, but the positions are close enough for the CMbB algorithm to see them as one object. There is also a chance that objects start far from each other, but the velocities and accelerations cause the objects to collide at a later stage.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4.5.png}
\caption{This figure shows the placement of the starting positions. Position 1 – 6 belongs to set Right, position 7 – 9 Center and 10 – 15 Left.}
\end{figure}
4.6 ELIMINATE HARMLESS SCENARIOS

Since the simulation assumes constant acceleration or constant velocity, the collision will be a non-event\(^1\), and the objects will continue as if nothing happened. These are also unwanted scenarios, since they do not occur in real life traffic situations.

The scenarios mentioned above can not be avoided by clever rules on how to combine the object states. Instead, as each scenario is generated, it is also simulated in order to find unrealistic behaviour. For a scenario to be appended to the set of test cases the objects in the scenario may not start too close to each other and they may also not pass each other too close. The distances are input arguments in order to allow for different degrees of enforcement of the filter.

4.6.4 Elimination of Scenarios with Scattered Objects

In opposite to collisions between objects the objects can also be too scattered. If two of the objects in a scenario never get close to each other, they will never interact in a way that causes the algorithms to alarm. These scenarios will therefore not be of interest and can be removed in the same manner as with scenarios with colliding objects.

4.6.5 Elimination of Scenarios with a Corridor

The scenarios have one basic goal, to produce a threat to CMbB decision makers. No matter how the decision makers work, any decent implementation should find the escape route to continue without any

\(^1\) Since the dynamical model does not model collisions the objects will continue with constant acceleration or velocity.
changes when it exists. If it is possible to proceed without accelerating, decelerating or veering, then this must certainly be optimal. In testing such a scenario would be useless, since no decision maker should be threatened by it. To eliminate these tests all scenarios where no object ever crosses the host's path are excluded.

4.6.6 Elimination of Scenarios with Invisible Objects

The basis of all tracking will be data from radar, laser or cameras. These will only be able to track the closest object in each direction. This means that an object positioned behind another object will not be seen by the input sensors, and the scenarios can therefore be removed from the set of test scenarios. This test is only run on the starting position, scenarios where one object start at a visible position but then pass behind another object will not be removed. This is done since this movement would only be simulated in the decision maker and not in the tracking algorithm.

4.7 Iterative Forwarding of Alarms

All algorithms have the characteristic that if a set of object states causes an alarm, all supersets will also alarm. This can be used to great advantage by testing in an iterative manner. Starting with only one object in the path of the CMbB host, it is possible to transfer an alarm to all scenarios with two objects where one object by itself caused an alarm. This procedure can be done every time another object is added. The result is that test results can be derived without ever running the test, this provides an ability to cut the number of performed tests without loosing any test coverage.

The iteration is done in the following manner. In the first step there is only one object. From the set of all possible object states, a set of all interesting scenarios with one object are generated and tested. This generates four sets of results, one for each algorithm. In the second step the set of all interesting scenarios with two objects are generated. Four result sets are also created, and all results from scenarios with one object are transferred to the scenarios with two objects. This means that for all alarms in an algorithm's test result set for one object, all supersets of this alarm is marked in the set of results for two objects. When the actual testing starts, the result set is checked, since the result may already be there. If so, the result has been derived from previous tests, and the scenario can be skipped.
In this scenario one object is invisible to the host. It does not go without saying that such scenarios should be removed however, a good tracking algorithm that model the movement of objects might provide the decision maker with such a scenario.
This chapter presents the reader with an overview of the test bench developed according to the ideas presented in earlier chapters. It gives a general overlook of the test bench architecture but also make a more detailed description of the functions and data structures developed.

This chapter will not give a full-blown requirement specification, nor design or implementation document. It is to be viewed as an overview of the tools developed for the testing and evaluation of the CMbB decision makers.

5.1 Design Aspects

Design is usually necessary when implementing software, so also for the test bench. Below follow some key aspects that were taken into consideration when designing the test bench.
5.1.1 Programming Language

The prototype algorithms are developed in Matlab. It was not an option to port\(^1\) the algorithms to any other language since this would break compatibility with other parts of the CMbB system. Porting the algorithms would also be time consuming and could introduce errors in the prototypes. However, porting the algorithms could probably be beneficial by choosing a language that would save valuable CPU cycles. Since it was decided at such an early stage that porting would not be made, no evaluation in time efficiency between different languages was conducted.

For simplicity it was decided that the test bench would be to written in Matlab too. The test bench itself would not put the same kind of demands on the CPU as the algorithms so there was no need for a faster platform. The memory resources could however be a bigger resource issue when testing lots of scenarios, but Matlab has no problem handle big matrices. Matlab also provides lots of functionality that probably would prove useful. Among others are Matlab’s built in tools for creating GUI's, the great ability to visualize large amounts of data and the many functions for managing and processing matrices.

5.1.2 Repeatability

The CMbB algorithms are under development. When an algorithm is modified it is obliviousy interesting to rerun a test in order to analyze the changes. It is also important to be able to trace the origins of test results in order to be able to draw accurate conclusions. For those reasons much key data must be stored on file. Following a test not only the test results will be saved, but the input as well. This is necessary for users to not only evaluate the behavior but to analyze the cause.

5.1.3 Batching and Distribution

The primary goal with testing was to be able to test 15 – 20 scenarios per second, round the clock for a week. This would result in the testing of about 10 million scenarios. However, the actual test coverage would be much greater then that. For example, almost each tested scenario would have a symmetric scenario that would not be tested and the filters would remove numerous scenarios that were doomed to be harmless. The derivability of some scenarios would also produce scenario results without ever actually

---

\(^1\) Porting is the process of translating and implementing code in one programming language to another.
running the scenario, which would also speed up the final number of tests that would be covered during a test.

It would also be beneficial to speed up the testing by distributing the testing on several computers and it was therefore decided that the test bench should be able to support batching. Batching means that the test bench rather than producing one large test of millions of scenarios divides these into several small tests, each containing just a fraction of the scenarios. These could then be distributed to several computers. The ability to run tests in parallel on several computers does not only serve to speed up the testing, it also provides the testing with robustness. If one computer would go down during the testing one would only lose the batches distributed to that computer. Also, by dividing the tests into smaller parts allow for tests not to require memory resources for unnecessary long time, and the risk that errors will occur is minimised.

An advantage when using Matlab is that the tests can be executed on a lot of different platforms. If Matlab is installed on the clients used, the testing routines can be run as they are. On clients where Matlab is not installed there is a possibility to compile Matlab code into binaries which could then be executed. As an example, a large test could be made by running the test bench in batching mode on a Windows client with Matlab. The batches could then be compiled into binary executables that could be distributed and scheduled to be run on UNIX stations.

When testing is complete all batches can be reassembled. The output data will be in the form of Matlab files so that the data can be processed and analysed easily, on any computer with Matlab.

5.1.4 Test Creation
The way tests are created needed to be both flexible and easy. It should be possible to specify large tests with a lot of possible positions, velocities and accelerations as well as smaller very specific tests. It should also be possible to add noise to the different variables. The solution was to use Matlab m-files as input files. This makes it possible to write code that assigns the variables complex values or just uses a simple constant since the Matlab interpreter is used to evaluate all assignments.

5.2 Architecture
The foundation of the test bench is a quite simple architecture. The idea and the flow are simple and straightforward: create objects, combine these
and finally test them. This description is of course an oversimplification but as a foundation from where the final architecture can be described it is well suited. Based on this architecture, as depicted in Figure 5.1 it is possible to follow the flow a little closer. From a file of input data the possible object states are generated. For this the input file must specify all the arguments needed to create the search area, the positions, possible velocities and accelerations. The result of this is a set of object states that are stored. These are then fed to a scenario generator which produces all scenarios. The scenarios are made by choosing a number of object states and again, these are stored. All scenarios are finally tested and the results are stored in large alarm matrices.

During the time complexity analysis it was found that the number of scenarios grows large very fast and many methods for reducing the number of test cases needed to be added in order to make the test method feasible. As these methods were added the primary architecture evolved and many modules were added.

### 5.2.1 Generate Object States

The testing process starts in the *Generate object states* block. This block takes the test specification and creates all possible combinations of starting positions, velocities and accelerations. It then stores these in a matrix so that the object states in later stages only need to be referenced by an index rather then by all the data in the state.

But there is more to this block. In order to avoid testing unnecessary scenarios the object states also need to be classified and sorted into the three classes *Left*, *Center* and *Right*. When doing this the matrix also becomes sorted according to the ideas described in 4.6.1. Even though the states belong to different classes they are all stored in the same matrix, this is essential since the index number of object states must be unique.

The *Generate object states* block also tries to minimize the number of possible object states by running a filter on all object states with the goal of removing all object states that for sure will not pose a threat to the decision makers. The result of the filtering is that several objects states will be removed from the matrix, leaving lesser states that can be combined when scenarios are created.
5.2.2 Generate Scenarios

The block *Generate scenarios* is in many ways the brain of the test bench. This is where scenarios are created when the block produces combinations of object states. This is one of the most complicated parts, and it is heavily dependant on the work made by *Generate object states* where all object states was sorted and classed into *Left*, *Center* and *Right* in the object state matrix.

This is the block where all unrealistic and seemingly harmless scenarios are eliminated. In the previous block all states were created and then they were filtered post production. In this block scenarios in turn are not processed in the same way. As a scenario is created it is instead fed to the filter by itself and if it passes all tests it is appended to the output matrix of scenarios. This is because scenarios appear in such large numbers and by doing it in this manner much memory can be saved.

*Figure 5.1*

*When simplified the architecture of the test bench can be seen as three modules, each producing their own output matrices.*
This is also where the iterative testing manner is the most active. The iterations mean that the number of scenarios is varied, and this lead to that different rules for combinations apply for each case. As an example, with only one object in each scenario all that the block has to do is to make each object state in Left and Center a scenario. The object states in Right can be omitted since they are symmetric to the once in Left and will already be tested.

When testing two objects however, combinations will occur, so one must give special attention on how scenarios are allowed to be created. Since there are three sets and two objects there are $3 \times 2 = 6$ possible combinations. But again, all must not be tested because of symmetry. One must be careful at this point, though. It is easy to make the error of thinking only half of the combinations need to be tested. This is not true, the combination of choosing two objects from the set Center gives a scenario that is symmetric to itself. Furthermore, when the combination is one object from Left and one object from Right some scenarios are their own symmetric reflection, but not all. The work of making sure that only half of the symmetric cases and all of the non symmetric cases are generated is done in the choosing of objects, so no unnecessary scenario will be generated and no filter are needed to search for such. At the end, four out of the sex combinations need to be generated, but not all scenarios in each combination get generated. The size of the set Center will generally also be much smaller than the others since it only contains objects right in front of the host, resulting in fewer possible combinations.

