Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Locust System Integration into Demo Vehicles

Examensarbete utfört i reglerteknik

av

Pär Palmebäck

Jonny Wei

LiTH-ISY-EX--07/3967--SE

Linköping 2007

TEKNISKA HÖGSKOLAN
LINKÖPINGS UNIVERSITET

Department of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden

Linköpings tekniska högskola
Institutionen för systemteknik
581 83 Linköping

Master's Thesis

Locust System Integration into Demo Vehicles

Examensarbete utfört i reglerteknik

vid Linköpings tekniska högskola

av

Pär Palmebäck

Jonny Wei

LITH-ISY-EX--07/3967--SE

Supervisor, Volvo Car Corporation: Martti Soininen

Supervisor, Linköping University: Markus Gerdin

Examiner: Fredrik Gustafsson

Linköping 2007-01-14

Avdelning, Institution
Division, Department

Automatic Control
Dept. of Electrical Engineering
Linköping University

Datum
Date
2007-01-14

Språk Language	**Rapporttyp** Report category	**ISBN**
___ Svenska/Swedish	___ Licentiatavhandling	_____
_x_ Engelska/English	_x_ Examensarbete	**ISRN**
	___ C-uppsats	LiTH-ISY-EX--07/3967--SE
	___ D-uppsats	**Serietitel och serienummer/ISSN**
Annat/Other:_____	___ Övrig rapport	_____

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2007/3967/

Titel/Title Locust System Integration into Demo Vehicles

Författare
Author Pär Palmebäck, Jonny Wei

Sammanfattning
Abstract

This thesis project was carried out at Volvo Car Corporation. It is based on an EU project called Locust in which a bio-inspired visual sensor system (the Locust sensor system) for automotive collision avoidance was developed. The Locust sensor system is designed to emulate the collision avoidance functionality of the Locust grasshopper, which is well-known for its extraordinary vision based collision avoidance ability, in particular with regard to its fast reaction times to perceived threats. Volvo Car Corporation is interested in the possibility of using the bio-inspired technology developed in the Locust project to improve its already existing collision avoidance systems. Pedestrian collision avoidance is of high interest, for which the properties of the Locust grasshopper are desirable.

The purpose of this thesis project is to develop two demonstrator vehicles to test the performance of the Locust sensor system, carry out the testing, and evaluate its usability for Volvo Car Corporation. The first vehicle is a scale 1:5 model car that was originally developed in a thesis project at KTH, and the second a full scale Volvo XC90.

It was found in the testing that the Locust sensor system is promising for pedestrian collision avoidance applications. The results for detecting other vehicles were also acceptable, but Volvo Car Corporation already has other collision avoidance systems with better performance in this regard. In general the test results were very good for speeds up to about 40 km/h. This indicates that the Locust sensor system would be most usable in a city driving environment, parking lot situations, and for driving in residential areas.

Nyckelord
Keywords: The Locust Project, CAN, TTCAN, collision avoidance, model car, visual sensor, microcontroller

Abstract

This thesis project was carried out at Volvo Car Corporation. It is based on an EU project called Locust in which a bio-inspired visual sensor system (the Locust sensor system) for automotive collision avoidance was developed. The Locust sensor system is designed to emulate the collision avoidance functionality of the Locust grasshopper, which is well-known for its extraordinary vision based collision avoidance ability, in particular with regard to its fast reaction times to perceived threats. Volvo Car Corporation is interested in the possibility of using the bio-inspired technology developed in the Locust project to improve its already existing collision avoidance systems. Pedestrian collision avoidance is of high interest, for which the properties of the Locust grasshopper are desirable.

The purpose of this thesis project is to develop two demonstrator vehicles to test the performance of the Locust sensor system, carry out the testing, and evaluate its usability for Volvo Car Corporation. The first vehicle is a scale 1:5 model car that was originally developed in a thesis project at KTH, and the second a full scale Volvo XC90.

It was found in the testing that the Locust sensor system is promising for pedestrian collision avoidance applications. The results for detecting other vehicles were also acceptable, but Volvo Car Corporation already has other collision avoidance systems with better performance in this regard. In general the test results were very good for speeds up to about 40 km/h. This indicates that the Locust sensor system would be most usable in a city driving environment, parking lot situations, and for driving in residential areas.

Acknowledgements

We would like to thank Mr Martti Soininen at Volvo Car Corporation for the opportunity and the trust to conduct this master thesis. A special thanks to Mr Roger Johansson at Chalmers University of Technology whom helped us greatly with the model car. Finally but not least, we would like to thank our families for their unconditional support and encouragement throughout this thesis work.

Contents

1 Introduction

1.1 Thesis Project Background

The employer for this thesis project is Volvo Car Corporation in Gothenburg, Sweden. It is based upon two other projects, a scale 1:5 model car developed in a master thesis at KTH in 2004, and a parallel EU-project called Locust that developed a visual sensor system (the Locust sensor system) for collision avoidance by modelling the behaviour of the Locust grasshopper.

Volvo Car Corporation already has several sensors for use in collision avoidance systems, and is interested in new techniques to improve the performance of these systems. An area of particular interest for improvements is pedestrian detection, since their current systems deal primarily with detection of collision threats in the form of vehicles.

Among biologists the collision avoidance ability of the Locust grasshopper is well-known, and Volvo Car Corporation hopes to be able to use the bio-inspired techniques developed in the Locust project to extend their set of sensors and improve the performance of their collision avoidance systems. The ability of the Locust grasshopper to respond quickly to perceived threats is a property that Volvo Car Corporation would like to incorporate in their cars.

1.2 Problem Specification

The objective of this thesis project is to build two demonstrators for the Locust project, one using the model car mentioned above, and one using a full scale car (a Volvo XC90), and evaluate the sensor performance and its usability for Volvo Car Corporation. More specifically the main problems that need to be solved in this thesis project are:

- To develop a communication bridge between the TCCAN network in the model car and the RS-232 serial communication interface of the Locust sensor system.
- To develop a relatively simple collision avoidance algorithm that performs active maneuvers using the brakes and/or steering of the model car.
- To help tune the Locust sensor system for suitable performance for in-traffic driving.
- To build an HMI (Human Machine Interface) to integrate the Locust sensor system with a Volvo XC90. The HMI should only give warning signals with light and sound, no active maneuvers like in the model car case.
- To choose test scenarios for the demonstrators, carry out the testing, and analyze the results.

1.3 Reading Guide

Chapter 2 introduces the Locust sensor system, gives a background to the Locust project and describes the fundamentals of the Locust sensor system concepts.

Chapter 3 describes the network protocols used in the Volvo XC90 and the model car, CAN and TTCAN.

Chapter 4 introduces some basic collision avoidance theory, and suggests an outline for implementation of a collision avoidance system.

Chapter 5 describes the hardware and software of the model car used in this thesis.

Chapter 6 describes the HMI developed in this thesis to integrate the Locust sensor system into a Volvo XC90.

Chapter 7 presents the test cases and results for the tests carried out with the XC90 demonstrator.

Chapter 8 contains conclusions and suggestions for future improvements.

2 Locust Sensor System

2.1 Locust Project Background

The Locust project has been going on since year 2002. The objective of the project is to conduct research about the visual perception system of a grasshopper, called Locust, to develop bio-inspired visual perception systems and implement them on single chips. The reason why the Locust grasshopper was chosen for this study is because of its remarkable ability to detect and avoid collisions. This trait of the Locust grasshopper made it particularly interesting for the automobile industry.
The project was partly funded by the European Union and included several research groups within Europe. These were:

- School of Biology at University of Newcastle upon Tyne, in Great Britain.
- Instituto de Microelectronica de Sevilla, IMSE, in Spain.
- Hungarian Academy of Sciences, Computer and Automation Research Institute MTA SZTAKI, in Hungary.
- Volvo Car Corporation, in Sweden.

The project was divided into four stages, each of them assigned to one research group. The School of Biology at University of Newcastle was responsible for the first stage. The main task during this stage was to study the physical visual system of the Locust grasshopper and then develop mathematical models according to these. IMSE would then, during stage 2, use the models to construct a microchip that simulates the Locust grasshopper's visual system. To test the performance of the chip, electronic hardware as well as software had to be implemented. This task was carried out by MTA SZTAKI in Hungary in stage 3. In stage 4 the Locust system was installed in an automobile vehicle for final testing. This was done at Volvo Car Corporation in Gothenburg Sweden. This thesis project was an important part of stage 4.

2.2 Collision Avoidance of the Locust Grasshopper

The Locust, *Locusta Migratoria*, grasshopper is interesting in terms of collision avoidance because of its remarkable ability to detect approaching threats. The reason to this is because of the characteristics of the Locust's eye.
The surface of the eye is a compound of approximately 8000 lenses. Each lens has a series of arranged neurons reaching from the retina through the medulla and into the lobula, where they are conneted to the *Lobula giant movement detector (LGMD).* These neurons pass excitation to the LGMD when stimulated by an approaching object. The greater the number of signaling neuron units the more active the LGMD becomes.
The Locust neurovision system is depicted in figure 1 [22] and figure 2 [22].

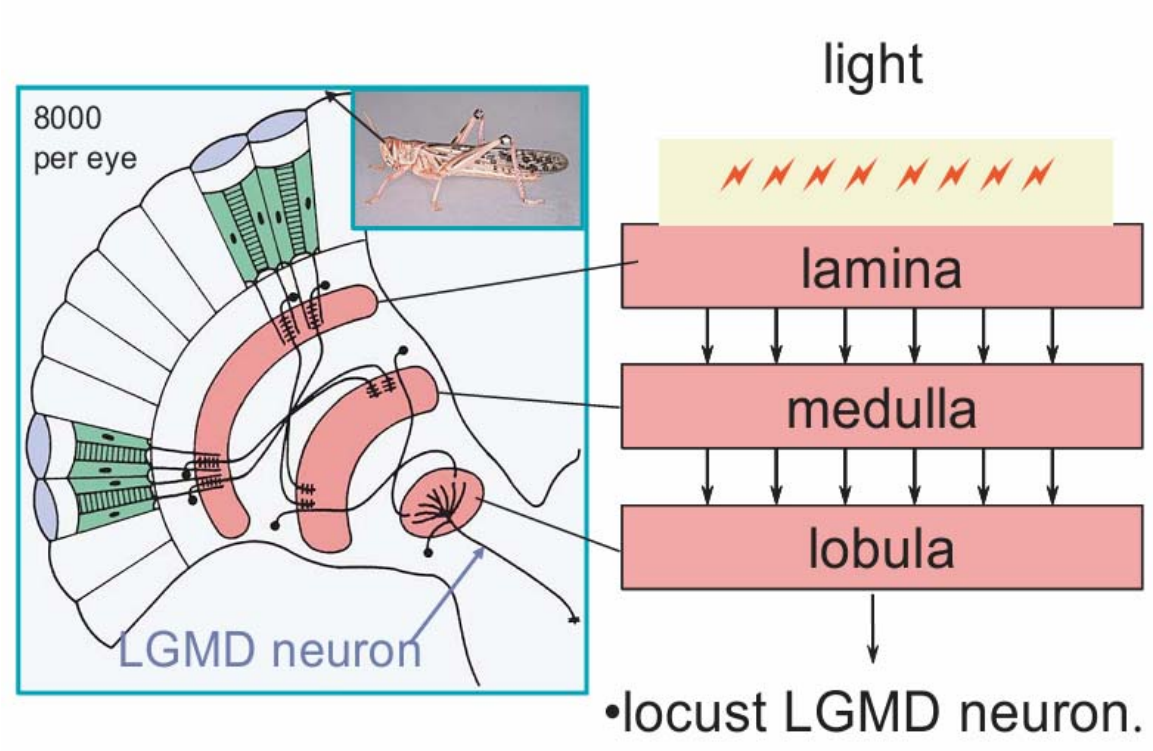


Figure 1: The Locust neurovision system.

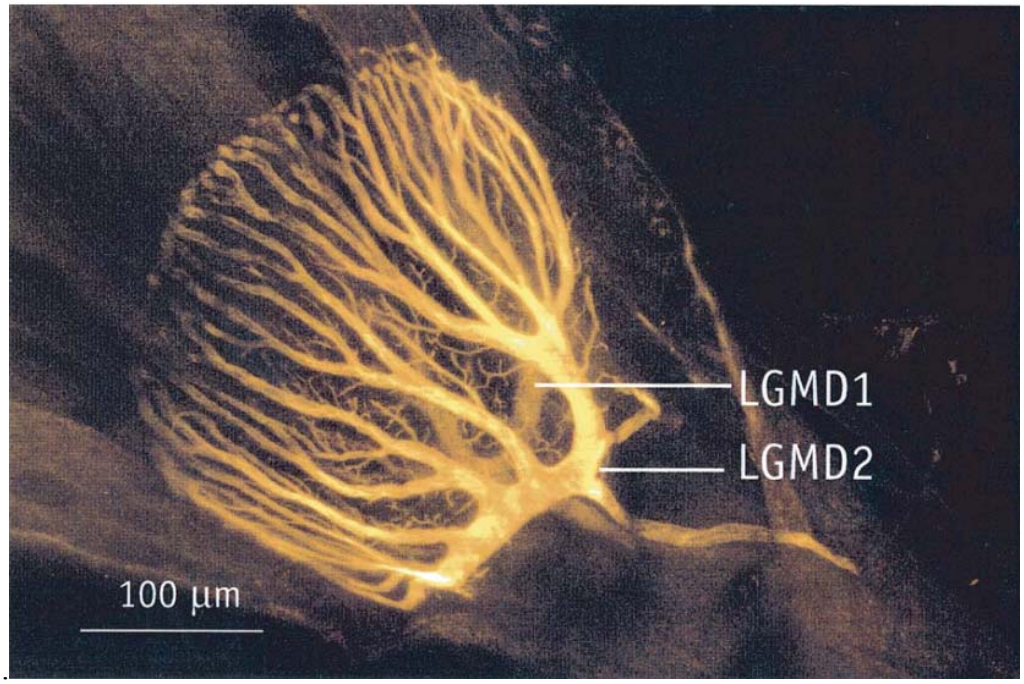


Figure 2: The lobula giant movement detector, LGMD.

2.2.1 The LGMD

The principle of the LGMD is based on the expansion of the image on the Locust's eye. The LGMD responds strongly to approaching objects as the image of them becomes bigger. At a certain stage, when an object is close enough, the intensity of LGMD signals to the brain reaches a certain high level, which will trigger the brain to execute an action to avoid the imminent threat.

For receding or stationary objects, the images don't grow so the activity of the LGMD stays small. This can be seen in figure 3 [22].

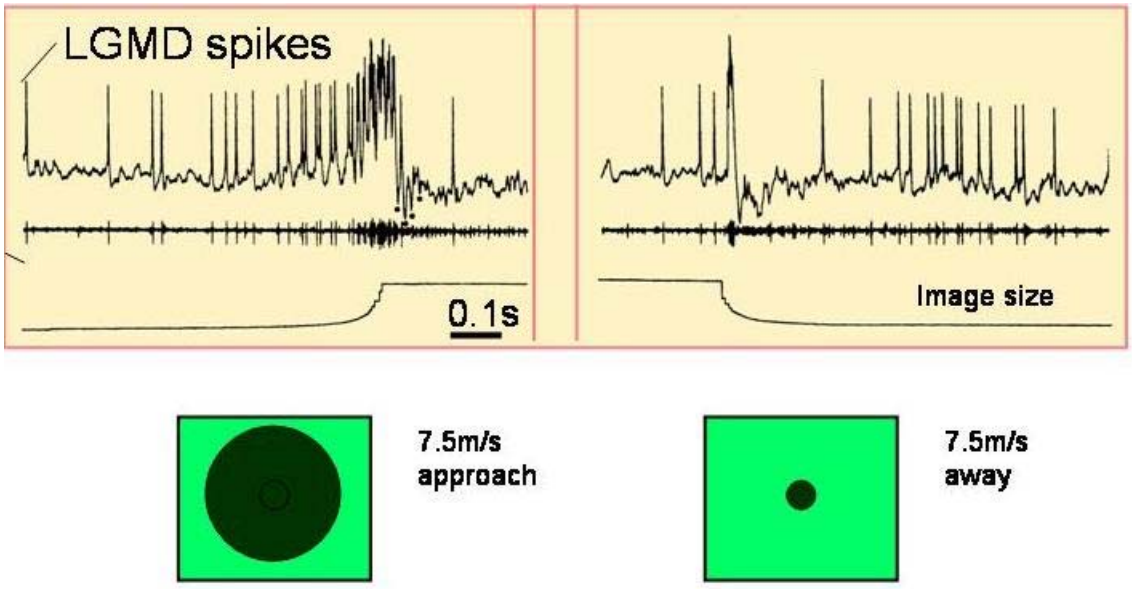


Figure 3: LGMD activity for an approaching and receding object

Because of the complex Locust neurovision system, the Locust's brain is not involved in the image processing process. This greatly benefits the reaction time against approaching objects.

2.2.2 The LGMD Model

In order to use the LGMD network as a collision detector for automotive applications, it must be tuned to detect the kinds of stimuli that a car might face. This was done in the earliest stage of the Locust project at University of Newcastle. The resulting model which is based on the input organization of the Locust LGMD neuron is shown in figure 4 [22]. Each element within the network represents a cell type in the Locust eye. The LGMD model is composed of four groups of cells: *photoreceptor, excitatory, inhibitory and summing*, and two single cells: *feed-forward inhibition* and LGMD.

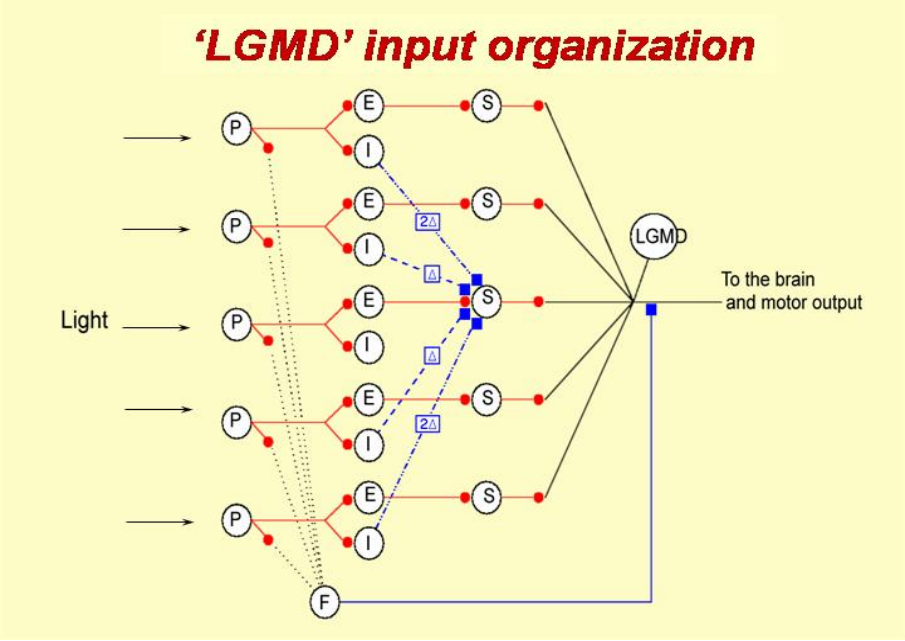


Figure 4: The LGMD model

2.3 Implementation of the Locust Model

2.3.1 Locust Sensor Chip

The hardware implementation of the Locust model is an emulation of the biological behaviour of the Locust grasshopper's collision avoidance system, which was studied in work package 1 of the Locust project. The hardware prototype was developed by a design team at IMSE. The result is a sensor chip that we refer to as the Locust sensor chip, which can be seen in figure 5.

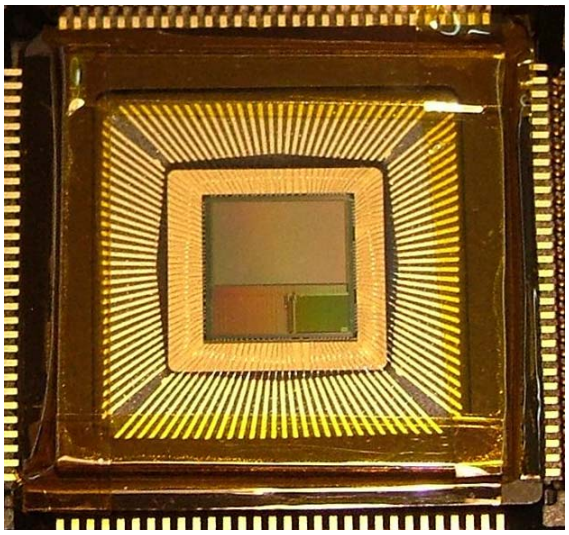


Figure 5: The Locust sensor chip.

As described in the previous section, the main strength of the Locust collision avoidance structure is that it uses heavily localized algorithms. Some data processing is done immediately at pixel level, which allows for quick reactions to perceived threats. Most vision systems used today consist of a camera for data acquisition and a separate digital processor to process it. This approach is too slow for some scenarios in automotive collision avoidance. The Locust sensor chip integrates the following functionality on a single chip:

- Sensor circuitry.
- Processing circuitry for adapting the sensor responses to changing environment conditions, analyzing the spatio-temporal characteristics of the visual flow, extracting the relevant information, and forwarding it to a higher level processor.
- Limited higher level data processing.

This approach can hopefully help improve the performance of current automotive collision avoidance systems, as well as reduce some of the hardware costs for automotive collision avoidance applications. The above functionality has been implemented using a mix of analogue and digital techniques, but the specifics cannot be presented here, since it is classified information that is yet to be published in an official paper by the Locust sensor chip design team at IMSE.

2.3.2 Bi-i

To be able to use the Locust sensor chip in a car, it was mounted inside a compact vision system called the Bi-i. This was done by our Hungarian partners from MTA SZTAKI in the Locust project. The Bi-i is essentially a powerful image processing unit capable of handling input from two visual sensors at the same time. In the Locust project only input from the Locust sensor chip was used for collision avoidance. So what we refer to as the Locust sensor system is a Bi-i unit with the Locust sensor chip mounted inside. A picture of the Locust sensor system can be seen below in figure 6.

Figure 6: The Locust sensor system consists of a Bi-i unit, which holds the Locust sensor chip.

The main reason for using the Bi-i in combination with the Locust sensor chip is that it was found in the biological studies that the Locust grasshopper's collision avoidance behaviour is extremely sensitive to threats coming from the side of the grasshopper. In particular it has a large percentage of false warnings when objects coming from the side pass in a perpendicular path close in front of the grasshopper [21]. Perhaps in the environment of the grasshopper these situations are not very common, since one of the main functions of its collision avoidance system is to avoid collision when flying in large swarms. Or perhaps for the Locust grasshopper a large number of false warnings in these situations is simply acceptable, since it might be more important that its warning system alarms for all real threats, even if that also means that the number of false alarms becomes high. Whatever the reason, it is clear that for an automotive application such behaviour is not acceptable. We want to capture the trait of the Locust grasshopper's quick collision reactions, but we do not want to keep the large number of false warnings associated with the above situations, since these situations are very common in a city driving environment for example. This is solved by applying additional image processing in the Bi-i. The specifics will not be presented here, but the general idea is to devide the image into several areas, and suppress the optical flow in areas to the side of the vision field. How to define these areas, and how to choose the suppression parameters was something that needed to be done through testing, which was mainly carried out at Volvo Car Corporation in Gothenburg.

3 Communication Protocols

3.1 Introduction

This section describes a couple of communication protocols that are used in the automobile industry. Knowledge of them was essential when performing various tasks within this thesis work.
In order to extract the desired numerical information such as speed, yaw and wheel angle from the XC90 test car, we had to understand how the communication network using the *CAN* protocol works. To accomplish the same goal with the model test car, the *TTCAN* protocol needed to be fully understood.

Controller Area Network, CAN, is the most commonly used communication protocol in modern automobile industry. It is event-triggered which means that messages can be sent from any control unit to the network bus whenever they want. As the reader will see, CAN is a simple protocol, fully equipped to accomplish typical vehicle functionalities. However, time-critical systems require a deterministic behavior in their networks. For this purpose, CAN is not a very good alternative and therefore TTCAN was developed.

Time Triggered Controller Area Network, TTCAN is still under development and at the moment only exists for research purposes. It is time-triggered which means that network units are fully synchronized with each other and message passing in the network follows a predefined schedule. This gives a deterministic communication with very little latency which makes TTCAN more appropriate for time- and safety-critical systems.

For a more detailed description of them, see [1], [2] and [3].

3.2 CAN

3.2.1 General

Controller Area Network, CAN, is a serial communication protocol created by Robert Bosch GmbH in the 1980s. It was specifically developed for the automobile industry and has since then become the most widely used communication protocol in modern vehicles.

3.2.2 CAN Protocol Layer

The layered structure of the CAN protocol is depicted in figure 7 [1].

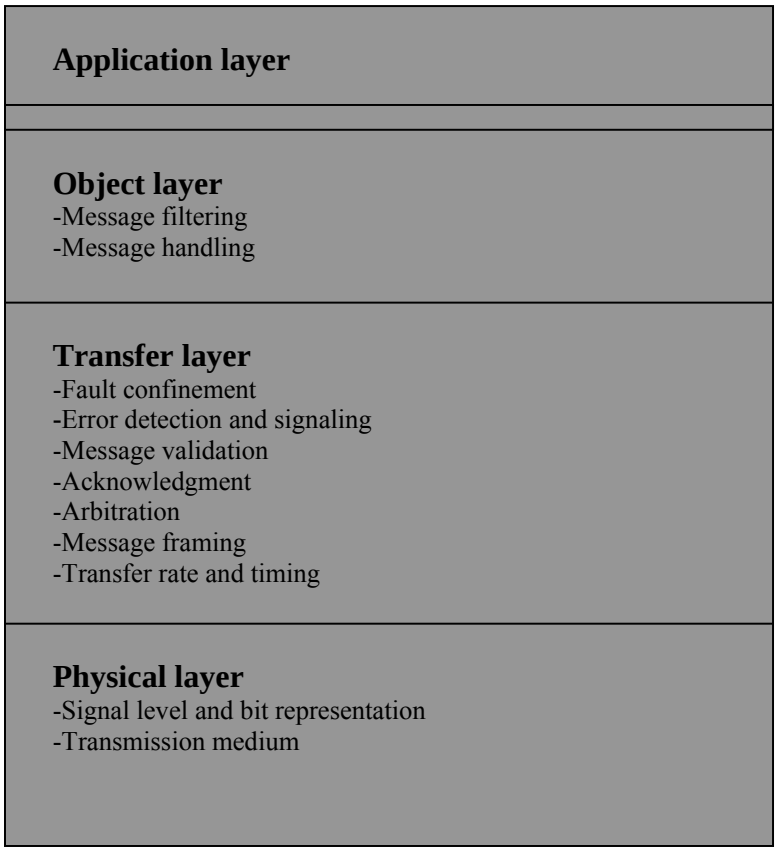


Figure 7: The layered structure of the CAN protocol.

The application layer provides the interface for user applications to access the information on the CAN network. Information passing between the layers is conducted through messages. Messages are forwarded to the application layer through the object layer which is responsible for message filtering as well as message handling. Message filtering is the process of selecting the right messages among many that are presented from the underlying layers. Message handling are all functions concerning the transmission of messages from and to the CAN network.

The core of the CAN protocol is the transfer layer. One can look at it as a control unit with the main task of keeping correctness during message transmission. This task is carried out with functionalities that perform synchronization, message framing, acknowledgment, error detection and signaling, and fault confinement.

The physical layer provides an interface to the serial bus line. It also defines physical requirements such as electrical levels and transmission medium.

3.2.2.1 Message Format

The transmission of information on the CAN bus is conducted through messages of fixed format in different but limited length. There are four different frame types:

- DATA FRAME – contains data information from a transmitter to all receivers on the network.
- REMOTE FRAME – A node can request for specific data by sending a REMOTE FRAME. The sender of the data will transmit the corresponding DATA FRAME upon receiving this message.
- ERROR FRAME – is transmitted when any unit has detected an error on the bus.
- OVERLOAD FRAME – is transmitted by any node to request for an extra delay between consecutive DATA or REMOTE FRAMEs.

The general format of CAN messages is depicted in figure 8 [3].

Figure 8: CAN message format

Apart from fields such as Data and CRC which one usually can find in other communication protocols, the Identifier is something uniquely of CAN. It works as an identity tag that distinguishes a message from the rest of the messages. The Identifier is used in message filtering and also in the process of arbitration, which is an essential part of the media access method used in CAN. This will be explained later in this chapter.

3.2.2.2 Transmission

Information on the bus is sent in digital form and can have the two complementary logical values: ‘dominant’ and ‘recessive’. These correspond to the values '0' and '1' in switching and logical theory. In the case of simultaneous transmission of ‘dominant’ and ‘recessive’ bits, the resulting bus value will be ‘dominant’. This can be seen in figure 9.

Bus state with two nodes transmitting			Logical AND		
	Dominant	Recessive		0	1
Dominant	Dominant	Dominant	0	0	0
Recessive	Dominant	Recessive	1	0	1

Figure 9: The state in a CAN bus can be either dominant or recessive.

3.2.2.3 Arbitration

Any node in the network can start to transmit a message whenever the bus is free. The CAN protocol uses a media access method that is called Carrier Sense Multiple Access, CSMA. With CSMA, any transmitting node that detects a collision on the bus has to stop sending if another message has higher priority. This procedure is called arbitration and can be seen in figure 10 [4].