The goal of the iterative testing is to be able to derive test results as a scenario was generated. In the first step of iteration there has been no previous testing so no results can be derived. In the second iteration step each scenario will contain two objects. But it is possible that one of the objects by itself caused an alarm in the first iteration step and if so the scenario will cause an alarm again. The result matrix of the previous iteration is therefore controlled and if an alarm is present for any of the object states the alarm is forwarded. Since alarms are forwarded for each iteration step it is also possible to track during which iteration step the alarm first occurred for a scenario. This result in a very useful part of information, even if there are three objects in a scenario it might be that only one or two of the objects was active in causing the alarm.

---

1 When the width resolution of positions are even no position will be centred and the set will even be empty.
5.2 Architecture

5.2.3 Test Scenarios

Test scenarios are the simplest block of the three. It simply runs all decision maker algorithms and saves the results in alarm matrices. It creates a new alarm matrix for each iteration, so each step can be analysed post testing. There is one feature that needs mentioning though. When the iterative testing tries to derive results from previous tests, it needs the whole alarm matrix from the previous iteration. Therefore iterative step two cannot be distributed to several computers, since the result matrix must be intact when scenarios for the third step are generated. This means that the test bench can only be run on one computer up to iterative step three. If the user wants to distribute the test it is at this point possible to exclude the Test scenarios block in which case all scenarios with three objects are created and all alarms are forwarded to the last alarm matrix. It is then possible to distribute the testing and to reassemble the results manually.

The end result is that Figure 5.1 does not correctly represent the test bench, mostly due to the iterative testing. A more correct, but also to some extent a more complex representation is the one found in Figure 5.3.

![Figure 5.2](image)

The figure shows four possible scenarios. A and B are scenarios with two objects in Right and two objects in Left respectively. It is clear that each scenario of the first type will have a symmetric case of the second type. C and D are both scenarios with one object from set Left and one from set Right. Even though both are of same type they are symmetric to each other.
5.3 Implementation

In this section a small walkthrough of the test bench will be made. This is done so that readers will be better prepared to follow the testing and evaluation process. The structures used for input and output will be presented together with the larger functions for testing and for analysing.

5.3.1 Input

A lot of variables can be changed in order to create a test suitable to the tester’s needs. In order to make them easier to handle, these variables have been collected and categorized in structures. The structures and their variables are described below.

host_param

This structure contains all necessary data about the CMbB host. It is used to place the host in the scenarios and also contains the size of the host.

Figure 5.3

*With the iterative testing a loop is created between the Generate scenarios and Test scenarios blocks. With each rotation the number of objects in a scenario is increased and a new test case matrix and alarm matrix is generated.*
The structure contains the fields $x_0$ and width. $x_0$ is vector containing the position and speed of the host vehicle\(^1\). Width is the width of the CMbB host vehicle but it also doubles as the obstacle objects widths.

**obj_state_param**
The fields in this structure dictate the possible initial object states. This structure is further divided into three substructures called **pos**, **vel** and **acc** and dictates the possible positions, velocities and accelerations. Each one of these substructures contains a common set of fields:

- **vector**: This is the most important field since it contains the possible values that the respective variable can be assigned. It is in reality a two row matrix where every column forms a $(x, y)$ or $(r, \theta)$ couple, depending on which coordinate system that is used (see below).

- **coord_sys**: This variable can be set to **cart** for a Cartesian coordinate system, or **polar** for a polar coordinate system.

- **remove_extra_zero_radii**: This variable was added to solve a problem that arises when combining vectors. Combining vectors is a smooth way to specify many different values. For example, a test should use speeds of 0, 10 or 20 m/s in four different directions. These can be combined into twelve different velocities, but four of them have a speed of zero. In most cases these three possible values can be removed without making a difference.

- **random**: This Boolean variable should be set to use random values. If random values are used, the possible values in **vector** are interpreted as the means of the variable.

- **distribution**: The distribution to use with random values is specified with this field. Possible values are **uniform** and **Gaussian**.

- **rand_param**: If it is decided to use random values, then this matrix should be provided to specify either the variables span if uniform distribution is used, or the variable’s variance if Gaussian distribution is used. **rand_param** should have the same size as **vector**, thus every value in **vector** has its own random parameter.

\(^1\) No acceleration is needed in the case of the host vehicle. For the objects the acceleration is used to model their movement. But the host’s acceleration does not affect the algorithms since it is assumed that they can change the acceleration instantaneously.
• **num_repeats**: This variable tells how many times the randomization process should be repeated to create more values. For example, if `num_repeats` is 20 and the number of possible positions specified by `vector` is 10, the test bench will use $20 \times 10$ different positions when generating the object states.

The subfield `acc` has some extra fields:

• **coord_sys_reference**: The possible acceleration vectors can be given in to different ways. The normal way is to state the accelerations in the same coordinate system as the velocity. The other way is to state the acceleration vectors in a coordinate system local to the host, where x-axis points in the direction of the host’s movement. This is useful when doing tests where the objects should for example only brake but moves in different directions. In a global coordinate system this would mean that for every possible velocity vectors, an acceleration vector pointing in the opposite direction would be needed. `coord_sys_reference` can have the values `global` and `host`.

• **remove_undefined_direction_states** A problem with using acceleration vectors relative to the velocity vectors arise when the zero vector is a possible velocity. This would mean that the direction of the host in undefined and thus also the acceleration vector would be undefined. The test bench can be told the skip these cases by setting this Boolean variably, e.g., all objects with zero velocity will become stationary.

**filter_param**
This structure is used to set all the filters in the test bench. The structure contains all variables that affect which object states and scenarios that will be filtered out of testing. The fields are:

• **harmless_dist**. It is used when harmless objects are filtered out of the object state matrix. The filtering process is described in chapter 4.4. If an object, when simulating its movement with the constant acceleration model, never reach within `harmless_dist` of the host vehicle, it is determined to be harmless. All such object states are removed before any test cases are generated.

• **overlap_dist**. This variable is used in the filtering of overlapping objects. This is described in chapter 4.6.3. If two objects start at positions within this distance of each other they are considered to be overlapping and any scenario with these two objects will be removed.
This will remove not only scenarios where objects share a starting
position but also scenarios where objects are to close to each other.

- **collide_dist.** This variable is used in the filtering of colliding
  objects. This is also described in chapter 4.6.3. If the distance between
two objects anytime is less than `collide_dist`, the objects are
determined to be colliding. This is partially because of the constant
acceleration model in the algorithms. These can not model the objects
movement in a collision so there is no use in testing such scenarios.

- **min_pass_dist.** In chapter 4.6.4 the filtering of scattered objects are
described. `min_pass_dist` is the minimum distance that at least two
of the objects need to pass each other during a constant acceleration
simulation in order to be added to the test case matrix.

- **passage_width.** If it is possible to drive straight ahead without
colliding, the scenario is not interesting. This is described in chapter
4.6.5. `passage_width` is the width of the corridor right in front of the
host vehicle that an obstacle object must enter in order to make the
scenario interesting. The distance to the centerline is measured from
the center of the objects, this means that a width twice as high as the
width of the host vehicle will result in the narrowest possible corridor
provided that the objects have the same width as the host.

- **cover_width.** This variable is used when scenarios with invisible
  objects are located. The filter is described in chapter 4.6.6.
  `cover_width` is used to calculate the "shadow" cast by an object.
  Generally `cover_width` will be equal to the object width but it must
  not. It could be slightly less than the object width if the tracking
  algorithm would be able to model objects for a number of simulation
  steps after the visual sensors lost contact with the object. The filter
can also be turned of by setting the variable to zero.

### 5.3.2 Output
At the end of a test all results are saved to a Matlab .mat file. This file can
later be loaded into Matlab no matter the platform. The file includes many
matrices needed for the analysis of the algorithms’ behavior.

**obj_state**
This matrix contains all possible object states. Each state consists of a
position, velocity and acceleration as described in chapter 0. A state is
uniquely identified by its index in the matrix.
test_case_1, test_case_2 and test_case_3
These matrices contain the test cases. Each column represent one test case and consists of one, two or three objects states, depending on how many objects the test case use. test_case_1 consist of all scenarios with one object, test_case_2 with two objects and test_case_3 with three objects. For the object states only the identifier and not the complete state is saved in order to save memory.

alarm_1, alarm_2 and alarm_3
These matrices contain the results from the testing. All matrices have four rows, one for each algorithm. The number of columns is equal to the number of test cases. In each position the result of the test is stored as an integer. If the algorithm found an escape route a zero is stored. If instead an alarm is triggered, then the lowest number of objects to cause the alarm is saved.

param_filename
This is name of the test specification file that was used.

5.3.3 Testing Functions
In this section the functions that were created and that are executed during the testing are described.

start_test
This is the main function which runs the whole testing procedure: generating object states, generating test cases and finally executing the tests. The user can specify a test specification file as input. If the user chooses not to, start_test will open a file browser dialog to let the user specify the file.

It is possible to run start_test in a mode that will skip the final three object test, this is used when preparing a batch run and is specified by setting the argument prepare_batch.

The output file will be stored at the same location as the test specification file. It will also have the same name as the specification file, except a .mat-extension and that the date and time of the test completion appended to the filename to avoid overwriting old results.

retest
retest work very much as start_test, the only difference is that retest uses test cases from another test instead of generating them from scratch. As in start_test, the file to retest can optionally be specified as argument or browsed in a file browser dialog.
**combine_vectors**
In the specification files is often useful to combine two sets of values to one set with every possible combination. For example, when specifying a lattice of possible object positions, it is tedious work to write down all different positions. Instead `combine_vectors` can be used to generate every point from one vector with radii and one vector with angles.

**generate_obj_states**
This function generates all possible object states as stated in the test specification file.

**generate_test_cases**
The generation of test cases is a tricky process. Special care must be taken to not generate permutations of test cases, nor should the test cases be symmetric. Permutations are easily avoided by nestling for-loops where the inner loop start at the outer loop's index. Avoiding symmetries can be done by limiting the number of object on each side. For instance, if all scenarios were generated with three objects on the right side and zero objects on the left, or two objects on the right side and one object on the left, no symmetric scenarios would be generated. When combining these methods the goal is achieved, but it is hard to make it a general method for any number of objects. Since it was decided not to test with more than three objects, `generate_test_cases` was not written to cope with more than that.

It is in `generate_test_case` that all the filtering of test cases takes place. Immediately after a test case has been generated it is checked if it fulfills the requirements. This is necessary since it would take to much space to first generate all test cases and afterwards filter out the unwanted test cases.

To generate the test cases the object states is of course needed, but also a struct with the boundaries of the object state sets `Left`, `Center` and `Right`. Furthermore the number of objects for the test case and filter parameters are needed.