During arbitration every sender compares the value of the bus with the transmitted bit value. If they are equal the sender will continue to send, if not, the sender has lost arbitration and must stop sending and withdraw.

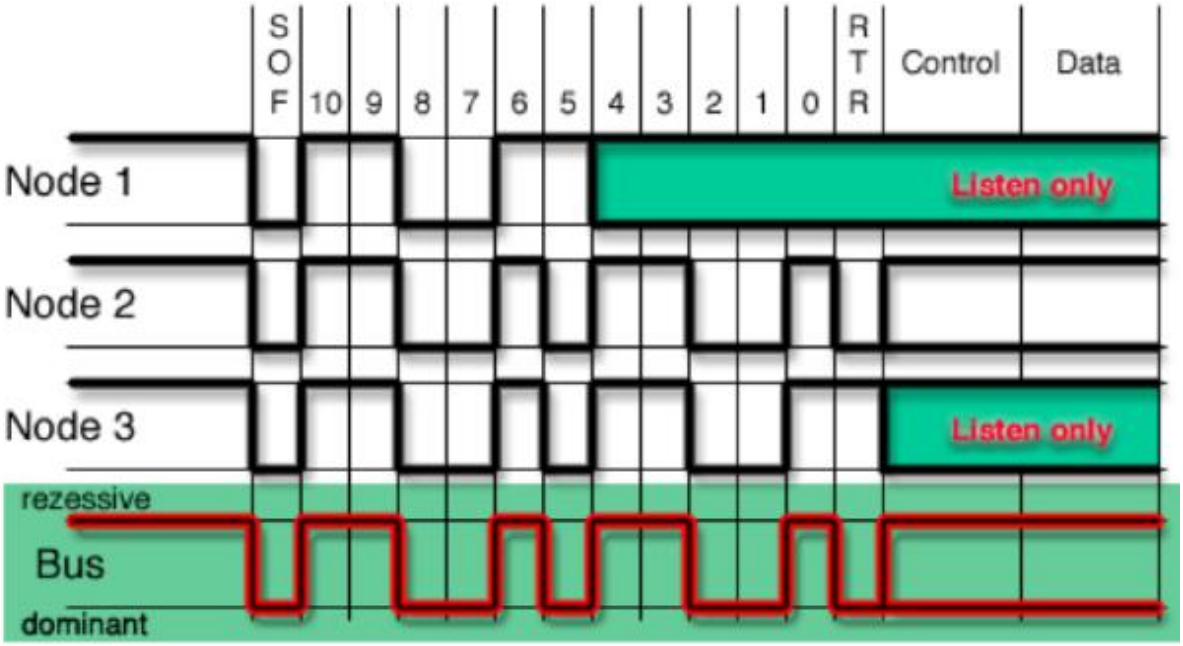


Figure 10: Arbitration with 3 nodes where node 2 eventually will prevail because the message it is sending has the highest priority.

3.2.2.4 Error Detection and Signaling

Safety in data transfer is enforced with various different mechanisms implemented in every CAN node. These are:

- Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus).
- Cyclic redundancy check
- Bit stuffing
- Message Frame check

Readers are referred to [1], [2] and [3] for more reading about these functionalities.
Network units can signal errors in the event of corrupted messages. These messages are discarded and will be automatically retransmitted by the sender.

3.3 TTCAN

Event-triggered protocols such as CAN have some disadvantages:

- There is no notion of a global time. This is not suitable for time-critical systems such as real time systems.
- Messages with lower priority can wait for transmission unreasonably long time.
- The usage of bandwidth is not optimized due to uneven demand for the bus. This is a result from the absence of message scheduling.

To overcome these disadvantages and still keep the simplicity of CAN, TTCAN was created. TTCAN is an extension of CAN that enables synchronization with a global time. This creates a time-triggered communication network which is deterministic and that uses the bandwidth much more efficiently. As a result of the deterministic characteristic of TTCAN, messages can be sent according to a schedule.

3.3.1 TTCAN Module

The TTCAN module is illustrated in figure 11 [2] and is basically a standard CAN controller with extensions of two new modules:

- Trigger memory unit – A memory unit that can store up to 32 time marks for the system matrix.
- Frame synchronization entity – A unit that consists of functions that handle the time schedule and the global system time.

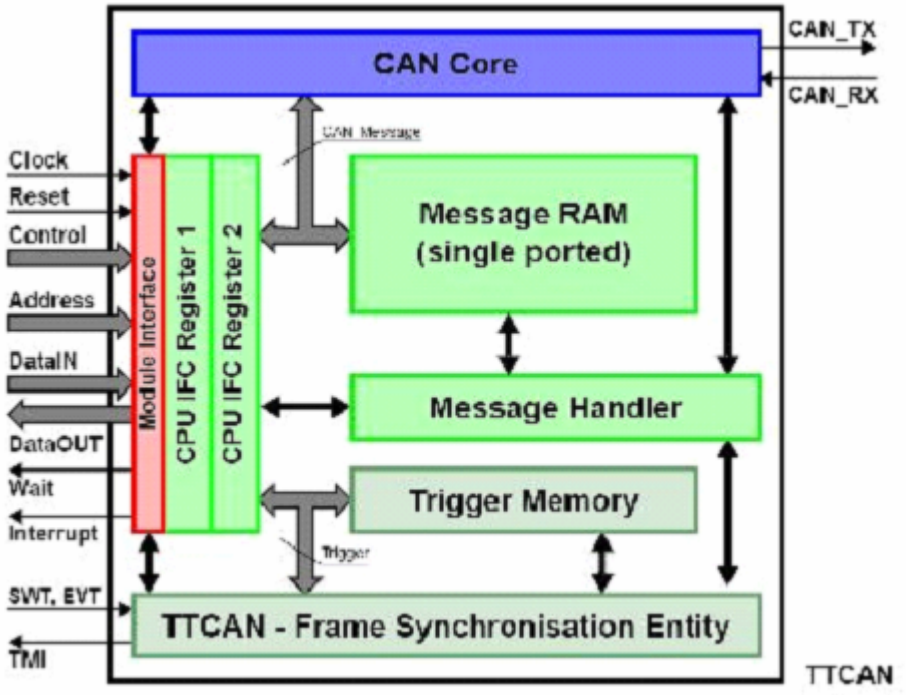


Figure 11: The TTCAN module.

Communication is still conducted through a CAN network with uniquely predefined message objects. Masks are used to identify and separate between messages. These are stored together with the message objects in the Message RAM. The message schedule is defined by time triggers. These are stored in the Trigger Memory. All functions concerning the handling of messages such as message acceptance filtering and transfer of messages are implemented in the Message Handler. The global system time functions are implemented in the Frame Synchronisation Entity.

TTCAN can be implemented in Level 1 or Level 2. The difference between these two is that Level 2 has functionalities which handle synchronization in a better way which enables higher accuracy in terms of transmission.

3.3.2 System Matrix

The message schedule, the system matrix, specifies the sequence of messages that are to be transmitted on the CAN bus. The system matrix contains one or several basic cycles that are composed of time slots. Each time slot is a time event for transmission of a specific message. This can be seen in figure 12 [5].

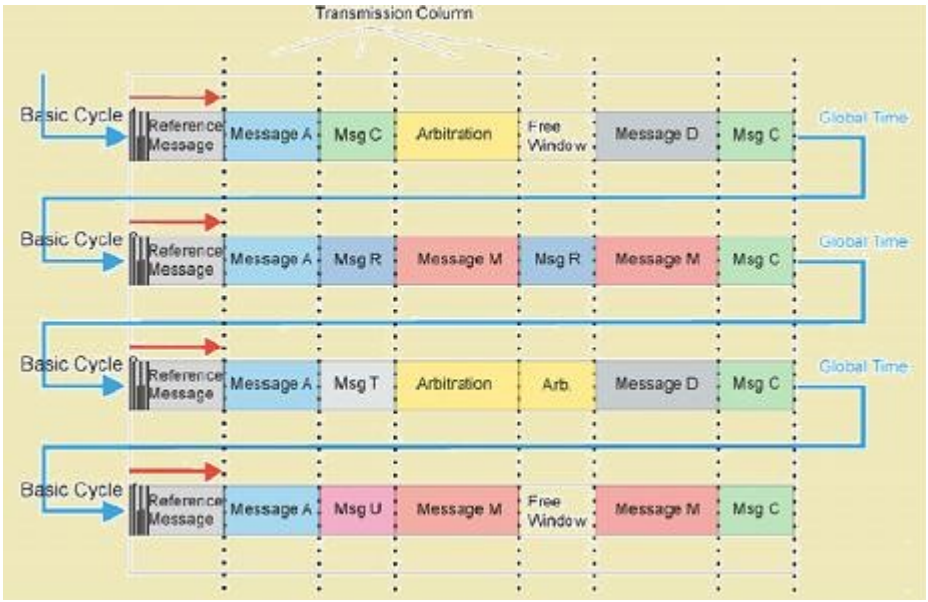


Figure 12: A matrix cycle consisting of four basic cycles.

All basic cycles start with a reference message which is transmitted by the time master. The time master is one of the nodes in the network that has taken the role as time master after a system reset. Any node with the potential to be the time master can take the role when it cannot detect a reference message on the bus after reset. It does so by sending a reference message with its own identifier. After receiving this reference message, a node has to synchronize its local time to the master's time mark which is included in the reference message.

The time slots in the matrix cycle can be of three different types: exclusive time window, arbitrating time window and free time window. During the exclusive time window only one node is allowed to access the bus and then transmit. On the contrary, all nodes can compete for the bus in the arbitration windows. This works exactly like communication with the CAN protocol. Two arbitration slots can be merged together to implement a relatively long CAN communication time period. This can be interesting for some specific applications. The free window slots are reserved for future extensions and implementation.

3.3.3 Triggers

Every single activity in the matrix cycle, either to transmit or to receive a message is activated by a trigger. The trigger contains information about at what time the trigger will activate, which message object the trigger concerns and if the message will be sent or received. There are 7 different triggers:

- Tx_Ref_Trigger - Sends reference message.
- Tx_Ref_Trigger_Gap – Sends reference message when in Gap mode.
- Tx_Trigger_Single - Sends date message.
- Tx_Trigger_Merged – Starts a merged arbitrating window.
- Watch_Trigger – Checks for reference message.
- Watch_Trigger_Gap – Checks for reference message when in Gap mode.
- Rx_Tripper – Checks for received message in exclusive slots.

3.3.4 Time Synchronization

Every TTCAN node controller has a time base, *Local Time*, which is incremented after every *Network Time Unit, NTU*. The NTU is the same for every node and is configured based on the global time and a prescaled variable called *TUR*. This can be seen in figure 13 [2].

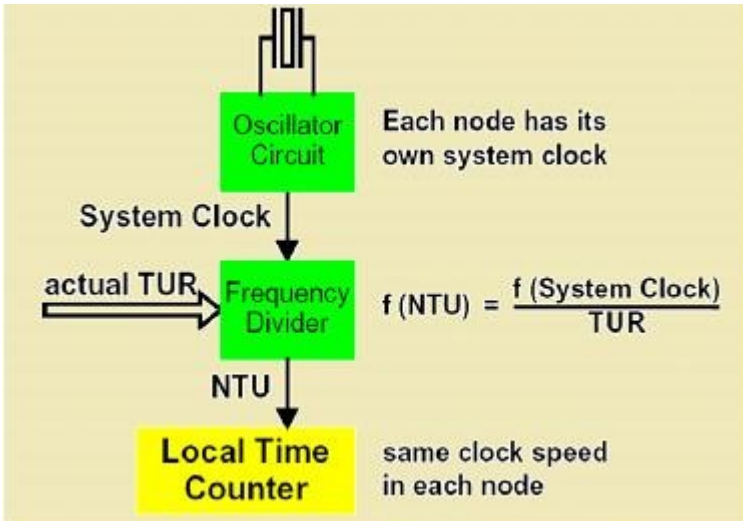


Figure 13: The calculation of Local Time.

A basic cycle time length is set by a timer which is reset at the beginning of every basic cycle. The cycle time is calculated from the Local Time which is showed in figure 14 [2]. The Local Time from every received message is kept in *Sync_Mark*. If the message turns out to be a reference message the Local Time value would then be forwarded to *Ref_Mark*. The Cycle Time will be derived as the difference between the recent local time and the Ref_Mark.

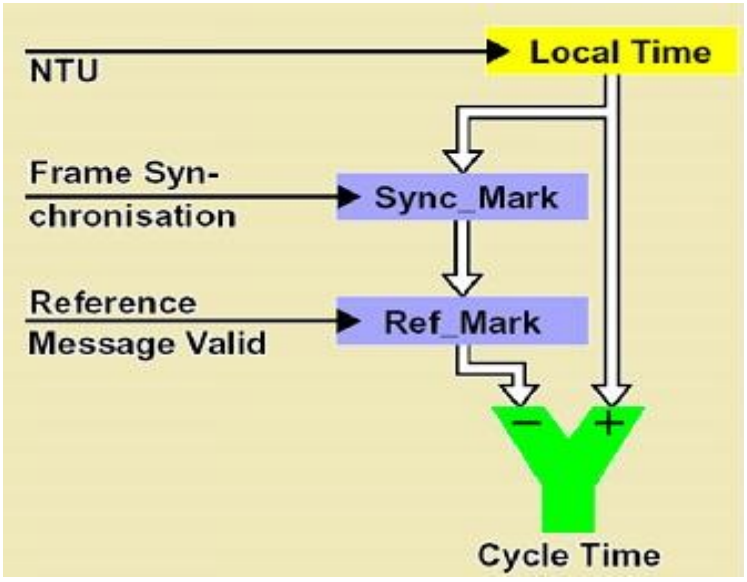


Figure 14: The calculation of Cycle Time.

The core of time-triggered systems is the notion of a global time and it is used to calibrate the Local Time in each node. The global time is obtained according to figure 15 [2].

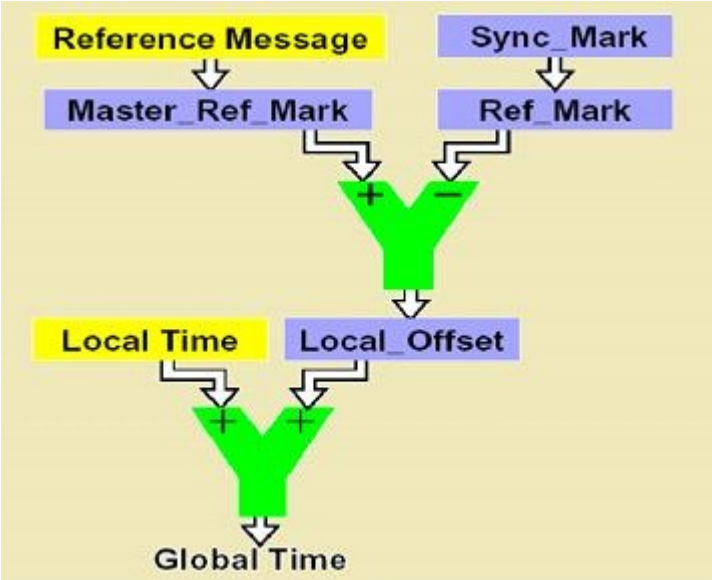


Figure 15: The calculation of Global Time.