**forward_alarms**
Forwarding of alarms refers to the process of deriving some of the results of tests with \( n \) objects from the alarms of tests with \( n-1 \) objects. This can be done since all algorithms for sure will alarm if objects are added to test cases where an algorithm already alarms, see also 4.7.

The complexity of this process grows quickly with the number of objects. To forward alarms, one must either from the smaller test find all supersets in bigger test, or from the bigger test find all subsets in the smaller test.
Example 1. In a test with \( n \) objects states, there was an alarm for the two
object test \( \{1, 2\} \). This alarm affects all three object test cases \( \{1, 2, x\} \)
where \( x \) is any integer from 3 to \( n \). Thus, only this test case affects up to \( n-2 \) test cases with three objects, which must be searched for in the three
object test case matrix. If there is \( m \) three object test, the result will be
\( m*(n-2) \) comparisons to find all affected test cases for every two object
alarm. In general with \( k \) object and \( j \) alarms to forward this makes the
complexity \( O(j*m*(n-k)) \).

Example 2. In the same test there is a test case \( \{1, 2, 3\} \). The two object
test cases that might affect this are \( \{1, 2\} \), \( \{1, 3\} \) and \( \{2, 3\} \). Thus for every
three object test case, one need to search the two object test cases three
times. In the general case with \( p \) two object test cases the complexity
becomes \( O(m*C(k,k-1)*p) \).

Since \( k \) never is bigger than 3 in tests used, the second method is much
faster and thus the method used in the test bench. It is further optimized by
using a lookup matrix instead of searching the two object test cases, e.g., a
\( n*n \) matrix is created where a position \( (i,j) \) represent the test case \( \{i,j\} \).
This is method is however only practical up to three or maybe four objects
since each new object add another dimension to the lookup matrix, and thus
the memory demands quickly becomes huge.

run_test
The process of testing is very simple. The testing routing iterates through
the fed scenarios, looks up the states of each object in the \texttt{obj\_state} matrix
and executes each algorithm with the data. If an algorithm alarms, the
number of objects that were used is stored in the alarm matrix. This gives
the possibility to analyze only test cases where for instance all three objects
participated in causing the alarm, not only just two of the objects.

remove_harmless
This is the filter that removes object states that never passes the host object
within a certain distance described in 4.4. \texttt{remove\_harmless} takes the
trajectories from simulating each object state with the constant acceleration
model, and returns those that pass the host vehicle within a certain radius.
This radius is called \texttt{harmless\_dist} and is part of the struct \texttt{filter\_param}
which is also fed to the function.

---

\(^1\) Not all of the supersets have to exist, some of them might have been trapped in a filter when
generating the test cases, hence \textit{up to}. 
**5.3 Implementation**

**simul_car**

*simul_car* is used by many functions to simulate the movement of objects according to the constant acceleration model described in 3.4. Input is one or more object states, the number of steps to simulate and the time difference between each step.

**is_close**

*is_close* takes vectors with one or more positions for two or more objects and a maximum distance as arguments. If the distance between two vectors in any corresponding position for any two objects is less than the maximum allowed distance, *is_close* will return true. Many filters use this function to determine if objects ever get close to each other or the host.

**is_covered**

*is_covered* tests whether any object is hidden behind another object. It does this by first sorting the objects according to their distance from the host, then it checks if any of the other objects is covered by the first one. If none is it continues with the second closest object and checks if any the objects further away is hidden behind it. This procedure continues until a hidden object is found or the object furthest away is reached which of course can not hide any object. *is_covered* take the position of any number of objects as input.

**prepare_batch_run**

This function is used when the tester wants to divide a test into several pieces that can for example be distributed to many computers. It takes the name of a test specification file or the name of an already tested file. Then it tests or retests with one and two object since it is only the final three object test that is batch tested. *start_test* and *retest* can be run in a mode where the final test is skipped by calling them with the parameter *prepare_batch*, this is used by *prepare_batch_run*. Then the matrices test_case_3 and alarm_3 are partitioned into the number of specified of chunks which can then be distributed to several clients.

**merge_batch**

After a batch run is completed it is necessary to merge the batch files together to be able to analyze the results, this is done with *merge_batch*. *merge_batch* takes the name of a not fully tested file that was created by *prepare_batch_run*. This name is used to locate the files belonging to the batch which should reside in the same directory. Then the batch is merged together and appended to the original output file.
5.3.4 Statistical Tools

When a test run is completed the tester is left with matrices of thousands of test cases. The results do not speak for themselves so several tools were developed in order to gather statistics from the results.

**alarm_stats**

*alarm_stats* counts occurrences of each alarm combination for the algorithms. For instance, how many test cases were found where MOCC and MOHC alarmed but SOCC and MOES did not? In the example table below one can see that in this test, there were 35 cases like that and in 16 of those there were three objects involved.

**Table 5.1. alarm_stats output example.** These numbers comes from small test with 2,091 test cases. The test uses 45 positions, 9 velocities and 2 accelerations.

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No algorithm alarms</td>
<td>962</td>
<td>1,919</td>
<td>2,058</td>
<td>814</td>
</tr>
<tr>
<td>Only MOCC alarms</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Only MOHC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Only MOES alarms</td>
<td>0</td>
<td>76</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>Only SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOCC &amp; MOHC alarms</td>
<td>0</td>
<td>22</td>
<td>16</td>
<td>35</td>
</tr>
<tr>
<td>MOCC &amp; MOES alarms</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MOCC &amp; SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOHC &amp; MOES alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOHC &amp; SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOES &amp; SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOCC, MOHC &amp; MOES</td>
<td>0</td>
<td>74</td>
<td>4</td>
<td>81</td>
</tr>
<tr>
<td>MOCC, MOHC &amp; SOCC</td>
<td>121</td>
<td>0</td>
<td>0</td>
<td>70</td>
</tr>
<tr>
<td>MOCC, MOES &amp; SOCC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MOHC, MOES &amp; SOCC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>All algorithms alarms</td>
<td>1,008</td>
<td>0</td>
<td>0</td>
<td>1,059</td>
</tr>
</tbody>
</table>

The output contains four columns; the three first shows the number of alarms where a certain amount of objects were active, the fourth shows the number of alarms where any number of objects caused an alarm. These numbers can sometimes be confusing; for example, what happened to the 76
cases where only MOES alarmed with two objects? In most of those test cases the other algorithms alarmed with one or three objects which makes the total only 19.

This information shows how efficient the algorithms are in finding escape routes. It also proved useful to find errors that made the algorithms’ behavior diverge from the expected. For example, an error made SOCC alarm for more than one object, which should not be possible.

**alarm_stats_pairs**
As mentioned above, `alarm_stats` could sometimes produce confusing results. `alarm_stats_pairs` could then be a better tool to use since it compares algorithms one by one. `alarm_stats_pairs` takes an alarm matrix from a test with any number of algorithms and run `alarm_stats` on each possible combination of those. The output will be $C(n, 2)$, where $n$ is the number of algorithms compared, different tables which looks similar to the output from `alarm_stats`, but with only two algorithms per table.

**run_test_sim**
To get a value on the time difference between the algorithms alarms it is necessary to simulate a scenario for a certain amount of time, and apply the algorithms at each time step until all algorithms alarms, this is done by `run_test_sim`. `run_test_sim` takes a test_case matrix, a obj_state matrix, a host_param structure and a sim_param structure as arguments.

When choosing test cases by first running an ordinary test to find interesting test cases there is a problem. In the interesting scenarios found there are usually some algorithms that alarms, but there is no information about the history which means that the alarming algorithm could alarm at an earlier stage in the given scenario. This will lead to inaccurate alarm time estimations. To prevent this there is an optional parameter called rewind_steps. This parameter will make `run_test_sim` perform a backwards simulation before starting the real simulation which makes it possible to find the first point in time when the algorithm starts alarming.

**plot_trajectory**
`plot_trajectory` plots one more object trajectories. A trajectory consists of a series of positions. This function is mainly used by `plot_test_case` (see below) but can also be used to study the affect on area coverage when filtering object states like in the example below. An example where `plot_trajectory` has been used can be seen in Figure 5.4.

**plot_test_case**
This is small function that is helpful when plotting the trajectories of test cases. As input it takes one or more objects states, how many steps the states should be simulated and finally the time difference between each step.
It simulates the movement of the objects according to the constant control model and finally plots the trajectories of the objects with the `plot_trajectory` function.

**obj_state_stats**
This function counts how many of the possible object states are that actively caused an algorithm to alarm. Actively in this case meaning that if any of the objects were removed from the test case, the algorithm would not alarm. This tool is used to see in what stage of the iterative testing that the object states are used. Such information could be useful when specifying a new test, a high rate of object states that are active with only one object indicates that many objects are placed to close to the host vehicle.

**Figure 5.4**

*This example shows the outcome of the remove harmless filter. Grey trajectories belong to objects states that never reached the host vehicle close enough. Black trajectories belong to kept object states. The white line is the path of the host vehicle.*
In order to be able to study the suggested escape paths, the algorithms had to be slightly modified to return the paths they found. But since the algorithms work in such different ways, the returned path uses different formats. escape_path_tool is hard coded to deal with these differences which makes it necessary to rewrite the program if new algorithms should be tested.
escape_path_tool has proven to be a very useful tool when working with the algorithms. Without it, it would have been very hard to understand what was going on when testing the algorithms.

5.4 Verification

A verification of the most important parts of the test bench has been done to insure that the test result won't be affected by errors in the test bench. Particularly it is very hard to manually verify that the test case generator does not produce too many test cases, e.g. symmetric cases or permutations, or too few test cases.

**Figure 5.6**

Screenshot of the escape path tool. In this example, the suggested escape paths of MOES and SOCC can be seen.
To verify that there did not exist too many test cases, a special set containing all the symmetric cases and permutations were generated for the test cases. The original set of test cases was then checked against this new set of test cases. If there were a match, the test case generator produced faulty test cases since that would mean that there were at least two symmetric test cases in the original set.

To verify that all wanted test cases were generated, a set with all possible combinations of the object states, e.g. a set including all symmetric cases, were generated. Then it was checked that for all of these test cases there existed one test case in the original set that was a symmetric case, a permutation or an identical one.

The other parts of the test bench were verified with handcrafted test cases or by using a debugger to trace the code while executing.
6 Test Execution

This chapter covers the test executed during the thesis. It discusses how the decision on the number of scenarios to be included was made, how the objects were placed and the velocities chosen. It also discusses how the influence of some of the known weaknesses in the algorithms was minimized.

6.1 Pre Test Evaluation of the Decision Makers

In order to increase test efficiently the decision maker algorithms needed some pre-test editing. For this reason several small evaluations were made, both in means of code inspection as well as by running the test bench with a small amount of scenarios. It was used to give some early evaluations of the decision makers so that bugs, inefficient code and output could be corrected before exhaustive testing was started.