Each reference message contains the time master's reference mark, *Master_Ref_Mark*. The difference between the own Ref_Mark and the Master_Ref_Mark gives the local offset which is then added with local time to calculate the global time.

4 Collision Avoidance

4.1 Overview

This chapter presents a summary of the basic theory for collision avoidance, with automotive applications in mind. We also propose an outline of how such a system can be constructed. When we talk about a collision avoidance system, we mean a system that has as its objective to avoid collisions and/or mitigate the effects thereof. Slightly different definitions exist but this is the one that we use in this chapter. A block diagram of a general collision avoidance system is shown below in figure 16 [9].

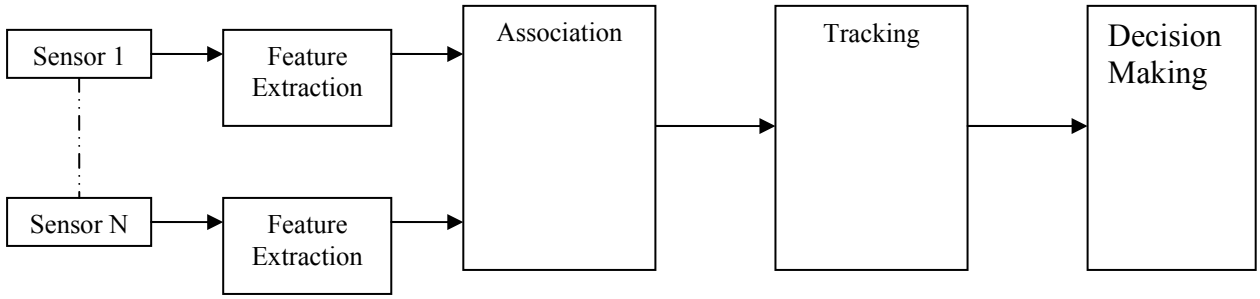


Figure 16: Block diagram of a general collision avoidance system.

First we need sensor(s) to provide our system with knowledge of the host vehicle's surroundings. In this project we have only one sensor, the Locust sensor system which is a visual system, but other types of sensors exist as well – in the automotive industry radar sensors are very common (often in combination with visual sensor systems). If there are multiple measurements, which is generally the case, the data from the sensors needs to be associated with the correct object (other vehicles, or stationary targets by the roadside for example). Since sensors typically do not measure the information we are interested in perfectly, because in reality there is always measurement noise, the data needs to be processed in some way to give an estimation of the interesting information based on previous measurements. In collision avoidance literature a very common way to do this processing is to use the Kalman filter. This is not the only way to make these estimations, but since it appears to be the most commonly used in many fields our description will be based on use of the Kalman filter. The above mentioned procedure - to use sensor measurements, associate the measurements to an object, and to estimate the interesting information for each object based on the measurements associated with that object – is commonly referred to as tracking. The tracking generates estimated movement paths of the objects 'seen' by the sensors, and can do so (more or less well) for points in time in the passed, present, and also in the future (and is then called prediction). For a collision avoidance system we are naturally interested in predictions of positions of threatening objects in the host vehicle's surroundings. Given these predictions the question left to deal with is what to do with the information, and this is in collision avoidance literature referred to as decision making. So in short a collision avoidance system typically consists of two main subsystems, a tracking system and a decision making system.

To do the tracking mentioned above, we need to model the motion of objects in some way. This model can be more or less complex, and the type of model chosen for any given application will naturally affect both the performance of and processing power needed for the

implementation, which is a trade off. The steps mentioned in this overview are described in more detail in the following sections.

4.2 Vehicle Motion Models

Since we want to use the Kalman filter for estimation and prediction of object positions we need a state-space model to describe the objects. Note that the main types of objects we have in mind are vehicles (cars, trucks, motorcycles), pedestrians, and stationary objects by the road side (typically about the same size as vehicles or pedestrians). We found the models described below in [9], and they are geared towards describing the motion of vehicles. We have not investigated other models that might be more suitable to describe the movement of pedestrians, instead we first planned to try a very simple model and hope that the performance would be good enough for both vehicles and pedestrians, and in case not try more complex models until the performance is acceptable. All the models presented below are two-dimensional where the vehicles are seen from above and their longitudal and lateral motion on the ground is described by x-y coordinates. The coordinate system is centered in the host car.

4.2.1 Constant Velocity Model

The constant velocity model assumes that the velocity is piecewise constant in between two consecutive measurements. It also assumes that motions in x- and y-directions are independent of each other. Changes in velocity (acceleration) are modelled by noise, which we will assume to be Gaussian. The equation for motion in one direction is described by the well-known formula (the time between two consecutive samples is the constant T, and n denotes the sample at time nT with n being an integer ranging from 0 to infinity):

$$x[n-1] + v[n-1]T = x[n]$$

The state-space equation in matrix form for the model is given below:

$$x = (P_x \;\; P_y \;\; V_x \;\; V_y)^T$$

$$x[n+1] = \begin{pmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} x[n] + \begin{pmatrix} T^2/2 & 0 \\ 0 & T^2/2 \\ T & 0 \\ 0 & T \end{pmatrix} w[n]$$

This model is linear. This is the model that we planned to try out first, since it is the simplest one and we can not get too much performance out of our hardware. The linearity of the model is also nice since the Kalman filter, which is described later, requires linear models. Therefore using this model means that no extra work to linearize the model for use in the Kalman filter is needed.

4.2.2 Constant Acceleration Model

The constant acceleration model is an extension of the constant velocity model where the acceleration is assumed to be piecewise constant between consecutive measurements. The changes in acceleration are modelled by noise, which we assume to be Gaussian. In one dimension the extension of the model yields (a denotes acceleration):

$$x[n-1] + v[n-1]T + a[n-1]\frac{T^2}{2} = x[n]$$

The state-space equation in matrix form then becomes:

$$x = (P_x \;\; P_y \;\; V_x \;\; V_y \;\; A_x \;\; A_y)^T$$

$$x[n+1] = \begin{pmatrix} 1 & 0 & T & 0 & T^2/2 & 0 \\ 0 & 1 & 0 & T & 0 & T^2/2 \\ 0 & 0 & 1 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & T \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} x[n] + \begin{pmatrix} T^3/6 & 0 \\ 0 & T^3/6 \\ T^2/2 & 0 \\ 0 & T^2/2 \\ T & 0 \\ 0 & T \end{pmatrix} w[n]$$

This model is linear.

4.2.3 Nearly Coordinated Turn Model

The nearly coordinated turn model assumes that the velocity is piecewise constant, but also models that vehicles can turn at a piecewise constant turn rate ω. The turn rate is usually not known and must then be included in the state-space equation so that it can be estimated. The state-space model is given below in matrix form:

$$x = (P_x \;\; P_y \;\; V_x \;\; V_y \;\; \omega)^T$$

$$x[n+1] = \begin{pmatrix} 1 & 0 & \frac{\sin(\omega T)}{\omega} & \frac{1-\cos(\omega T)}{\omega} & 0 \\ 0 & 1 & -\frac{1-\cos(\omega T)}{\omega} & -\frac{\sin(\omega T)}{\omega} & 0 \\ 0 & 0 & \cos(\omega T) & \sin(\omega T) & 0 \\ 0 & 0 & -\cos(\omega T) & \sin(\omega T) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} x[n] + \begin{pmatrix} T^2/2 & 0 & 0 \\ 0 & T^2/2 & 0 \\ T & 0 & 0 \\ 0 & T & 0 \\ 0 & 0 & T \end{pmatrix} w[n]$$

This model is non-linear. If we remove the row corresponding to the turn rate ω we get what is called the coordinated turn model, which requires that the turn rate is known.

4.3 Data Association

For a more complete description of the topic, see [8] and [9]. When the tracking system is active, measurements from the sensors are continuously forwarded as inputs to the tracking system. Data association is the process of associating observations to predicted track positions. This is an important issue in CA since a tracking system is usually multi-targeting, which means that several objects are being tracked simultaneously, and it is therefore essential to know if a new measurement belongs to a predicted track or not. If not, the tracking system will then decide whether or not to initiate a new track for that measurement. The data association process involves two steps:

- *Gating*
- Using a *Measurement association method.*

4.3.1 Gating

Gating is the first step of the data association process. Its task is to eliminate unlikely observation to track pairings. The gating procedure is illustrated in figure 17 [8] and is started by forming a circular gate around every predicted track. Predictions of tracked objects are here denoted with Ps and observations with Os.

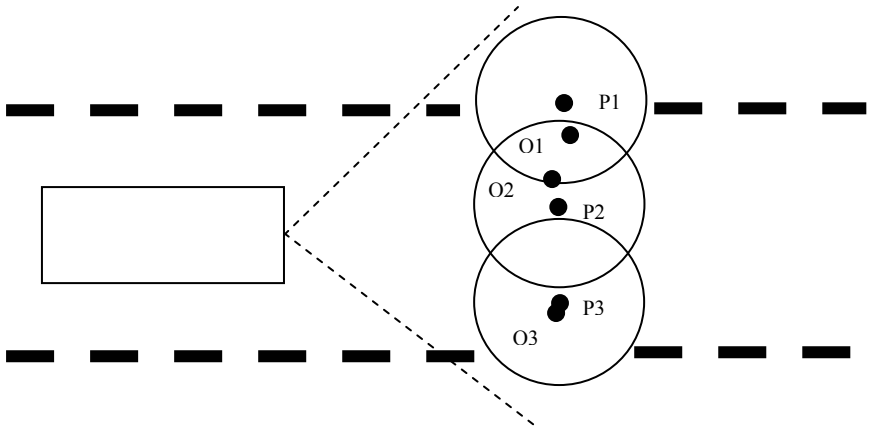


Figure 17: Moving vehicle with active tracking system that performs gating.

If an observation falls within a gate, it is considered to be a candidate to update that track. Several observations can fall within the same gate and an observation can also fall within more than one gate. In these situations further conflict resolving techniques are required to determine which observation belongs to which track. This is carried through in a later stage of the data association process. If one observation falls within only one gate and there are no other observations in the same gate, then the observation will be used to update that track.
In this example, usage of circular gates has been illustrated. There are other gating techniques such as, *Rectangular* and *Ellipsoidal* gates. These are further discussed in [8] and will only be mentioned in this thesis.

4.3.2 Measurement Association Methods

As stated before, the gating procedure can result in conflicts, for instance when a track cannot be associated to one unique observation. This can be resolved by using a measurement association method.

4.3.2.1 Nearest Neighbour Approach

The nearest neighbour approach is a simple method which uses only one observation to update one track. It does so by creating distance measures for every observation to every track and then choose the assignments with the shortest total distance.

Another way that is similar to the nearest neighbour approach is proposed in [10]**.** The idea is to use a distance measure between the observations and existing tracks, just like above. The observation-track pair closest to each other are associated with one another and then removed from the association problem. This is iterated until all observations have been associated with a track. This forms what could be called a greedy nearest neighbour approach and is not an optimal solution. However it has the advantage of being computationally less demanding than the regular nearest neighbour approach.

There are many other association methods available, such as: *The Multiple Hypothesis Approach* and *Clustering Techniques.* These are further described in [9] and we only mention them in this thesis.

4.4 The Kalman Filter for State Estimation and Prediction

The Kalman filter has been widely used in many signal processing applications. We have gathered information about the Kalman filter mainly from [9], [12], [17], and [18]**.** It was originally described in [19] and there are many other sources that present the Kalman filter.

The Kalman filter requires the use of a state space model to describe the dynamics of the system we deal with. Such a model is generally described by the state-space equations (with n denoting the sample taken at time nT, T being the sample time and n a non-negative integer):

$$x[n+1] = A[n]x[n] + B[n]w[n] \qquad w[n] \in N(0, Q[n])$$
$$y[n] = C[n]x[n] + v[n] \qquad v[n] \in N(0, R[n])$$

The Kalman filter is a linear filter and needs linear equations. x is a vector of states (this can be position and velocity for example) and the first equation describes the dynamics of the system. The matrices A and B can vary over time. w is the process noise and describes uncertainty in the system behaviour, such as driver input (which is not known by the system) and/or extraneous circumstances affecting the system like wind (that can change the course of a vehicle). The second equation describes what we measure, y[n], and we here assume that we are interested in knowing the states (otherwise we could construct a linear combination of the states to form another linear expression, but the states hold all the necessary information to capture what we are interested in, so we leave the notation as simple as possible). v is the measurement noise and models the fact that we cannot measure anything perfectly, in reality there are always some unknown things affecting our measurement. The noises w[n] and v[n] are described as stochastic processes and can be of any distribution. Q[n] is the covariance matrix for w[n] and R[n] is the covariance matrix for v[n]. Both Q and R can vary with time. The relation between Q and R describes how much we trust the model in relation to how much we trust our measurements. A large Q in relation to R means that we think the model cannot be trusted very much and instead we rely more on the measurements. Vice versa means we think the measurements are not very reliable and so we instead rely more on our knowledge of the system dynamics (that models for example that a car can only perform a rather limited set of maneuvers, given its speed and position). If the noise distributions are Gaussian the Kalman filter gives the best estimation, in terms of minimizing the state estimation error covariance P[n]. If they are not Gaussian the Kalman filter is the best linear

filter (but there might be better non-linear filters). The Kalman filter results in the following equations for the current state estimation $\hat{x}[n \mid n]$, 1-step prediction state (the filter prediction of the state one sample ahead in time) $\hat{x}[n+1 \mid n]$, the current state estimation error covariance matrix $P[n \mid n]$, and the 1-step prediction state estimation error covariance matrix $P[n+1 \mid n]$:

$$\hat{x}[n \mid n] = \hat{x}[n \mid n-1] + P[n \mid n-1]C[n]^T (C[n]P[n \mid n-1]C[n]^T + R[n])^{-1} (y[n] - C[n]\hat{x}[n \mid n-1])$$
$$P[n \mid n] = P[n \mid n-1] - P[n \mid n-1]C[n]^T (C[n]P[n \mid n-1]C[n]^T - R[n])^{-1} C[n]P[n \mid n-1]$$
$$\hat{x}[n+1 \mid n] = A[n]\hat{x}[n \mid n]$$
$$P[n+1 \mid n] = A[n]P[n \mid n]A[n]^T + B[n]Q[n]B[n]^T$$

The above equations give a recursive algorithm to calculate the filter in every time step. The first two equations are called the measurement update, and the last two equations are called the time update.
The expression $P[n \mid n-1]C[n]^T (C[n]P[n \mid n-1]C[n]^T + R[n])^{-1}$ is often denoted by L. The measurement update adjusts the state estimation according to the new information received from the measurement. The time update makes a prediction of the next state (the state in the next time step) given the information received up to the current discrete time n. To get started we need initial values for $P[n \mid n-1]$ and $\hat{x}[n \mid n-1]$, i.e. we need to seed the filter with (desirably well-chosen) values for $P[0 \mid -1]$ and $\hat{x}[0 \mid -1]$, assuming we start the filter at discrete time n=0.