6.1.1 Choosing an Object

As described in 3.9.1 SOCC specify one object as the most threatening and then ignores all other objects when searching for an escape route. In the initial version of SOCC the assigning of the most threatening object was done by calculating the TTC for each object that was in the path of the host. The object with the lowest TTC was then chosen. There is an obvious logic to that choice, the object you are closest to time wise must be avoided the earliest.
However, during small tests with the test bench it showed that the TTC was not always suitable to go by. The test gave numerous examples of scenarios where objects with the lowest TTC easily could be avoided but the object with the second lowest, or even the third lowest, TTC could not be avoided at all. An example of these scenarios is depicted in Figure 6.1. The solution to this was to make sure that every object is avoidable before looking at the TTC. If any object was unavoidable then for sure it could be considered the most threatening\textsuperscript{1} and SOCC must return an alarm.

### 6.1.2 Stop Heuristic Simulation

During the evaluation of the filters of the test bench some indications was given that MOHC behaved oddly. A small number of the scenarios that got caught in the filters because they contained harmless objects caused MOHC to alarm. An analysis of the scenarios showed that it indeed was MOHC and not the filters that were erroneous.

The heuristic behaviors of MOHC specified that the host primarily would brake if it was possible and otherwise steer away from objects. If no escape route existed then full braking would be applied. This is also described in section 3.9.3. The problem was that this behavior was simulated throughout the two second time frame, no matter if an escape route was found or not. This meant that an escape route could be found at an early stage and as the simulation continued, leading the host along the escape route the heuristic behavior could suddenly change, causing the host to brake. As the host decelerated it stayed in an exposed area longer than necessary and collided. The solution to this was to stop the simulation when an escape route was found, since this is enough information for the algorithm to base its decision to not alarm.

\textsuperscript{1} Unavoidable is atomic, there are no degrees of unavoidability, so any of the unavoidable object can be chosen if several objects are.
By stopping the simulation as soon as an escape route was found or a collision occurred the time complexity of the algorithm could be lowered considerably as well. In a benchmark run to get an idea of the speedup, the new algorithm was almost eight times faster than the old version. This was very welcome since MOHC stood for a substantial part of the time consumption in tests, even though MOES theoretically should have been much worse.

### 6.1.3 Optimize Search Pattern

MOES, to some extent, suffered from the same disease as MOHC. Several escape paths was calculated even though only one was needed. In the initial version of MOES all branches of the search tree was calculated before the

---

**Figure 6.1**
The figure shows two scenarios with the initial version of SOCC. In the scenario to the left the lack of an escape route indicates that the object cannot be avoided and a collision is imminent. In the scenario to the right another object has been added and suddenly there is an escape route, leading straight into the original, unavoidable, object. This behaviour in SOCC was removed by checking for unavoidable objects.
search for escape routes was commenced. This was extremely tedious since the tree grew exponentially.

In order to make the decision maker faster it should be implemented so that it searches the tree as the tree is calculated. It should also search the tree depth first, so that a complete manoeuvre is calculated as soon as possible. This makes it possible to make a decision without having to calculate all branches. In the best case only one complete manoeuvre needs to be calculated. It is also possible to affect the probability that the best case will occur. It is optimal to calculate and search the tree starting with the most likely escape route and end with the least likely escape route. When the test bench generates scenarios the elimination of symmetries will cause far more objects to be placed to the right of the host than on the left. MOES was therefore implemented to start by trying to turn full left. The result was that MOES over a large number of scenarios clearly executed faster. A test that before the change finished in 874 s was completed in 640 s, a 27 % drop.

6.1.4 Return Escape Route
During the testing of the decision makers the main data is whether or not an alarm is triggered for the scenarios. But there are some interesting characteristics that can be useful beside alarms. One of the most interesting is the escape routes. The escape route is the reason that a decision maker does not alarm, and it is vital that the decision maker makes the decision for the right reasons. For that reason all algorithms were edited so that they also returned the chosen escape route when such existed.

6.1.5 Object States
An object state contains the position, velocity and acceleration for an object. It is supposed to contain all data needed by the decision maker in order to produce a decision, but the decision must fulfil the requirements set on the system. These get reflected onto the object states as well, if the states do not provide the required data then the decision maker will not be able to produce a sound decision.

Given the dynamical model of the decision makers position, velocity and acceleration are required in the states, so they do not contain any redundant information. As stated in chapter 3.5 there is some information that might be lacking though. The size and direction of the object is crucial information in order to be able to produce satisfactory decisions, and should therefore be added to the states.
This would however not only require some large changes to the decision makers, but it might also make the testing less effective.

Adding information to the states mean that more combinations can be created and the number of scenarios can be generated grows immensely. During an exhaustive test this is of course unwanted, adding width and length of objects will cost in the lower number of possible positions, velocities and accelerations in the test.

It is not certain that the sizes actually present any new challenges during the exhaustive test though. For the multiple objects decision makers the most significant factor is not the states of the objects by themselves, but how the objects relate to each other. A scenario with a very wide object might very well be interpreted just the same as a scenario where a regular size object is travelling perpendicular to the host’s direction. Since the aim is to conduct an exhaustive test, there is very likely that each variation of size is covered by one of the original scenarios.

Furthermore, in a common traffic environment objects generally fit into one of five different groups shown in Table 6.1. The two groups of stationary objects can not be simulated with the test bench since all possible state combinations will be created, meaning that such objects would be destined to be assigned velocities in many scenarios even though they should not. This will be discussed more in chapter 8.4 where we evaluate the test bench. The group of objects at 0.5 m and the group of 2 m are so similar that they
easily could be seen as one. This leaves only two groups that could be of use to vary in the exhaustive test.

Given the small number of variations needed and the fact that the decision makers are likely to be confronted with similar scenarios anyway, it was decided that the object states were to be left as they were. From a testing point of view it seemed that the width and length did not need to be added to the states which in turn would benefit the number of positions, velocities and accelerations that could be generated for the test. It also meant that the decision makers could be left as they were.

**Table 6.1. Classification of types of objects in a typical traffic environment and the size of objects in these classes.**

<table>
<thead>
<tr>
<th>Size</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0.2 m</td>
<td>Stationary objects such as road signs and trees</td>
</tr>
<tr>
<td>0.5 m</td>
<td>Pedestrians, bikes and motorcycles</td>
</tr>
<tr>
<td>2 m</td>
<td>Cars, trucks and busses travelling parallel to the host</td>
</tr>
<tr>
<td>4 — 20 m</td>
<td>Cars, trucks and busses travelling perpendicular to the host</td>
</tr>
<tr>
<td>Infinite</td>
<td>Stationary objects such as fences, dividers railings and buildings</td>
</tr>
</tbody>
</table>

**6.1.6 Miscellaneous**
Beside the big changes described in the previous sections there were some other changes made. Some bugs that caused the decision makers to crash were found and corrected. Some small changes were also made to make the code execute faster but overall, these were small changes and fixes that are not of the magnitude that they need to be presented within this thesis.

**6.2 Minimise Known Differences**
The four decision makers have several differences that were known before the big test. Since a large part of the thesis was focused on the analysis of the results it was important that the differentiating results did not only reflect the already known differences. Some parts of the test bench and the
algorithms therefore got some extra attention, in order to make the scenarios and the algorithms to work as equivalent as possible.

6.2.1 Object Position and Dimensions
As mentioned in section 3.5 it is vital to make sure that all decision makers interpret the scenarios in the same way. This is not fulfilled with the four algorithms in this thesis though. SOCC and MOCC currently just take the position in the object state estimate and extend it to a line 1.8 meter wide. In other words they can be seen as only calculating and trying to avoid the front bumper of a car. MOES on the other hand also calculate for the length of a car, simulating that it is 4 meters long. MOHC is a hybrid of the two, it uses the method of SOCC and MOCC when searching for escape routes, but uses the method of MOES when it simulates the movement of the host and the objects.

To make the test results more consistent it was decided that the assumed object length in MOES and MOHC would be changed from 4 meters to only 0.2 meters. This would make the interpretations roughly the same in all four algorithms. It would clearly have been beneficial to go the other way, to make the objects 4 meters in all the algorithms. This would have made the scenarios more realistic. But SOCC and MOCC were ignoring the length of objects. To remedy that would not only demand editing of the code but changes to the train of thought for the algorithms. Since the primary aim was to evaluate the existing algorithms it was decided that these changes fell outside the scope of the thesis and the needed changes would be left as suggestions for improvements.

6.2.2 Tunneling
In the filters as well as in some of the decision makers a phenomenon called tunnelling can occur. Tunnelling is a possible consequence when a time-continuous event is simulated in a time-discrete environment. With CMbB systems a tunnelling occur when, in a simulation, two objects pass through each other without detecting the collision since it occurs between two simulation steps. An example of tunnelling is shown in Figure 6.3.

There are several ways of reducing or eliminating the frequency of tunnelling occurrences. The most obvious one is to increase the frequency of the simulation steps but this result in a linear worsening of the time consumption. Another way is to increase the size of objects. The larger the objects are the more time is needed between each simulation step for a tunnelling to occur.
The problem with this method is that size may be fixed. One can certainly not increase the length of a car, so this could not be used either. A third possible method is to restrict the relative speed between objects, since the slower objects move the more time they need to pass through each other. But there is a problem with that as well, tunnelling can occur over infinitely small parts of objects, so the relative speed might be limited extremely for tunnelling to become unusual.

At the end no single solution was found, but all the methods were used to some extent. The simulation frequency was set to 10 Hz. This is a number that very easily could have been varied, but raising it would have come with the cost of heavy time consumption during the testing.

Figure 6.3
Tunnelling occur when objects pass through each other in between simulation steps. In the two examples above A is in motion while B is stationary. It is clear that A passes through B but since they does not share any space in a simulation step it will not be recognised as a collision.
The length of objects in MOHC and MOES was set to 0.2 and not lower, which would have been more useful for interpretation purposes as in 6.2.1. The length of the host was though kept at 4 meters. Even though it would have been beneficial to lower that number in order to make the conditions for the decision makers more equal, just like with the objects, it was decided that it should be kept at 4 meters. If it were to be lowered MOES would simply tunnel in almost every escape route and produce faulty data.

The primary plan was to allow for the host and the objects to be able to meet at relative speeds up to 50 m/s but this number was lowered to 35 m/s. In a head-on collision where the objects are 0.2 meters and the host are 4 meters there is a distance of 4.2 meters that need to be covered in a simulation step for a tunnelling to occur. With a relative speed of 50m/s and a simulation frequency of 10 Hz about 16% of the head-on collisions are likely to produce a tunnelling but at 35 m/s it should be non existent. The host may very well decelerate or even accelerate before the collision so tunnelling can still occur even at 35 m/s but they should be rare. For collisions that are not head-on the percentage will obviously be much higher since the distance needed to be covered during a simulation step is much lower.