In a CA tracking system we probably want state estimations for more than one time-step ahead (if we have a sample rate of 30 Hz for example, this means that a 1-step prediction only predicts the states of the system about 30 ms ahead, which is too short for a CA system used in a car). To get predictions several time-steps ahead the time update equation is simply iterated as many times as desired, which can yield a prediction for any number of time-steps ahead. However each such re-iteration will increase the uncertainty of the prediction (for that step) which means that the state estimation error covariance matrix P[n] will grow larger and larger. In other words the more time-steps ahead we try to predict, the less we can trust that prediction, and after a large enough number of predictions ahead the information from that prediction will be virtually useless.

A special case of the Kalman filter is obtained if the noise covariance matrices Q and R, as well as the A and C matrices are constant (do not vary over time). In this case the state estimation error covariance matrix, P, converges towards a constant matrix. This results in what is called the stationary Kalman filter, in which case P and L can be calculated in advance, to speed up the filter for real time applications. The equations to obtain these can be found in for example [12]**.**

If a non-linear state space model is used (for example the coordinated turn model) the model can be linearized around the current set of states in each time step. The Kalman filter can then be applied to the linearized model. This approach is called the extended Kalman filter, and is not the optimal solution (even if the noise is Gaussian), but a linear approximation. Note that the measurement equation $y[n] = C[n]x[n] + v[n]$ also should be linear. Linearizing can be done by using Taylor expansion around the current set of state estimations. Another way to deal with non-linear measurement equations is to do a non-linear transformation so that the transformed measurement equation is linear. An example of this is given in [9]**.**

Other ways to deal with non-linear models and measurement equations, as well as non-Gaussian noise distributions exist, such as the particle filter. However we have not investigated this and it is not treated further here.

4.5 Decision Making

The information in this chapter has been collected from [9] and [11]**.** The tracking system results in estimates of current and future positions of potential collision threats to the host vehicle. The problem left to deal with is to decide what to do with this information. This can be divided into two subtasks:

1. Determine whether or not the CA system should intervene at any given time, given the current information of potential threats.
2. What kind of intervention do we want the CA system to perform, given our knowledge of the situation.

By intervene we here mean both warnings and actions (like breaking or steering). In commercial products it is very important that the number of false interventions be kept to a minimum, especially in the case of autonomous CA actions. If a CA system would make a faulty intervention and thereby cause an accident it could result in severe consequences for the producing company. For this reason Volvo Car Corporation is very strict in its policy to only sell so called driver assistance systems, which means that the driver always has to make the triggering decision, and no completely autonomous actions are taken by for example CA systems. Some different types of warning/action strategies are discussed later in this chapter. First we briefly present some common ways to handle the first point above, to determine whether or not there is a dangerous situation coming up.

4.5.1 Deterministic Approach to Decision Making

A CA decision is based on the state estimate of the host vehicle and the surrounding objects. Because of process noise and limited accuracy in measurements, state estimates are not fully accurate but contain some level of uncertainty. With the deterministic approach, decisions are made without taking uncertainties into consideration. Before we go further into discussions about interventions, we have to define certain stages of a moving vehicle that are of our interest, according to [9]:

- Collision Avoidable: Here there is a threat to the vehicle. A non-negligible risk exists that a collision will occur. In this state it is still possible to avoid the imminent collision by an appropriate avoidance maneuver. Typically this state is perceived by a human as dangerous. What is perceived by a CA system as a dangerous situation will of course vary depending upon the CA decision function used. It is normally in this state that collision warning systems are activated. Any system intended to avoid collision has to act in this state.

- Collision unavoidable: In this state a collision is imminent, and cannot be avoided by any maneuver. Although the collision cannot be avoided, it might still be possible to significantly reduce its severity by reducing the collision speed and by taking other mitigating actions.

These states are not universal and hard to calculate because they are depending on factors such as the weather and the condition of the car. Despite this, it is clear that CA interventions are of interest in both stages. Interventions in terms of warning alarms should be triggered in

the collision avoidable state so that the driver has enough time to do a maneuver before reaching the collision unavoidable state.

Regardless of vehicle states, active interventions such as steering and breaking are always of current interest. According to [9], the action to take between steering and breaking, depends on the distance to the object in front, and the distance required to avoid an in front object is illustrated in figure 18 [9].

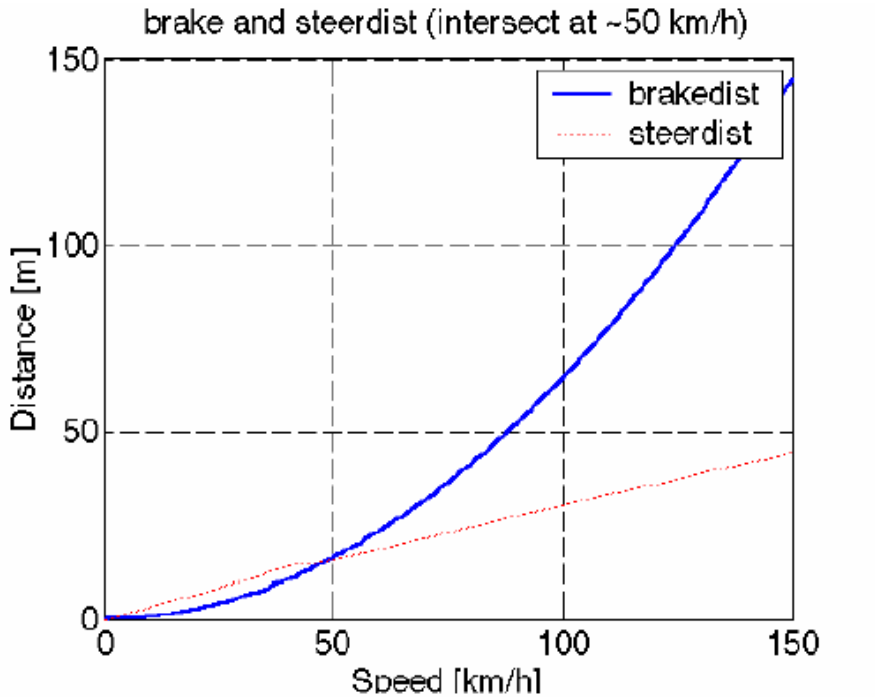


Figure 18: Distance needed to avoid a stationary obstacle with a width of 2 m by means of braking and steering.

This relationship is based on the assumption that the vehicle can instantaneously reach $9.82\,m/s^2$ deceleration and $9.82\,m/s^2$ lateral acceleration. As we can see in the graph, it is clear that for speeds lower than 50 km/h breaking is preferable over steering whereas steering is the better choice for higher speeds.

According to [9], a function g(X) that measures the threat level that the surrounding objects constitute is used to make any decision. This function usually bases its assumption on the future trajectories of each object. With a constant acceleration model, an object's trajectory can be estimated by:

$$P(t) = P_0 + V_0 t + \frac{1}{2} A_0 t^2$$
$$V(t) = V_0 + A_0 t$$

The Kalman filter can also be used for this as mentioned in section 4.4.

Some common decision functions are listed below:

- *Time to collision.* Assuming that the relative acceleration is constant, time to collision is calculated by solving:

$$0 = \tilde{p}_{x,0} + \tilde{v}_{x,0}t + \frac{1}{2}\tilde{a}_{x,0}t^2$$

where $\tilde{p}, \tilde{v}, \tilde{a}$ denote relative position, relative speed, and relative acceleration respectively. For the solution to this equation the reader is referred to [9]**.**

- *Headway* and *Headway Time*. The headway is the distance to the in front object, and is denoted p_{HW}. If we assume that the involved vehicles travel at equal and constant speed, the headway time t_{HW} is given by:

$$t_{HW} = \frac{p_{HW}}{v_0^{host}}$$

The *Time to collision* and *Headway time* metrics only handle one-dimensional motion, which means they require some other function to determine whether other objects are in the host vehicle's path.

- A decision function that considers motion in two (or possibly more) dimensions is the *Closest Point of Approach.* It is denoted P_{CPA} and is defined as follows:

$$P_{CPA} = \min_{t>t_0}(\left\|\tilde{P}(t)\right\|)$$

where $\tilde{P}(t)$ is the relative position of two vehicles. Assuming constant velocity and considering motion in two dimensions the separation, $\left\|\tilde{P}(t)\right\|$, between two vehicles is described as:

$$\left\|\tilde{P}(t)\right\| = \sqrt{(\tilde{P}_{x,0} + \tilde{V}_{x,0}t)^2 + (\tilde{P}_{y,0} + \tilde{V}_{y,0}t)^2}$$

This gives the *time for the closest point of approach*, t_{CPA}, by solving $\frac{d\left|\tilde{P}(t)\right|^2}{dt} = 0$:

$$t_{CPA} = -\frac{\tilde{P}_{x,0}\tilde{V}_{x,0} + \tilde{P}_{y,0}\tilde{V}_{y,0}}{\tilde{V}_{x,0}^2 + \tilde{V}_{y,0}^2}$$

which in turn gives the *closest point of approach* as:

$$P_{CPA} = \sqrt{\frac{(\tilde{P}_{y,0}\tilde{V}_{x,0} - \tilde{P}_{x,0}\tilde{V}_{y,0})^2}{\tilde{V}_{x,0}^2 + \tilde{V}_{y,0}^2}}$$

Readers that are interested in other decision functions are referred to [9]. In this thesis, they are only mentioned:

- *Potential Force*
- *Required Longitudinal Acceleration*
- *Required Longitudinal Acceleration for Piecewise Constant Motion*
- *Required Longitudinal Acceleration with Actuator Dynamics*
- *Required Lateral Acceleration*
- *Required Centripetal Acceleration*
- *Multi-Dimensional Acceleration to Avoid Collision*

4.5.2 Statistical Approach to Decision Making

The approaches in the previous section are all based on very intuitive metrics like relative velocity and position of the host and threat vehicles. In [11] another approach is presented, which is based on statistical theory. The idea is to calculate the risk of collision for some future steps in time. By steps in time we mean at future times nT were T is the update interval of the Kalman filter and n=0,1,2… . As mentioned earlier the Kalman filter is not the only possible solution to the tracking problem, but seems to be the one most widely used in collision avoidance literature.

The position estimates of the Kalman filter, together with the estimation error covariance matrix can be used to calculate confidence regions for the relative position of the host and threat vehicle, at a given (future) point in time (limited to sample points of the Kalman filter). The probability density functions (PDF) of the host vehicle and the threat vehicle (assuming there is only one) positions are combined to form the PDF of the relative position. If the host and threat vehicles are seen as points (with very little physical size) a relative position of (0,0) corresponds to a collision situation. We here assume a Cartesian coordinate system just like the one in section 4.2 previously. Of course real vehicles are not mathematical points but are rather described in the 2D ground plane by areas. A collision situation is then described as a situation where these areas overlap. The probability of a collision can then be calculated by integrating the PDF of the relative position over the overlapping area:

$$P(collision) = P(Pos_{host} - Pos_{threat} \in A) = \iint_{Pos_{host} - Pos_{threat} \in A} PDF_{relative\_position}(x, y)dxdy$$

where Pos_{host} is the position of the host vehicle, Pos_{threat} is the position of the threat vehicle, A is the overlapping area mentioned above, and $PDF_{relative\_position}(x, y)$ is the probability density function of the relative position of the host vehicle and threat vehicle.

4.5.3 Thoughts on Intervention Strategies

The decision making approaches presented above all have one thing in common, they deal with the problem of finding out whether a collision is about to occur, for a certain number of future time steps. They might calculate probabilities of collision or other measures to determine the risk of collision, in some meaning. A thought that struck us as we sifted through the various decision making strategies we have looked at is that none of them take into consideration the severity of a potential collision. They might decide on which maneuver is best to avoid a collision, given certain knowledge of the situation like relative position and

speed, among others. An extension of this could be to also consider what happens if a collision occurs. Consider the example from [9] where it is calculated that for a certain car, for relative velocities below 50 km/h the braking distance to avoid a collision is shorter than the distance needed by a steering maneuver to avoid collision. However since the tracking system only supplies us with estimates of the information we need, because there is process and measurement noise which means that states cannot be perfectly predicted, we cannot be totally sure that our calculations are always correct. This means that the CA system might think that it can completely avoid a collision (perhaps with a certain probability, in the case of a statistical approach), but there will be a possibility that the system is wrong. In this case when the collision occurs the CA system might have tried steering only (or at least not applied full breaks to keep the steering capability of the car as good as possible), which means that the collision speed could possibly be far greater than if full breaking had been initiated instead.

One way to combine the severity of collisions with a statistical approach like the one presented in [11] could be to introduce a severity measure of a collision. If the damage severity value of a collision with threat X is denoted $S(X)$, then the expected damage $E_{damage}(X)$ of the chosen threat X could be calculated as:

$$E_{damage}(X) = P_{collision}(X)S(X)$$

where $P_{collision}(X)$ is the probability of collision for threat X at a certain time step. How the damage severity value should be calculated remains a problem. It could be calculated assuming the CA system makes no intervention. However one can intuitively imagine that it should also somehow take into consideration the estimated probability that the CA system can actually avoid a collision with X. Perhaps in some situations the consequence of a failed avoidance maneuver might be so great that it could be worth to consider choosing a maneuver that will not lead to complete collision avoidance, but that one can be sure will mitigate the effects of collision, like applying full breaks for example. Even if the system thinks there is another maneuver that can completely avoid a collision, perhaps it is better to choose a lower speed collision, where one can be fairly sure the outcome will not be lethal, rather than go for complete collision avoidance, with the risk of colliding at clearly lethal speeds. According to [20], crashing into a fixed obstacle at speed 100 km/h has a 0.59 probability of mortal outcome, and reducing the speed by 15km/h reduces the probability of mortal outcome to 0.31. This is just an idea that we got, that has not been investigated further. It is also quite possible that someone has employed similar approaches in other literature, since it was beyond the scope of this project to do an extensive investigation about this subject.

4.6 CA System - Simple Implementation Outline

- Measurement association:
  1. Find the measurement that is closest to a track and associate the measurement with that track. Remove the track and measurement from the list of non-associated measurements.