6.3 Creating the Test

When running an exhaustive test with the test bench there are infinite variations of different tests that can be run. The variation of input arguments can produce widely differing tests and the user must choose them wisely in order to be able to interpret the results accurately. In this section a small description of the method used to find suitable arguments will be made, and the chosen arguments will also be accounted for.

6.3.1 Defining a Search Area

For the large testing in this thesis it was chosen that a circular search area would be used. Many lasers and radars on the market have a field of view (FOV) of about 20°, so this angle was used in testing as well. They also have a range of over 150 meters, but this is more then needed. Given the maximum relative velocity between the host and an object and the timeframe used by the decision makers it is easy to calculate the maximum distance at which objects can pose a threat to the host. For the exhaustive test a maximum distance of 100 meters was chosen.
It was also decided that the objects starting positions would be placed with a logarithmical distancing since this would produce more threatening objects. In this case more threatening meaning both that the level will be higher and that the number will be higher.

6.3.2 Setting the Filter Parameters

An important part of testing is the validation of the test methods and the test results. Just as with any form of information gathering one must always analyse the trustworthiness and the relevance of the information. From the testers point of view this means that testing by itself is not enough, one must also show that the results indeed reflect the status of the tested system or process.

For the testing in this thesis, there is a responsibility to show that the test cases generated do bear relevance. But the opposite must be shown just as well, it must be shown that the test cases omitted do not have any relevance. This means that it must be shown that the filters do not remove interesting scenarios.

For this reason it was decided that a small test would be made where all filtering was turned off. It would then be possible to evaluate the filters by running them on the scenarios and looking at the decision makers’ results on the scenarios caught. These results could then be used to resolve how firm the filters could be without removing too much interesting scenarios.

It would also be useful in order to get a good image of how many scenarios that were eliminated by filters and to assess the how many scenarios that needed to be tested given the input arguments. This could later be used to give a good estimate of what test parameters should be used in order to get a good test size.

All parameters were evaluated with a test using 105 positions and 3 velocities which led to 1,199,081 test cases with three objects. The filter variables are described in chapter 5.3.1.

- **harmless_dist**
  
  Figure 6.4 show the relation between the percentage of test cases that was kept and the value of harmless_dist. It clearly shows that there is a huge gain in filtering the object states, even if the variable is set high. It seems that harmless_dist could be set as low as 8 meters, which would remove 91 % of the scenarios while keeping 88 % of the alarms. There was no need to be extremely harsh in the exhaustive test so it was set to 12 meters where it removed about 85 % of the scenarios while keeping 97 % of the alarms.
• **min_pass_dist**
  Figure 6.5 show the relation between the percentage of test cases that were kept and the value of `min_pass_dist`. Is shows that at 15 meters close to 100% of the alarms are kept while over 15% of the alarms are removed. It was decided that 20 meters where to be used for the exhaustive test where about 12% of the test cases where removed.

• **passage_width**
  Since the host as well as all objects are 1.8 meters wide the it is a fact no collisions should occur when `passage_width` is equal to or higher than 3.6 meters. In Figure 6.6 the result of the evaluation if shown. Interestingly, this simple filter can remove about 25% of the test cases.

• **overlap_dist**
  The filter where this variable is active is one that is suppose to remove unrealistic scenarios. There is no real need to evaluate these in term of alarms versus test cases removed since the result of unrealistic scenarios is valueless. The variable was set to 2 meters so that it would allow cars to stand alongside each other without the scenario being removed.

• **collide_dist**
  The filter of this variable works much like the previous. It removes unrealistic scenarios. The variable was set to 2 meters for the same reasons as for `overlap_dist`.

• **cover_width**
  This also belong to a filter that removes unrealistic scenarios. It was natural to set it to the object width, 1.8 meters.

These tests showed that the filters indeed were able to remove a large amount of scenarios. The evaluation is somewhat misleading though. Since only one filter is active at each test correlations will not show in the test and there might very well show that one filter is covered by another.
Figure 6.4
Analysis of the number of alarms removed and test cases removed for different values on remove_harmless.

Figure 6.5
Analysis of the number of alarms removed and test cases removed for different values on pass_distance_filter.
6.3 Creating the Test

To make an exhaustive test there are some restrictions that must be made. Since an infinite number of object states can be generated it is a prerequisite that there is something that limits this number. The test bench has been created with versatility in mind, and it does not put any real restraint on the number of object states. The area where the test bench will complain first is probably when it runs out of memory for the output matrixes.

The goal with the exhaustive test was to test as many scenarios as possible with the condition that the testing may not take longer than a week. In order to get an estimate of the number of object states that would generate such a test some runs was made with a low number of object states. The data needed was the number of scenarios generated given the maximum possible number of scenarios, the number of scenarios that actually got tested and the number of tests that was made per hour. The small tests showed that 80,000 – 100,000 test cases per hour could be tested. Out of all the possible test cases left after the filtering was 1.9 %. Another 20 % of the tests were possible to derive from the tests with one or two objects. From these data it was possible to predict that a test with about 2,000 possible object states would generate a test that would take about a week to finish.
on two computers. However, this calculation does only include the time
needed for the three object test. With a big test like this, the time needed to
generate all test cases becomes substantial because of the filtering and alarm
forwarding. To compensate for this and to add some safety margin, it was
decided that about 1,500 object states would be enough.

6.4 Test Distribution
The primary plan for test execution was to distribute the test to a large
number of Unix/Solaris stations. With the aid of Volvo Information
Technology the tests would then be scheduled to execute during non-office
hours, since other employees might use the stations during office hours. But
as the small pre-test evaluations was made it became clear that the two
PC/Windows stations used for the development and implementation
provided the same amount of computational power, but were available 24
hours a day. At that point efforts were made to increase the number of
Unix/Solaris stations but unfortunately no more stations were available.
The decision was therefore made that the test would be distributed to the
two PC stations.

6.5 Running the Test
The test was executed in two parts. The first part consisted of the
generation and testing of scenarios with one and two objects. It also
included the generation of scenarios with three objects. This part was done
using one pc-station. The first part cannot be batched since the iterative
step from one to two objects requires a complete result matrix in order to be
efficient. This step took approximately 44 hours to execute.

The second step of testing was to transfer half of the test cases to a
secondary pc-station, to test all three object test cases and to reassemble the
test results on the primary pc-station. This step took approximately 74
hours to execute. The distribution and collection of tests took less than one
hour, leaving the process of testing each scenario responsible for just about
all of the time consumption. Since the tests were executed on two computers
concurrently the total computational time adds up to about 193 hours.

For the test an input file was created. In this file all vital parameters was
assigned their decided values. The input file can be found in Appendix A.
The meaning and use of all parameters are described in 5.3.1.
With the parameters the objects is built from 84 starting positions, placed along 7 uniformly spaced degrees along $\theta$ and 12 logarithmically spaced lines along $r$. There are 9 different velocities and 2 different accelerations available to each object. This makes it possible to generate 1512 different object states.

The test run resulted in 435 scenarios with one object, 77,642 scenarios with two objects and 12,265,925 scenarios with three objects. It is clear that the number of scenarios grows very large when the number of objects in each scenario grows. In order to test scenarios with four objects it is necessary either to reduce the number of object states or to increase the computational power. It is possible to increase the computational power either by using computers with faster CPU’s or by distributing the test to more computers.

The numbers above give some indication to how the test bench has worked. There are 1,512 different objects but the matrix of scenarios with one object does only contain 435 scenarios. This means that 1,077 objects have been filtered out during the first iterative step. In the first step the removal of harmless states is the most efficient. It removes 750 of the 1,512 possible objects, and these are removed permanently and are excluded from step two and three as well. While the gain in the first step might not seem that great, in step two and three the benefit becomes clearer. In step two it removes 852,375 out of 1,142,316 scenarios, and in step three 501,514, 000 out of 574,965,720 scenarios are removed from testing.

It is also possible to see that the other filters are working well. In each step they remove 315, 212,299 and 61,185,795 scenarios respectively. With the harmless objects removed in iterative step, this is out of 762, 289,941, 73,451,720 scenarios correspondingly.

By studying the result matrices it is also possible to analyse the gain from the iterative testing. In the result matrix for scenarios with two objects 42,468 out of 310,568 test results\(^1\) where able to derive the results from the result matrix of step one. In step three the numbers where 9,312,759 out of 49,063,700.

When using the 1,512 possible objects it is required to make 2,304,438,192 tests in order to make an exhaustive test of all scenarios with one, two and three objects. The test bench conducted 40,020,781 number of tests, and were with these tests able to produce 49,376,008 test results. The

\[1\] Note that the number of test results equals the number of test cases times the number of algorithms
2,304,438,192-49,376,008 test results that were not produced are either uninteresting or likely to show no alarms, so the total number of 2,304,438,192 results can be considered to be covered by the test bench.

An interesting point is the fact that of the 49,376,008 test results that were produced, as many as 39,944,794 were tests where the decision maker did not alarm. Of the 12,265,925 scenarios with three objects that were tested as many as 9,591,156 scenarios were scenarios where no decision maker alarmed. Of those remaining 2,674,769 scenarios where decision makers alarmed, only 43,350 scenarios made a decision maker alarm because of three active objects.
7 Evaluation of CMBB Algorithms

This chapter discusses the evaluation of the decision makers. It covers the general properties of the decision makers and point out the differences of the solutions. It discusses the reasons and consequences of these and gives suggestions of improvements.

During the evaluation focus was put on two issues. It was important to know how the different decision makers behaved in relation to each other. It was also of interest to find the scenarios that made the decision makers act especially bad or incorrect.

7.1 Movement Model

As previously stated, the algorithms use a constant acceleration movement model. The acceleration is said to be constant under the assumption that the object will continue the manoeuvre they are currently using.

The model uses the variables position, velocity and acceleration. Each of those variables consists of two components specified in a global Cartesian coordinate system. Since the acceleration is constant in this global system, it will not be constant in an object's coordinate system if the object turns. The
object will still accelerate in the same global direction, making the object move along a hyperbolic line instead of along a circular line, which is usually expected if the object is a car executing a constant manoeuvre. This is true for the host as well as for the objects. In order to make the cars move more genuine, a function that transfers the control manoeuvres into the coordinate system of the object could be implemented. The different acceleration vectors and simulated path are illustrated in Figure 7.1.

Another weakness in the model is shown when objects have a low velocity and accelerates in the opposite direction of the velocity. When the object decelerates to a full stop, the constant acceleration model will assume that the objects will keep the acceleration, which causes the object to start backing. It is unlikely that a driver will change gear into reverse and start backing immediately after braking to a full stop. A function that detects these situations and resets the acceleration and velocity to zero when this occurs could also benefit the algorithms.

7.2 Calculation of Escape Routes

There exist some irregularities in the way the escape routes are calculated. These irregularities are not inherited from the dynamical model, since the dynamical model only simulates the movement of objects. Rather, it is derived from the declaration of the possible maneuvers of the host.

It is assumed that an acceleration of 9 m/s² can be applied to the host in each simulation step. This acceleration may have any direction, no matter the current velocity or the acceleration in the previous step. In real life cars cannot behave this way though. Acceleration and velocity have some degree of correlation, at high speeds it is not possible to steer as sharply as in low speeds.

Much work has been put into developing efficient dynamical models for automobiles. Such models are used for ABS, traction control and yaw rate control systems. The most popular model by far is the bicycle model. It is used to such extent to model vehicle handling that it become close to standard. It approximates a car with a bicycle by seeing the fixed back tires as one tire and the two front steering tires as one. It models both longitudinal and lateral accelerations on the tires as well for the center of mass which means that it also is able to model the rotation of the vehicle. More about the bicycle model can be read in [3] and [11].
7.3 Positioning and Extension of Objects

In chapter 3.5 the importance of the algorithms handling the unknown extension of object uniformly was stressed. This is however not fulfilled due to the way the algorithms detect a collision. SOCC and MOCC, who only uses one manoeuvre, calculate the accelerations that are needed to avoid the objects. If the accelerations exceed the highest available acceleration there will be a collision. But the extension of the object is not taken into consideration when doing this calculation, which in some cases might make the calculated escape route go straight through the side of the very object it tries to avoid. An example of this is shown in Figure 7.2.

**Figure 7.1**
Comparison of the acceleration vectors and the movement with global coordinates and coordinates in the vehicles coordinate system.
For MOES and MOHC the situation is different. These algorithms change manoeuvres and thus need another way to detect collisions. They simulate the escape route and in each step look for an overlap between the host’s extension and the extension of the objects. If there is one, the escape route led to a collision and can be discarded. The algorithm will look for another escape route until one without collisions is found or all possible escape routes have been tried.

The changes made during the pre test evaluation proved to eliminate the differences between the decision makers. No clear case where the decision makers interpreted the scenario different was found during the exhaustive test, but this was much expected since the change in object length from 4 meters to 0.2 meters made the scenarios all but identical for all decision makers.

But as discussed in chapter 6.2.1 these changes worked against the realism in the testing. The problems within the decision makers was clearly too large to fix during the pre test evaluation stage. But after the exhaustive test it became clear that improvements were needed. The object states should be extended with all data that might be needed. The required data are the width, length and direction of the object. This is data that MOES easily can be edited to make use of. For SOCC, MOCC and MOHC some larger changes are required.

The problem is that they only try to avoid the front bumper of the cars it is presented with. This means that only two of the four corners of the object are taken into consideration. With the help of a reasonable amount of calculations all four corners can be considered. By repeating all calculations for the rear bumper all corners will be accounted for and logical functions
can determine which accelerations is needed to avoid the whole object. To improve the decision maker further the choice should not be the two bumpers, but the two diagonals of each object. This make the two lines representing the object cross, and this ensures that the host cannot pass between the two lines. Another, less effective but yet interesting, choice is to put the lines from mid-point to mid-point on opposing sides of the object. With this representation the object still have width and length but it is possible to cut the corners of the object. This is interesting in testing purposes since it makes the three decision makers behave much like MOES which can tunnel corners of objects. The different representations can be seen in Figure 7.3.

The reason that length, width and direction need to be added to the object state is so that the corners can be located. For the testing of this thesis it was sufficient to make a uniform size assignment to all objects and to use the velocity vector as a direction approximation. But there is no sure way to predict how the testing and generation of test objects change as the decision makers evolve, so the variables should probably be added to the object states so they are made complete as soon as possible.

7.4 Tunnelling

The tunnelling problem was described in chapter 6.2.2. It is a problem that only occurs in MOHC and MOES. For MOHC tunnelling is a quite small problem. There are 19 simulation steps in which a tunnelling can occur, and in these 19 steps there is likely to be no more than 5 steps where the objects are close enough to each other for a tunnelling to occur. This comes from the fact that the objects must have a large relative difference in velocity and/or directional vectors to enable a tunnelling. The movement of the host is also, unlike MOES, uncorrelated with the presence of tunnelling which makes a tunnelling a pure mishap and very unlikely. During the analysis of the exhaustive test no scenario was found where MOHC incorrectly outperformed MOCC due to the tunnelling phenomena.

MOES on the other hand suffers to a great extent of tunnelling. During the exhaustive test the exhaustive search could in worst case result in 36 900 simulation steps in which a tunnelling could occur. And a scenario in where all other decision makers alarm is likely to cause large tree search from MOES in order to find an escape route.
MOES big disadvantage is that the easiest way to avoid an object is to tunnel it so if a scenario demands a large tree to be searched then it becomes more likely that a tunnelling will occur and that that route will also be an escape route and therefore chosen. In some sense MOES can be said to want to fail, since it favours escape routes with tunnelling in them. There are several improvements that can be made in order to reduce the number of escape routes with tunnelling in them. Some of these are discussed in chapter 6.2.2 as well. In the end the decision maker must handle real objects so to set requirements on the size or relative speed of objects are not a valid solution. This leaves only the increase of the simulation step frequency as a long term solution. But increasing the simulation step frequency could make an already bad time complexity deteriorate heavily so it can not be done by simply raising the frequency across the board. A possible way to reduce the occurrence of tunnelling is to refine the tree search so that when an escape route is found this route is checked for tunnelling using a higher frequency.

The tunnelling detection function will obviously be bad from a time complexity perspective, but not nearly as bad as a general raising of the simulation step frequency. The reason for this is that it would not be applied to all branches of the search tree but only on the branches of the found escape route. All routes tested up to that point have caused a collision so to search for tunnelling in those is redundant.

An interesting aspect of the tunnelling phenomena is also the amount of work one should put into remove it. Since it can be considered non existent in MOHC only MOES is affected by tunnelling. But the possible escape manoeuvres that can be found by MOES is very limited by the number of

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7_3}
\caption{The leftmost object show the current representation that objects get in SOCC, MOCC and MOHC. The other four show alternative ways of representing the objects.}
\end{figure}
manoeuvres that the driver is assumed to be able to take. When tunnelling is removed to that degree that corners of considerable size can be tunnelled but no more, it might be sufficient to stop there. A tunnelling of a corner can be avoided by adjusting the escape route slightly so that the object is missed completely, but MOES is not able to make small manoeuvres. A human driver on the other hand is able to make far more manoeuvres than MOES ever will model, so a human driver would possibly find the escape route. So it seems that tunnelling at some point actually might improve MOES rather than being a setback.

7.5 Collision Detection

Another problem with the same origin as tunnelling is collision detection. Since MOHC and MOES simulate the movement of the host, it is possible that there is a collision in each simulation step. Time analysis of the testing showed that the collision detection functions used to search for these were very time consuming, and this was mostly because of the use of the Matlab function \texttt{inpolygon}. The collision detection was improved somewhat during the pre test evaluation, but not to the extent that the problem became secondary. It would clearly be useful to eliminate the use of \texttt{inpolygon}, and in order to do this an alternative method for detecting collisions all together might need to be created.

7.6 Testing Results

Below are the results from the testing presented. All results will be commented in 7.7.

7.6.1 Alarm Counts

In the tables below are the alarm counts from the big test presented. All algorithms are compared one by one to make it easier to make conclusions from the data. The columns named one object, two objects and three objects show how many alarms that were caused because of these exact number of objects. This is the reason why MOHC in Table 7.1 alarms in 648 cases for three objects where MOCC does not, despite that it in the total column shows that MOCC always alarm when MOHC does. In those 648 test cases MOCC alarmed because a fewer number of objects.
### Table 7.1. MOCC vs. MOHC

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>10,116,907</td>
<td>11,970,317</td>
<td>12,231,465</td>
<td>9,787,487</td>
</tr>
<tr>
<td>MOCC alarms</td>
<td>0</td>
<td>18,701</td>
<td>8,261</td>
<td>26314</td>
</tr>
<tr>
<td>MOHC alarms</td>
<td>0</td>
<td>0</td>
<td>648</td>
<td>0</td>
</tr>
<tr>
<td>Both alarms</td>
<td>2,149,018</td>
<td>276,907</td>
<td>25,551</td>
<td>2,452,124</td>
</tr>
</tbody>
</table>

### Table 7.2. MOCC vs. MOCS

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>9,905,973</td>
<td>11,919,243</td>
<td>12,223,050</td>
<td>9,591,156</td>
</tr>
<tr>
<td>MOCC alarms</td>
<td>254,799</td>
<td>158,645</td>
<td>29,696</td>
<td>368,400</td>
</tr>
<tr>
<td>MOCS alarms</td>
<td>210,934</td>
<td>51,074</td>
<td>9,063</td>
<td>196,331</td>
</tr>
<tr>
<td>Both alarms</td>
<td>1,894,219</td>
<td>136,963</td>
<td>4,116</td>
<td>2,110,038</td>
</tr>
</tbody>
</table>

### Table 7.3. MOCC vs. SOCC

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>10,116,907</td>
<td>11,970,317</td>
<td>12,232,113</td>
<td>9,787,487</td>
</tr>
<tr>
<td>MOCC alarms</td>
<td>0</td>
<td>295,608</td>
<td>33,812</td>
<td>329,420</td>
</tr>
<tr>
<td>SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Both alarms</td>
<td>2,149,018</td>
<td>0</td>
<td>0</td>
<td>2,149,018</td>
</tr>
</tbody>
</table>

### Table 7.4. MOHC vs. MOES

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>9,905,973</td>
<td>11,935,274</td>
<td>12,230,201</td>
<td>9,613,685</td>
</tr>
<tr>
<td>MOHC alarms</td>
<td>254,799</td>
<td>142,614</td>
<td>22,545</td>
<td>345,871</td>
</tr>
<tr>
<td>MOES alarms</td>
<td>210,934</td>
<td>53,744</td>
<td>9,525</td>
<td>200,116</td>
</tr>
<tr>
<td>Both alarms</td>
<td>1,894,219</td>
<td>134,293</td>
<td>3,654</td>
<td>2,106,253</td>
</tr>
</tbody>
</table>

### Table 7.5. MOHC vs. SOCC

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>10,116,907</td>
<td>11,989,018</td>
<td>12,239,726</td>
<td>9,813,801</td>
</tr>
<tr>
<td>MOHC alarms</td>
<td>0</td>
<td>276,907</td>
<td>26,199</td>
<td>303,106</td>
</tr>
<tr>
<td>SOCC alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Both alarms</td>
<td>2,149,018</td>
<td>0</td>
<td>0</td>
<td>2,149,018</td>
</tr>
</tbody>
</table>
7.6 Testing Results

Table 7.6. MOES vs. SOCC

<table>
<thead>
<tr>
<th></th>
<th>One object</th>
<th>Two objects</th>
<th>Three objects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No alarms</td>
<td>9,905,973</td>
<td>12,077,888</td>
<td>12,252,746</td>
<td>9,730,243</td>
</tr>
<tr>
<td>MOES alarms</td>
<td>210,934</td>
<td>188,037</td>
<td>13,179</td>
<td>386,664</td>
</tr>
<tr>
<td>SOCC alarms</td>
<td>254,799</td>
<td>0</td>
<td>0</td>
<td>229,313</td>
</tr>
<tr>
<td>Both alarms</td>
<td>1,894,219</td>
<td>0</td>
<td>0</td>
<td>1,919,705</td>
</tr>
</tbody>
</table>

7.6.2 Alarm Times

To get an understanding for the importance of these differences in alarm rates presented above, scenarios from a smaller test with similar alarm distribution was simulated. The results are presented below.