  2. Iterate the above until all measurements have been associated with a track.

- Tracking:
  1. Do the measurement update of the Kalman filter, for each track:

$$\hat{x}[n \mid n] = \hat{x}[n \mid n-1] + P[n \mid n-1]C^T[n](C[n]P[n \mid n-1]C^T[n] + R[n])^{-1}(y[n] - C[n]\hat{x}[n \mid n-1]$$
$$P[n \mid n] = P[n \mid n-1] + P[n \mid n-1]C^T[n](C[n]P[n \mid n-1]C^T[n] + R[n])^{-1}C[n]P[n \mid n-1]$$

  2. Do the time update of the Kalman filter, for each track. This can be iterated for predictions more than one time step ahead (which will certainly be needed for a CA system):

$$\hat{x}[n+1 \mid n] = A[n]\hat{x}[n \mid n]$$
$$P[n+1 \mid n] = A[n]P[n \mid n]A^T[n] + Q[n]$$

- Decision making:
  1. For all of the time predictions made for each track, calculate the probability of collision in that particular time step:

$$P(collision) = P(Pos_{host} - Pos_{threat} \in A) = \iint\limits_{Pos_{host} - Pos_{threat} \in A} PDF_{relative\_position}(x, y)dxdy$$

  2. If the probability of collision for some track is higher than a threshold value, initiate automatic breaking (perhaps proportionate to the collision risk). More complicated maneuvers could be considered such as steering and/or breaking, given the estimated trajectory of the threat vehicle (track).

5 Model Car

5.1 Hardware

This section describes the model car used in this master thesis to host the Locust sensor system. The model car was originally developed in a project called FAR at KTH and later redesigned by Jonas Cornelsen and Patrik Dahlqvist in their master thesis at KTH. Both of the above projects were done in cooperation with Volvo Car Corporation in Gothenburg where the model car now resides. In this master thesis the model car's hardware has been left virtually unmodified but we still give a short summary of its general components and how they work together to give an overview of the system. All of the technical information has been collected from [13]. During the course of the project we put a lot of work into changing components and fixing wires and connections, to make the model car more robust, but the hardware design has not been changed. Later in the model car section we also describe the software developed for the model car. In this master thesis we modified and extended the previous code and it is the new version that will be presented. The hardware description is divided into two parts, mechanical platform and electrical platform. A complete description of the hardware design can be found in [13], the hardware description presented here is a shorter version of the same.

5.1.1 Mechanical Platform

The car has been constructed using parts from two standard scale 1:5 RC-cars purchased from the German manufacturer FG. These cars were originally outfitted with gasoline engines and were rear wheel driven and four wheel steered. The model car was built using the front parts from two such cars, so that all wheels in the resulting car have both driving and steering capability. All wheels are also equipped with disc brakes. The original gasoline motors have been replaced by DC-motors, one for each wheel. Furthermore each wheel has sensors for steering, braking and acceleration for feedback purposes to the control software.

The steering functionality uses four servos, one for each wheel, to be able to use different steer modes. The braking functionality also uses four servos, to be able to implement individual braking for each wheel and the acceleration functionality uses four DC-motors with servo amplifiers, to be able to use different drive modes. For a more detailed description of the mechanical platform see [13].

5.1.2 Electrical Platform Overview

A schematic of the electrical platform architecture is shown in figure 19 below.

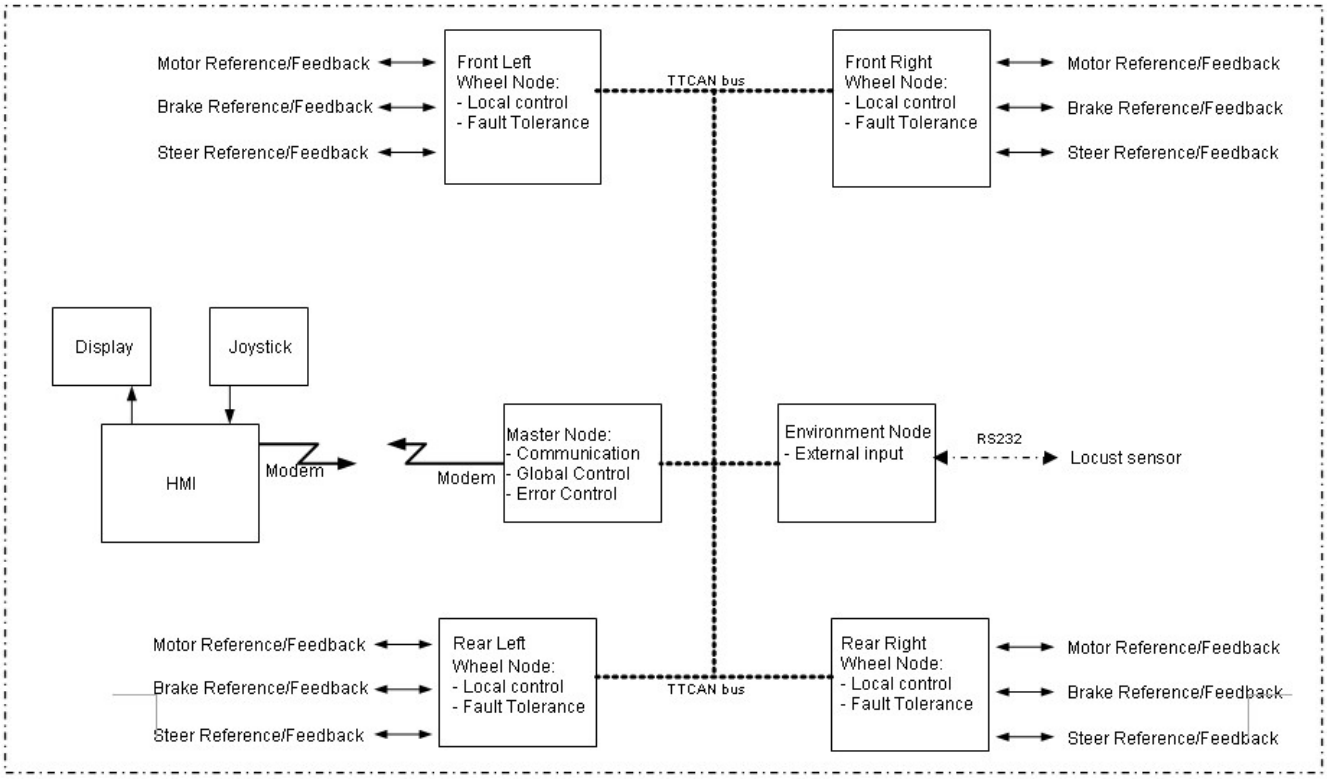


Figure 19: Schematic layout of the electrical platform.

All the control electronics are placed on a plastic board on top of the mechanical platform. The electronics consists mainly of six nodes and some connection boxes that connect the nodes to the mechanical platform servos and can also interface with a PC for programming and debugging. There is one node for each wheel, one master node for global control that also takes care of radio communication with the driver HMI (Human Machine Interface), and finally one environmental node that handles input from external units such as sensors. In this project the environmental node interfaces with the Locust sensor unit. The nodes make up a distributed computing environment and communicate with each other using the TTCAN protocol. For a more thorough description of how that works see the TTCAN section of this report.

Each node consists of three PCB:s (Printed Circuit Board) that are stacked vertically on racks. The bottom PCB is a GAST RTCOM TTCAN board (described later) that hosts two TTCAN controllers that handle all the triggers and basic functions for the time triggered communication between the nodes.

On top of the RTCOM board is a GAST G1 board (described later) which is the main microcontroller board in the node. All the control functionality resides in this board and the RTCOM board functions are called from here. In this thesis the G1 boards were reprogrammed to include the new necessary functionality for the Locust project, the RTCOM boards were not reprogrammed or modified in any way. Since the G1 board has a very central role in this thesis project it is treated in a separate subsection.

On top of the G1 board is an adaptation electronics board, which contains circuitry for power control and adaptation. Some functions in the model car need a 24V power supply, some need

12V, some need 5V and some 3,3V. There are also switches for fault injection that can be used for testing the robustness of the system. The adaptation boards are slightly different depending on the type of node which is described later. A picture displaying the three layers of one node is shown in figure 20.

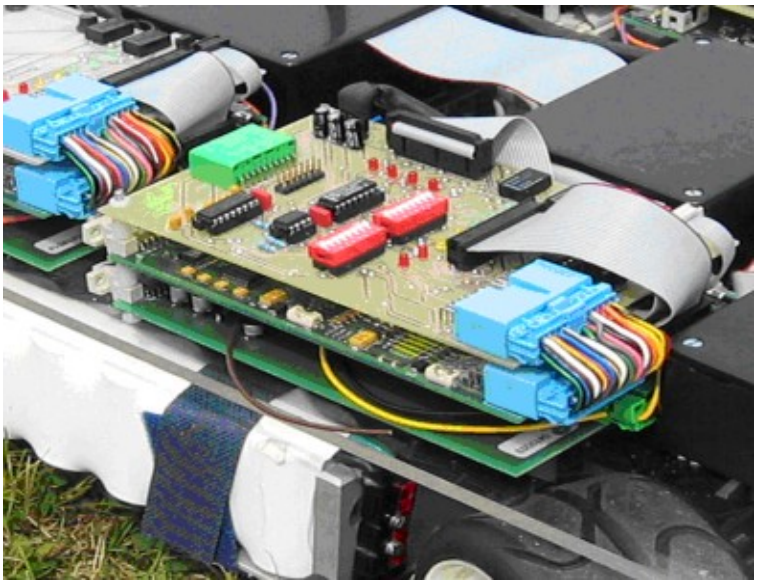


Figure 20: The realization of one node. It consists of three layers (described above).

5.1.3 GAST G1 Board

The GAST G1 board is a development platform for safety-critical systems. It supports several types of time triggered communication protocols. The G1 board is based on the 16-bits Motorola HCS12 family, also known as Star12, which is the next generation of the popular HC12 family. (The Motorola microcontroller branch is now its own company called Freescale Semiconductor.) HCS12 has more memory and peripherals and higher performance than the HC12. The G1 board is equipped with the MC9S12DG256 microcontroller, which has a bus frequency up to 25 MHz and has BDM (Background Debug Mode) capability. The G1 board itself holds the microcontroller and gives access to the peripherals. It communicates with the GAST RTCOM TTCAN board through a backplane. Each G1 board is capable of connecting with three communication controller cards (such as the GAST RTCOM TTCAN board) at the same time, but only one is used in the model car. A picture of the GAST G1 board can be seen in figure 21 [13]. For more information see the GAST project website [14].

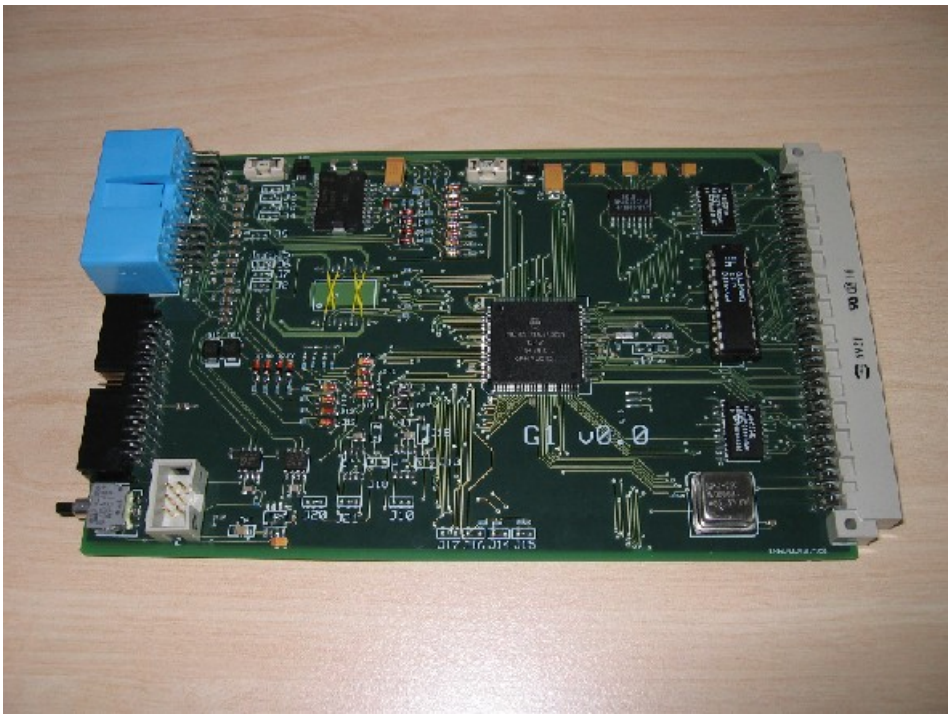


Figure 21: The GAST G1 board.

The MC9S12DG256 microcontroller contains a lot of useful peripherals. All of these are memory mapped which makes for a very appealing programmer's model. Also all of the registers are completely memory mapped which makes it easy to program the microcontroller using the C programming language, which is what we use in this thesis project. The microcontroller is equipped with among others both RAM and FLASH memories and also hosts two A/D-converters, a PWM (Pulse Width Modulation) module, two serial communication modules (referred to as SCI, Serial Communication Interface), a timer module (referred to as ECT, enhanced capture timer) and two CAN controllers (note that these are not used in the model car, but instead the separate GAST RTCOM TTCAN boards are used for inter-node communication. However in the XC90 part of this thesis the CAN functionality of the MC9S12DG256 microcontroller is used.).

The microcontroller also has many general purpose I/O pins that can output TTL voltage levels with a maximum current of about 10mA. Furthermore the microcontroller supports interrupts for most of the above mentioned modules, as well as interrupts for all of its general purpose I/O-pins. A simple schematic of the MC9S12DG256 microcontroller peripherals and features is shown in figure 22. For an in depth description of the HCS12 see Freescale Semiconductor's website [15].

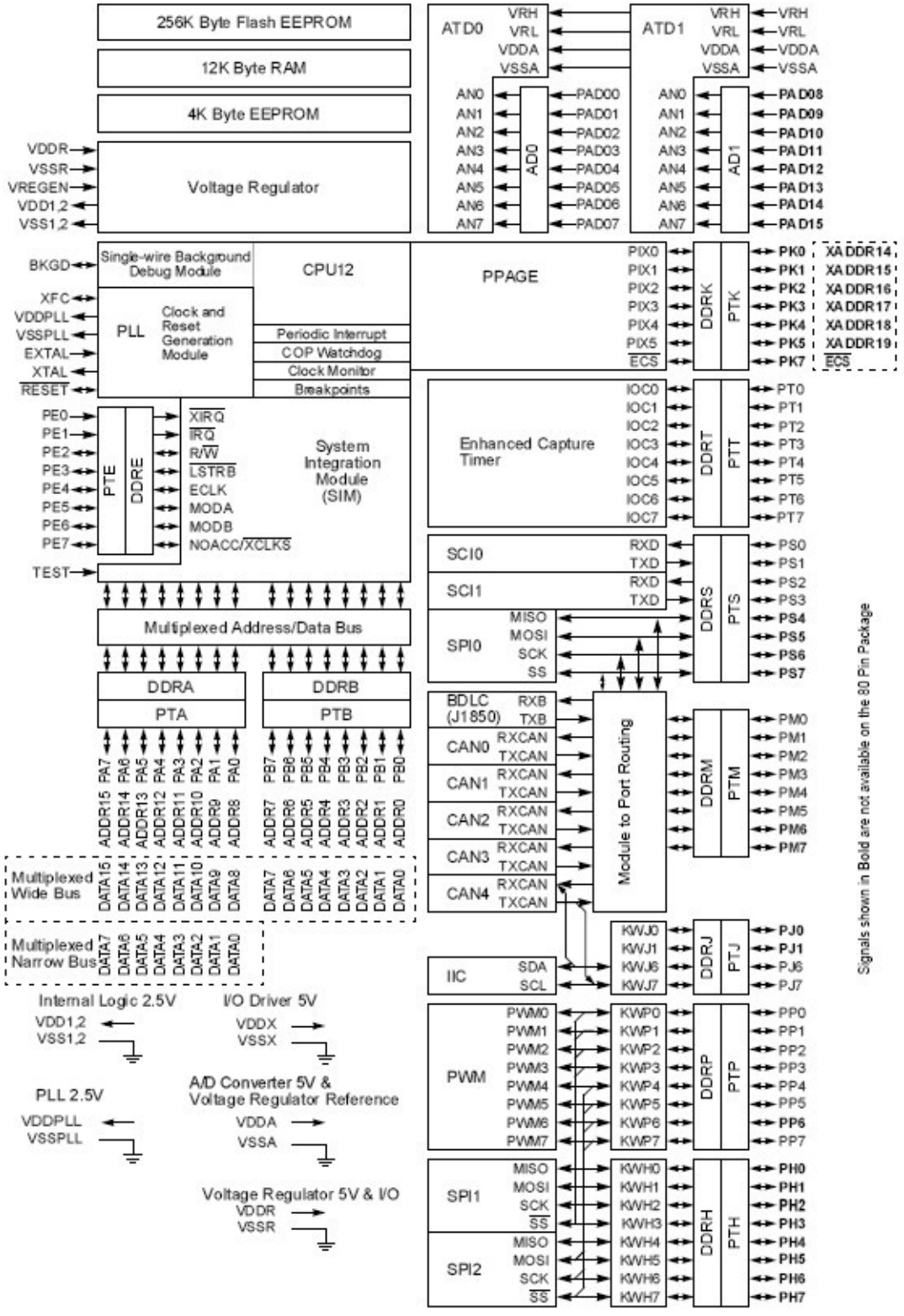


Figure 22: Main features and peripherals of the MC9S12DG256 microcontroller (HCS12).

5.1.4 GAST RTCOM TTCAN Board

The TTCAN controller board is a part of the GAST toolkit and it hosts two TTCAN Evaluation Chip controllers. The TTCAN controllers communicate with the G1 board through a backplane and it also gets its power supply through the backplane. In the model car only one of the TTCAN controllers is enabled because the two controllers interfere with each other (some problem that is not worked out in the evaluation version of the chip). To be able to interface with the TTCAN controller the G1 microcontroller (i.e. the MC9S12G256) has to be in expanded wide mode, which enables the external bus. Figure 23 shows a picture of the GAST RTCOM TTCAN board.

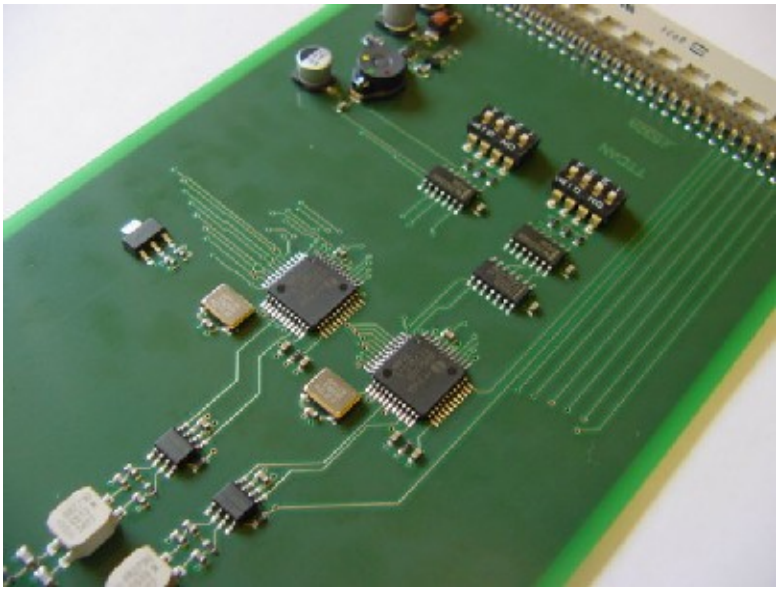


Figure 23: The GAST RTCOM TTCAN board.

5.1.5 Adaptation Electronics Board

Not all signals can be handled directly by the microcontroller on the GAST G1 board and need to be transformed. This is handled by the adaptation electronics board placed on top of the GAST G1 board. All signals pass through the adaptation electronics board which makes it easy to remove the whole node by simply disconnecting the adaptation electronics board.

The functionality of the adaptation electronics board is:

- Redistribute power to the GAST G1 board.
- Convert a PWM signal to an analogue signal to control the DC motor.
- Handle the encoder signal from the DC motor.
- Handle fault injection.
- Display the node status with LED:s.
- Enable BDM and RS-232 communication.

There are some switches for fault injection to be able to test the robustness of the model car as a whole. All adaptation electronics boards are identical for all the wheel nodes, with the exception of two jumper settings which are used to distinguish between the wheel nodes giving them each a unique ID. Figure 24 [13] shows a picture of the adaptation electronics board for the wheel nodes.

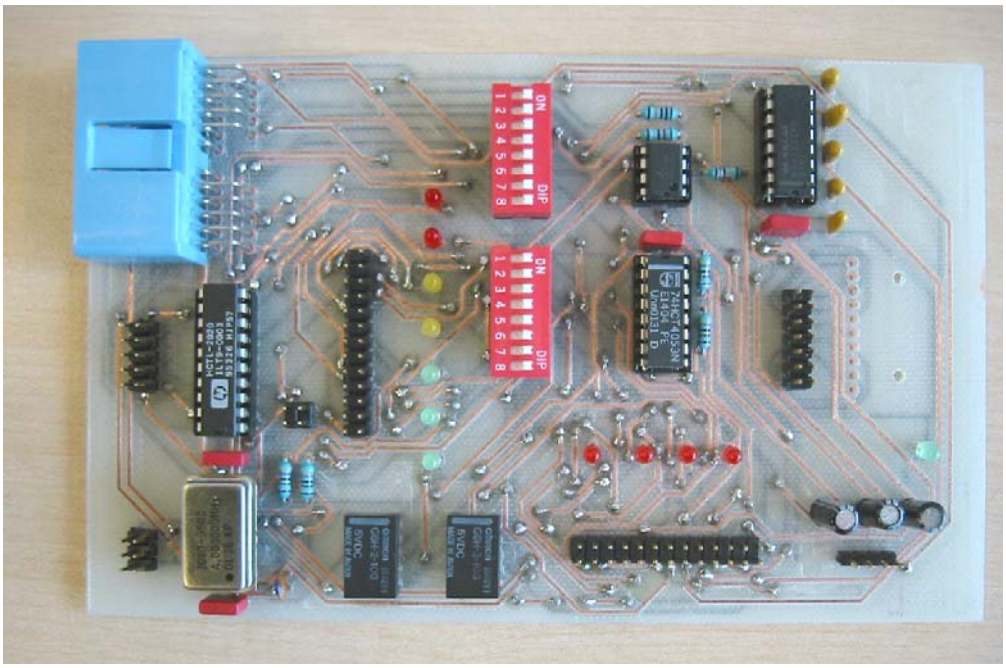


Figure 24: Wheel node adaptation electronics board.

The adaptation electronics board for the master node and the environmental node are different from the wheel nodes. They include all the functionality of the wheel nodes adaptation electronics, but the layout is different. In addition to this they both have some extra connectors for external input on general purpose I/O-pins as well as RS-232. The RS-232 connector on the master node adaptation electronics is used with a radio modem to communicate with the driver HMI to receive steering signals and send model car status information. A picture of the master node adaptation electronics board can be seen in figure 25 [13].

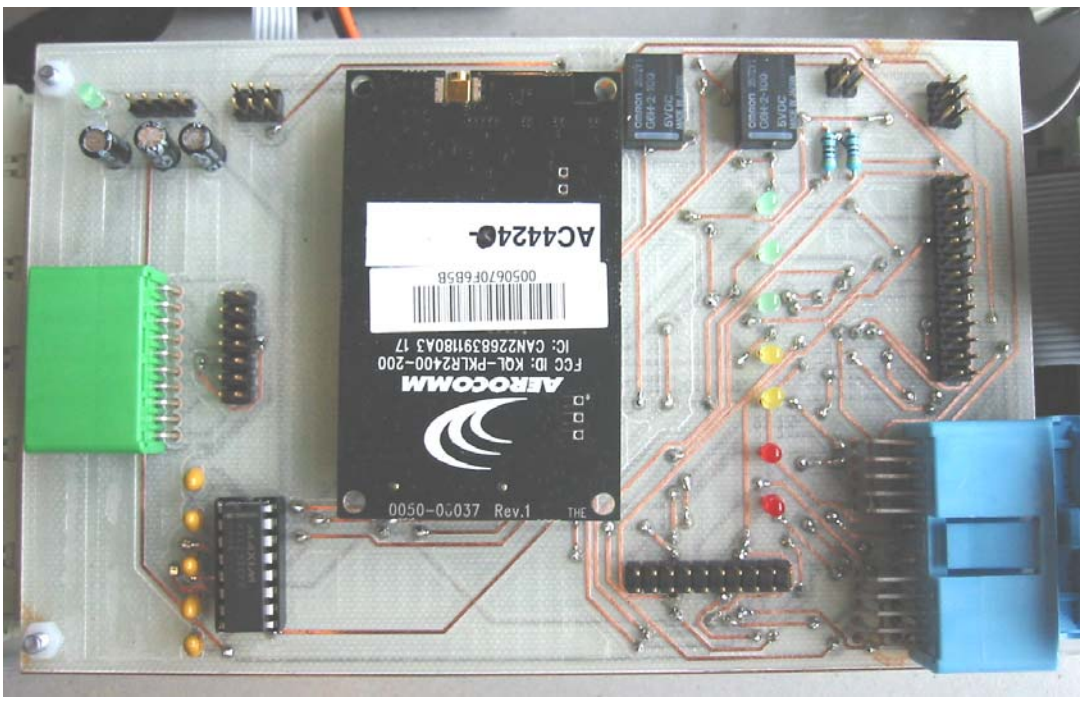


Figure 25: Master node adaptation electronics board.

The environmental node adaptation electronics board is identical with that of the master node, except that it does not have a radio modem. The idea is that the environmental node is to be used as connection port for miscellaneous external devices. In this thesis project it is used to interface with the Locust sensor unit using an RS-232 line. Other possible external connections include PWM and A/D pins. A picture of the environmental node adaptation electronics board can be seen in figure 26 [13].

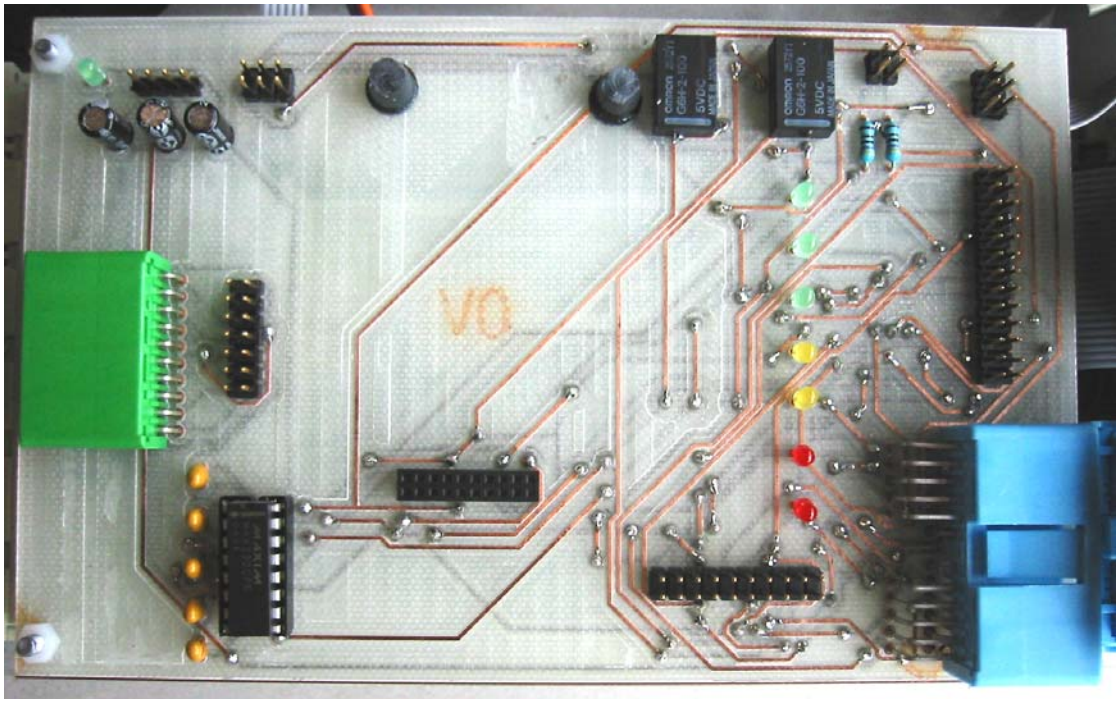


Figure 26: Environmental node adaptation electronics board.

For a detailed description of the design of these boards see [13].

5.1.6 Power Supply

The model car is supplied with power by two battery packs each consisting of 20 NiMH battery cells connected in series to give an output of maximum 24V and 55A. During driving with all nodes switched on the model car requires roughly 16A current to operate and with fully charged batteries the total driving time is about 25 minutes. Even just leaving the car running without actually driving requires much higher current than regular microprocessor power supplies can output. This was a bit of a problem for us since there are only two sets of battery packs available and the recharge time is about 7 hours (read one working day). In addition one of the battery packs is bad and only lasts about 3 hours of programming and debugging. This meant that some days we were out of luck and had to stop work half way through the day since we simply had no power supply for the model car. To always recharge the batteries at the end of the day is not a good option since that will eventually lead to even worse battery life time.

5.1.7 Physical Layout

A schematic of the physical layout is shown in figure 27 [13] below. The modules are numbered according to:

1. Adaptation electronics board.
2. The six nodes.
3. Connection box used to collect all sensor and actuator signals for the front wheels.
4. Power supply placed at the same level as the servo amplifiers.
5. The three middle connection boxes are used for the distributed control system.
6. A "token ring" connection box, used to manually select a node for debugging and programming.
7. Connection box used to collect all sensor and actuator signals for the rear wheels.