Table 7.7. Mean times from simulation start to alarm. In these simulations the mean time from simulation start to collision was 1.2683 seconds.

<table>
<thead>
<tr>
<th></th>
<th>MOCC</th>
<th>MOHC</th>
<th>MOES</th>
<th>SOCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>All scenarios</td>
<td>0.5974</td>
<td>0.6030</td>
<td>0.6186</td>
<td>0.6815</td>
</tr>
<tr>
<td>More than one active object</td>
<td>0.3917</td>
<td>0.4553</td>
<td>0.5992</td>
<td>0.8636</td>
</tr>
</tbody>
</table>
7.7 The Algorithms

This chapter will compare the decision makers based on the testing results above.

7.7.1 SOCC

It isn't very much to say about SOCC, it performed exactly as expected. As can be seen in Table 7.7, it alarms about 0.25 to 0.45 seconds later than the multiple object algorithms in situations with multiple objects involved. Given that full braking would lead to a deceleration of 9 m/s\(^2\), this means that the multiple object algorithms could reduce the speed 2.3 m/s to 4.1 m/s before SOCC would alarm.

The number of false alarms is on the other hand non-existing. As can be seen by Table 7.3, Table 7.5 and Table 7.6, only MOES finds escape routes that SOCC does not find. This is caused by two reasons, tunnelling and the
fact that it in motion model used is more efficient to steer and brake simultaneous. These issues are MOES specific and will be discussed in 7.7.4.

### 7.7.2 MOCC

It can be seen in Table 7.1, Table 7.2 and Table 7.3 that MOCC proves to be the algorithm that alarms most often. This indicates, and is confirmed by Figure 7.4, that MOCC is the algorithm that in general alarms earliest. The reason for this is the limitation in the search of escape manoeuvres.

With only one object, MOCC and SOCC are equivalent. But with multiple objects MOCC alarms in 15.3 % more cases than SOCC. As expected there was not any single case where SOCC alarmed and MOCC did not.

### 7.7.3 MOHC

Looking at the total number of alarms, MOCC alarms in 1.1 % more cases than MOHC. More interesting though, is that in multiple object situations MOCC alarms is in about 6.7 % and 29,1 % more cases for two respective three active objects.

Looking closer at these situations it becomes clear that is very often enough to brake just for a very short while before turning. Awaiting a turn often solves the problem since the objects move out of danger or produces a gap possible to make use of. As can be seen in Table 7.1, there exist no test cases with one object in which by initially braking makes it possible to turn out of danger. This means that braking with the intention of afterwards making a steeper turn is not a better solution since MOCC and MOHD in those cases produce the same result.

MOHD uses the same method of collision detection as MOES in the cases where braking is necessary which makes theoretically possible to experience tunnelling with MOHD too. But this means that the host vehicle has been braking all the way to dangerous object, thus reducing the relative speed so much that tunnelling is very unlikely. There was not a single case of tunnelling observed with MOHD in the tests.

Since MOHC in some way is an extended version of MOCC implementation wise, there were no cases where MOHC alarmed and MOCC did not.

### 7.7.4 MOES

In some sense, a MOES algorithm with a large set of manoeuvres and enough resolution in them can be regarded as a reference since it should give
a close to optimal result. But the tested MOES algorithm does not test very many escape routes and thus should not always give the correct result. This however does not explain the quite high number of test cases where MOES alarms but the other algorithms do not. Instead it is the different ways of detecting a collision which becomes a disadvantage to MOES in the majority of these test cases. In chapter 7.3, the extension of objects and their impact on the different collision detection methods was discussed. What was not discussed was the extension of the host vehicle. When the other algorithms calculate an escape route, they only consider the front of the host vehicle, but in a lot of cases that escape route leads to having an object colliding into the side of the vehicle. MOES on the other hand takes the whole car into consideration and thus does not find the escape route that the other algorithms find.

The rather small set of possible manoeuvres also affects the testing results. In some cases this makes it impossible to find ways through narrow gaps, where the other algorithms have no problem calculating a possible manoeuvre that will take the vehicle through. This problem is further aggravated by the differences in the collision detection methods described in the previous paragraph. While the other algorithms only finds an escape route that takes the front of the vehicle through the gap, MOES looks for escape routes that takes the whole car through the gap. This requires a wider gap than if only the front should be cleared.

The tunnelling problem has been thoroughly discussed in this thesis and by studying Table 7.4 one can get an idea of its effects. With three objects involved, the difference is evident. In those test cases MOHC and MOES are almost complementary as it is in a small minority of test cases that they return the same result. In how many of the test cases where MOES differs from the other algorithms because of tunnelling is hard to estimate. Even though it would be theoretically possible to study all those cases, it is possible that there exists another legitimate escape route since MOES aborts the search as soon as one is found.

The ability to change manoeuvre a lot does not seem to be of great importance. In most test cases there will be enough with only one or two manoeuvres. It is however hard to analyse this thoroughly due to the algorithm's current implementation where it always tries going left first. When it would simply be enough with a right turn MOES could use several anyway. For example in cases where a right turn would be sufficient, MOES could initially turn right, then turn left for a while if possible and finally turn right again.

A somewhat surprising advantage compared to the other algorithms seems to be the algorithm's ability to brake and turn simultaneously. This is an
ability that the other algorithms lack, and with the motion model used gives a tighter turn.
This chapter will give a brief evaluation of the test method and test bench. Some statistics showing the performance of the test bench will be shown and discussed. Some alternative ways of generating test cases will also be presented.

8.1 Number of Test Cases
During the duration of the thesis a goal has been to make an exhaustive test. Since all object states originate from continuous domains\(^1\) an infinite number of variations can occur and an exhaustive test can of course not be made. The goal was instead interpreted as trying to test as many, as varying and as threatening scenarios as possible.

When analyzing the test data some doubts sprung whether or not an exhaustive test really was the best suited test method for the four current decision makers. The reasons for this are several but the strongest one is

\(^1\) In real life position, velocity and acceleration can vary with infinitely small increments.
that the algorithms had a lot of differences and faulty behaviors that were known even before the test. When four algorithms were tested with such internal differences the results will contain a very large number of scenarios with errors and differences that was already known. To analyze such data proves very difficult, much effort goes into the analyzing of behavior already known before testing and new errors might not be found since they mistakenly could be categorized as an already known error.

An exhaustive test is really only useful for two things. For one, they can be used to verify the behavior of a decision maker by comparing the results to previous versions. Secondly it can be used to compare several different decision makers but in that case it is important that there are no major known differences, so that the size of the set of interesting scenarios is kept low, and that the elements show new and unknown behaviors.

Another way to use the test bench more efficiently is to iterate testing, starting with small tests and to let the size grow larger as erroneous behaviors grows larger and the algorithms get more trustworthy.

A prerequisite to doing an exhaustive test is the ability to remove large parts of the output data. A very easy way of doing this that has been used is to simply only consider those test cases where the algorithms return different output. A risk with this kind of simplification is that of removing test cases where all algorithms perform badly. Thus, the method can only be used to verify the algorithms’ behavior against each other. A possible solution would be to run the algorithms against an improved MOES with very high resolution in the possible maneuvers but that would not be feasible with millions of tests.

8.2 Filters

In chapter 6.3.2 there is a discussion about filter parameters settings along with some results of how efficient some of the filters are. To get a more complete picture of how many test cases that really are removed, an examination was done on a large number of tests with different number of object states. The result can be seen in Figure 8.1. As can be seen in the figure, less than 0.7 % of all possible test cases are left after filtering. It should be noted though that these tests used a remove harmless parameter value of 5 meters. The big test used 12 meters, which led to about 2 % of the test cases being kept. Another important thing to keep in mind is that these numbers includes the removal of symmetric test cases, which is about half of the test cases.
The forwarding of alarms has also been an important way of reducing the number of tests. Looking at the alarm forwarding numbers in Figure 8.2, they are somewhat linear. As more velocities and accelerations are added the ability to forward alarms drops. This is because tests with a high degree of static object states placed close to host vehicle will lead a lot of alarms in the one object test, which will then be forwarded. As velocities and accelerations are added, the degree of these object states drops and thus also the forwarding rate drops. This also explains the peaks in the beginning of the graph. Since these are the tests with the lowest number of objects and all tests have about the same number of objects states with a position close to the host, the degree of close object states will be higher in these tests.
Filtering of test cases has been essential to achieve a high resolution in the tests. But as always with filters there is the risk of removing too much. When the filters were developed they were designed assuming that the decision makers were functioning decently. As it turned out, however, this was faulty assumption. It was discovered when evaluating the filters that there were test cases which should not impose any problem that made algorithms alarm. Thus, the filters have to be used with caution. If the testing is done to verify a correct behaviour of an algorithm, the filters should be used restrictively. Another way could be to run two tests, one large test with filters and one smaller without.

A possible improvement was thought off in the late work. All decision makers except MOES today make an initial calculation whether any object is in path or not. *In path* means that the object is in such a state that an action is required by either the host or object to avoid a collision. This could of course be taken advantage of by writing a filter that removes such test cases where there are no objects in path. This can be seen as an improvement of the way to eliminate scenarios with a corridor, as described 4.6.5, since the test cases removed by this filter should be subset of the ones removed by the new filter. This can easily be realised by considering that

**Figure 8.2**

*This figure shows how many percent of the tests with three objects that do not need to be tested because the outcome can be foreseen with results from the one and two object tests.*
removing test cases where no manoeuvre is necessary includes removing test cases where no object will ever pass in front of the host. As stated above MOES does not do this initial in path check, the filter should however be applicable in the case of MOES too, as long as going straight ahead is one of the possible manoeuvres.

8.3 Other Methods

Given that the evaluation should be done by evaluating the decision makers’ behavior with test cases, there are some different ways to do this. The first way would be to create the test cases by hand. The advantage of this method is the very high grade of realism that is achieved. Since the number of test cases will be few it is also easy to study the behavior in each test case. The big disadvantage is that important aspects of a decision maker’s behavior could be missed because a particular behavior did not show with the test cases used. Another disadvantage is that it is hard, except in the obvious case with three objects on a line in front of the host vehicle, to create test cases where all objects are active. Of those about twelve million test cases tested in the big test only about 0.35 % had three active objects.

Another popular method is to use random data when evaluating systems, particularly in early stages of development when the need for high quality of test cases is less important. To see how much it differs between generating test cases randomly and the way that the test bench does it, two small tests with the same number of object states were run. The first test used random objects states and was run ten times. The generated test cases made the decision makers alarm in 0 – 27 % of the test cases. The same measure for the regular test was 19 %. This means that a test with random object can in some cases produce more alarms but in most cases do not.

8.4 The Test Bench

The test bench has been a great tool when evaluating the algorithms, particularly the escape path tool was essential when studying the algorithms’ behaviour. There are however some issues that need to be taken into consideration when developing algorithms and using the test bench.

8.4.1 Requirements

In the current version of the test bench there are some requirements that the algorithms need to fulfill. The algorithms needs to e symmetric, e.g., the
algorithms must produce the same result with symmetric scenarios. They also need support alarm forwarding. Even though it would be easy to rewrite the test bench to support algorithms that do not, alarm forwarding not only speed up testing but has also provided another advantage. It also makes it possible to find those really interesting test cases with multiple objects involved, thus it is almost fundamental that the algorithms support this feature.

8.4.2 Shortcomings

In the current version of the test bench it is only possible to use one kind of object, the object can have different position, velocity and acceleration but all share the same width and extension. As described in 6.1.5 there are other types of objects that do not share these properties. This limits the test bench’s capabilities to be used to test particular situations, for example, situations with a rail making it only possible to turn one way. This particular example could be approximated using an object that moves along the host vehicle, hindering it from turning.

When an error is found in an algorithm it is common to run a new test with the corrected algorithm. It would be enough to just test the modified algorithm but this is unfortunately not possible. All algorithms must be tested again. With small tests this is usually fine, but with huge tests this could be several wasted days. The only possible solution is currently to comment out the lines that execute the algorithms that do not need to be tested, but this is tedious work. A possibility to select which algorithms to test with an optional argument would be sufficient.

Another big problem related to the algorithms and which to run is the problem of integrating a completely new algorithm. Many of the tools written to analyze test results are hard coded to run with the four algorithms studied. There are also some differences in the way the algorithms return their found escape maneuver, MOHC and MOES simulate movement step by step and thus return an escape route consisting of several vehicle states. SOCC and MOCC on the other hand only calculate one single acceleration that would clear them from danger and thus only returns this value. This makes it necessary in the escape path tool to treat algorithms different and it would thus complicate the introduction of a new algorithm a bit. A good solution would probably be to make both SOCC and MOCC calculate the trajectory that the calculated acceleration would lead to in the same way as MOHC does in some cases. The return of an escape path could also be made optional, controlled by an input parameter, since the escape path calculation takes extra time and in only interesting when studying the algorithms’ behavior in detail.
For a long time there were plans of trying to write a tool that would do an automatic analyse of the test results and find a small set of test cases that would be representative for the various differences and shortcomings in the algorithms. A possible solution is to simulate the test cases where algorithms differ and then return those cases that have the most extreme differences. The risk is of course that these extreme cases originate from the same differences in the algorithms’ behaviour. What would really be useful is a way of classifying test cases into sets that cause similar behaviours of the algorithms, but this is really non-trivial work.

8.4.3 Usability
The test requires the user to be familiar with Matlab. This should however not be a problem among its future users. The most user-unfriendly part of testing procedure is probably when the user wants to find a subset of test cases where the algorithms perform in a certain way, for example, SOCC and MOCC alarms but not MOHC and MOES. This requires the user to write long logic expressions to find the right test cases from the alarm matrix.
9 CONCLUSIONS

This chapter contains the conclusions drawn from the evaluation of decision makers. It also gives some suggestions on future work on the test bench as well as some suggestions on how the development of the decision makers should proceed.

9.1 Conclusions

The test results clearly show that there are benefits in using a multiple object decision maker. MOCC clearly outperforms SOCC and increases the alarm time so that countermeasures get more time to mitigate the collision. In some cases the decision maker was able to double the alarm time and a by no means unrealistic scenario is that a collision is mitigated to the point when airbags do not need to inflate.

MOCC also shows that multiple object decision making does not need to be that advanced in order to be beneficial. MOHC on the other hand shows that MOCC introduces a new problem, the risk for false alarms. There is an aspect of dualism in this. The multiple objects property is beneficial because it buys time for countermeasures. But the property can also cause false alarms, and to minimize or eliminate these is vital. Minimizing the time gained is done be ensuring that it is utterly impossible to avoid the collision by manoeuvring. The dualism lies within the fact that the multiple objects property is introduced to gain time, but all effort goes into making sure that the time gained is the lowest possible.
By introducing a heuristic control algorithm, even such a basic one as MOHC, the result changes noticeably. About one percent of the alarms of MOCC were improved by MOHC. But even though MOHC can be considered a basic heuristic control algorithm, it is still a very advanced decision maker relative to SOCC. And therein lies the single most important conundrum with the multiple object property. How complex is the decision maker allowed to be? The decision maker needs to be able to reach decisions at a high frequency. The algorithms can be as complex as the developers like, but this in turn puts large requirements on the computational power.

Another issue that this thesis has not discussed, but that is of great importance, is the statistical occurrence of multiple object scenarios in real life traffic situations. It might very well be that the occurrence is so rare that the gain can not justify the extra costs. The CMbB system is very likely to give an alarm once in its lifetime, about a second before the collision in which it is destroyed. To evaluate the true properties of a CMbB system real life data must be collected so that the frequency in which scenarios appear is allowed to correlate with the results.

SOCC is a decision maker that in its simplicity guarantees the non-existence of false alarms. This makes it very safe to use, but it also makes it alarm later than possible if multiple objects are involved.

MOCC is identical to SOCC for one object but as multiple objects are introduced there is a good likelihood that false alarms will occur. To use a constant control manoeuvre to evade a collision then does simply not cover the possible escape routes. MOCC should at most serve as a platform from where a more advanced decision maker can be created.

MOHC also handles the one object very well, and it handles multiple objects in a better fashion than MOCC. There is a lower likelihood of false alarms, but false alarms can still occur. Differentiating it from MOCC is that MOHC is designed in a manner that makes it easy to modify. It is very easy to change the heuristic behaviour without making any principal changes to the decision maker’s framework.

MOES is built upon the idea of an exhaustive search. Initially it seemed to be very powerful but as the tests showed it was very week in two aspects. It allowed a large quantity of tunnelling and it was not able to make small adjustments. It is also very costly in term of computational power. The

1 Given that the world model in a correct way represents the world.
weaknesses are so severe that the decision maker probably can not be evolved into a usable version.

9.2 Future Work

In this section the future work on the decision makers and the test bench is described. This does not include descriptions of the located problems areas and the suggested fixes of these, but rather discusses the next step in development and evaluation of CMbB algorithms.

9.2.1 Test Bench

The test bench was developed to specifically evaluate the four decision makers discussed in this thesis. In order to be useful in the future work of multiple object CMbB, the support of various number of decision makers needs to be implemented. The different ways of returning an escape path is a problem that makes replacing algorithms difficult, this could easily be fixed by introducing a common interface.

A new filter that removes test cases where no object will collide with the host vehicle, given that all parties maintain their current accelerations, was thought of in the late work. Such a filter could be really efficient in removing test cases since this is necessary if the algorithms should even try to find an escape route.

9.2.2 Algorithms

The future development of the decision makers is two parted. First of all many suggestions to changes have been made on how the current implementations can be improved. While they are suggestions, some of them clearly induce such benefits or degrees of realism that they can not be left unnoticed. One of these is the inclusion of all four corners of objects, another necessary change is the addition of variables to the object states.

The other part is to find a way to reduce or eliminate the number of false alarms for multiple object decision making. MOHC clearly is the decision maker that shows the greatest potential in turning into a useful decision maker. The heuristic behaviour needs to be more advanced in order to eliminate the false alarms that still occur, and the test bench is a very powerful tool for comparing different versions of MOHC against each other.
Several methods are available to do the part of choosing the heuristic behaviour. One quite simple idea is to use potential fields to calculate and suggest a control signal in each step. The field of artificial intelligence also applies quite well to this problem. Beside the more direct approaches of decision trees and such, it is also possible to apply machine learning techniques as neural nets, Bayesian nets or decision tree learning in order to find an efficient heuristic behaviour.
References


% Created by:
% Johan Tjernström, Andreas Kivrikis
% Volvo Car Corporation

% --- Vehicle parameters ---
host_param.x0 = [0 0 20 0]';
host_param.width = 1.8;

% --- Object simulation parameters ---
sim_param.t_max = 2;
sim_param.num_steps = 20;
sim_param.dt = sim_param.t_max/sim_param.num_steps;

% --- Object states parameters ---
obj_state_param.pos.coord_sys = 'polar';
obj_state_param.pos.remove_extra_zero_radii = '1';
tmp_r_min = 10;
tmp_r_max = 100;
tmp_num_r = 12;
tmp_r = logspace(log10(tmp_r_min),log10(tmp_r_max),tmp_num_r);
tmp_th_max = 10*pi/180;
tmp_num_th = 7;
tmp_th = linspace(-tmp_th_max,tmp_th_max,tmp_num_th);
obj_state_param.pos.vector = combine_vectors(tmp_r,tmp_th);
obj_state_param.pos.random = 0;

obj_state_param.vel.coord_sys = 'polar'; % or 'cart'
obj_state_param.vel.remove_extra_zero_radii = '1';
tmp_speeds = [0 7.5 15];
tmp_directions = [0 90 180 270]*pi/180;
obj_state_param.vel.vector = combine_vectors(tmp_speeds,tmp_directions);
obj_state_param.vel.random = 0;

obj_state_param.acc.coord_sys = 'cart';
obj_state_param.acc.remove_extra_zero_radii = 0;
obj_state_param.acc.coord_sys_reference = 'host';
obj_state_param.acc.remove_undefined_direction_states = 0;
obj_state_param.acc.vector = [0 0;-9 0]';
obj_state_param.acc.random = 0;

% --- Filter parameters ---
filter_param.harmless_dist = 12;
filter param.overlap_dist = 2;
filter_param.collide_dist = 2;
filter_param.min_pass_dist = 20;
filter_param.passage_width = 2*host_param.width;
filter_param.cover_width = host_param.width;

clear tmp_*
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/