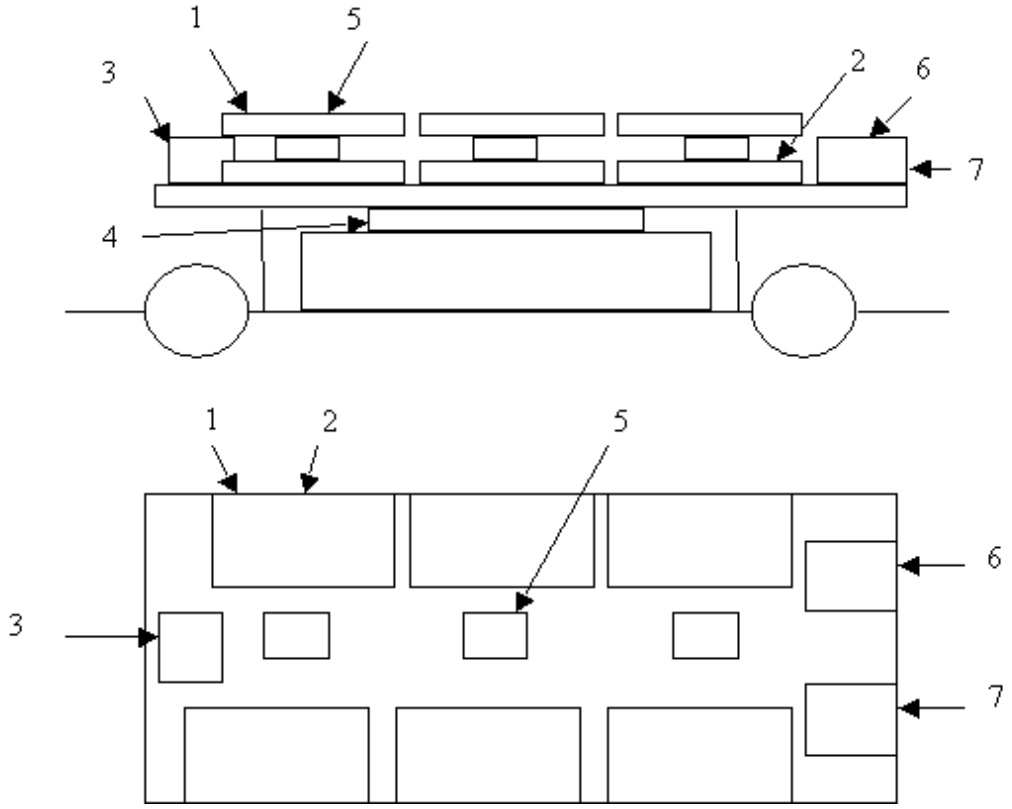


Figure 27: Schematic of the physical layout of the model car.

In figure 28 [13] and figure 29 [13] photos of the physical layout are shown from overhead and the side respectively. The reference numbers are the same as above.

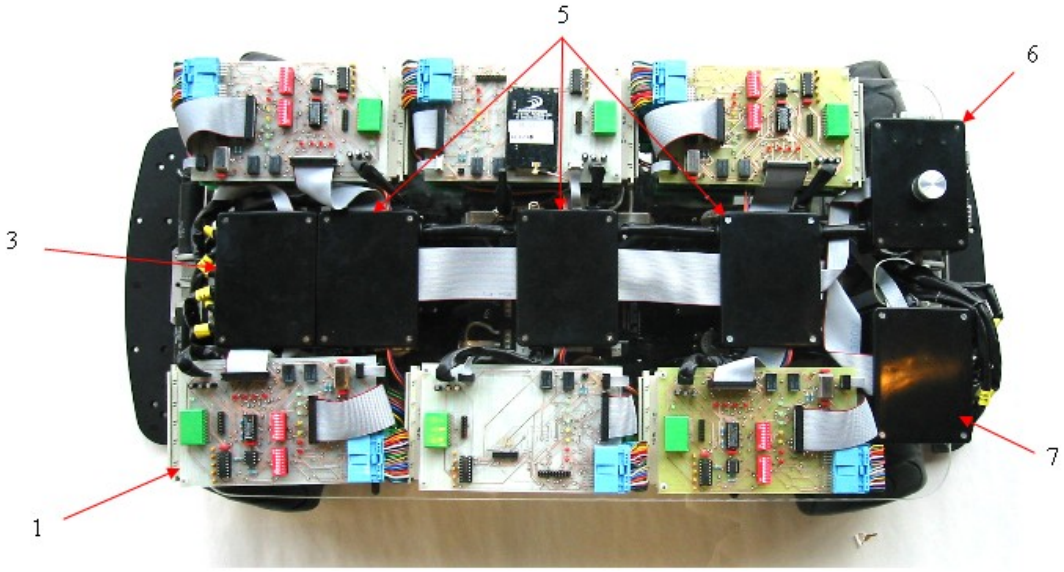


Figure 28: Photo of the physical layout of the model car from overhead.

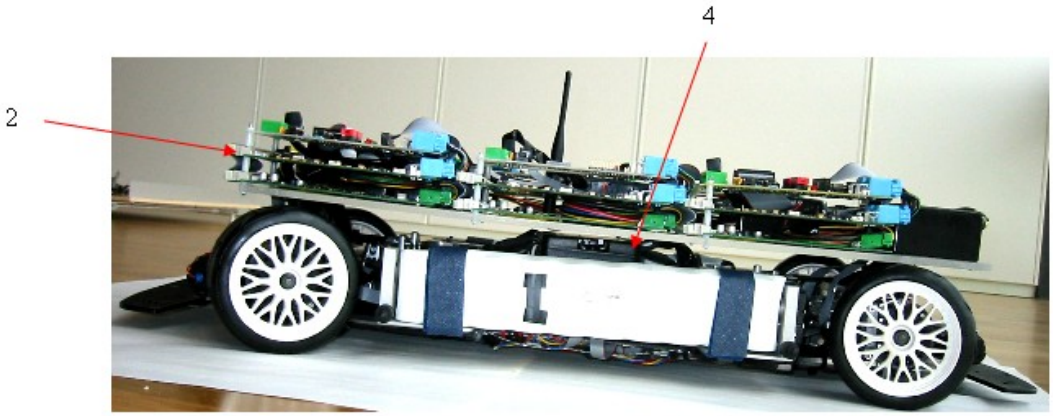


Figure 29: Photo of the physical layout of the model car from the side.

5.2 Software

This section handles the software of the model car. First a very brief description of the development environment is given. Then a general system overview is presented to give some understanding of how the model car works, and then a more detailed description of the communication interface between the model car and the Locust sensor system is given. Finally the collision avoidance strategy for the model car is presented. An overview of the C code files in the model car can be found in Appendix B.

5.2.1 Development Environment

This subsection is heavily influenced by [13]. All the code was written in C using the XCC Premium development environment. It is a cross C compiler supplied to us by Chalmers. The main features include:

- ANSI-C compiler, assembler and linker/loader.
- HCS12 simulator.
- Integrated Source Level Debugger, SLD, using the BDM interface.
- Full use of breakpoints, both in simulator and in hardware.
- Simple uploading of the compiled executables to HCS12 flash memory.

XCC Premium uses workspaces, which can include several projects. A project holds all information used for an application, such as settings and files used for the application. XCC has a project manager, which can be seen to the left in figure 30. It can also be seen which source files are included in the application. In the bottom window the output from the assembler, compiler and linker is shown.

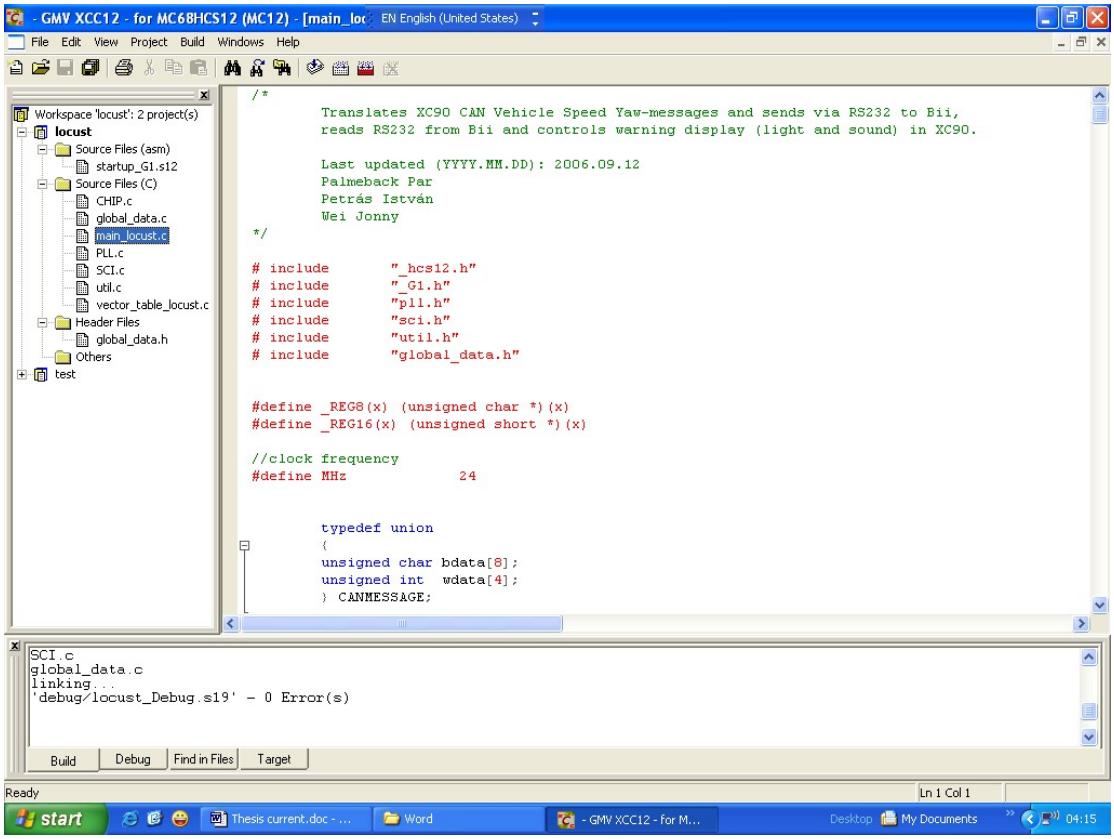


Figure 30: The XCC Premium development environment.

When the application is compiled and machine code is generated it is downloaded to the microcontroller with the Background Debug Mode (BDM) interface.

5.2.2 System Overview

The model car consists of six nodes, four wheel nodes, one master node, and one environmental node. An overview of the C code files in the model car can be found in Appendix B. The code is almost identical for all the wheel nodes, with a few minor exceptions that we will not go into here. However, to make the practical work with the code faster there is only one code version which contains the code for all the node types. Instead C macros are used to separate the functionality of the nodes (i.e. a statement like "#if NODE_TYPE == NODE_TYPE_WHEEL …"). This may be confusing when looking at the code, but it does make compiling for different nodes much faster since all of the code is always in the active project in XCC Premium. If the code had been separated into different files for each node, XCC would have to close and open a new project each time we want to switch node, which we had to do frequently. So it is much easier to just change a macro defining the current node type to compile for, even though reading the code can be a bit confusing at first. Were it not for this there simply would not have been enough time within the scope of this thesis project to implement anything, since the compiling and uploading of code to the microcontroller is rather slow (it takes about 4-5 minutes each time). Due to the complexity of the system only small changes can be made at a time, otherwise debugging is virtually impossible. Even though XCC fully supports breakpoints and simulation this is not an option in our case, because only one processor at a time can be simulated, and since most of the difficult-to-find errors are due to the distributed computing architecture of the model car, simulation of only a single processor at a time is of little use. Most of the problems are caused by timing errors between the nodes, since they all depend on the other nodes to compute the right thing at the right time.

When the car starts up there is some initialization that needs to be done. It consists mainly of TTCAN initialization, and it sets startup reference values for the control signals so that the car remains still before the radio communication is running correctly. Also the system clock, baud rates for the serial communication and the radio communication are set. Most of this initialization is done in the function initialize() that is called directly from the main() function. The macro defining which node type to compile for as mentioned above is located in the file initialize.h.

All of the nodes run a small real time operating system (RTOS) that has 11 task slots, except for the environmental node which only has 2 task slots (the reason for this is explained later in section 5.2.2.5 which covers the communication interface with the Locust sensor system). However not all of the slots are used, the wheel nodes use 9 of the slots, the master node uses 7 of the slots, and the environmental node uses both of its 2 slots. Keep in mind that each node has its own 11 slots (except the environmental node) and all nodes execute their code concurrently. There is a function called timer_tick(int microsec) that generates an interrupt with a period time measured in microseconds. The interrupt handler function timerIRQ() handles the scheduling and switching of tasks. The time base in the model car is set to 1 ms (millisecond) and hence everything is counted in whole ms. The RTOS has a loop time of 32 ms and in this time each task slot (from 1 to 11 in the case of wheel node or master node, 1 to 2 for the environmental node) is entered once. A task slot is entered regardless if it contains anything to execute or not, if not the system is just idle until the time window ends. At model car startup, after the initialization is done, this RTOS task-loop is entered and repeated indefinitely until the car is turned off or some error occurs (note that an error may stop execution, but it is not certain, in fact most of the time the RTOS just keeps running). Below the schedules for the various node types are shown. They could all be put into one schedule

(as in the C code files) but this makes it messier, so to facilitate overview schedules are given for each node type, even though this creates some repetition. The execution of one RTOS cycle begins at time 0 ms and ends at time 32 ms, the count is then restarted.

5.2.2.1 RTOS Schedule for Master Node

Task slot 1, execution window 0-1 ms:

- Synchronizes the RTOS with the TTCAN controller, this is done in the synchronize() function. The RTOS runs on the HCS12 controller on the G1 board and needs to be synchronized with the TTCAN controller. The TTCAN controllers in the various nodes also synchronize with each other as part of the TTCAN protocol so this way the whole distributed system gets a global time concept.
- Distributes reference values for steer, brake and speed received from the HMI via radio (or override reference values triggered by the Locust system, which is described later). The reference values are saved in the TTCAN memory in the call of save_data_HMI() in this task slot but are not put out on the TTCAN bus until the TTCAN controller triggers the appropriate message object to send. The TTCAN message trigger times are shown later in this section. So even though the reference values are written to memory in this time slot they are actually sent in a later time slot in the same RTOS cycle.

Task slot 2, execution window 1-4 ms: Idle.

Task slot 3, execution window 4-7 ms:

- Runs a fault detection function. These are of no concern for our project so they are not described further here. Detailed information about the fault detection functionality of the model car can be found in [13].

Task slot 4, execution window 7-9 ms:

- Prepares and sends data to the HMI via the radio link. This is done in the functions radio_prepare_data() and radio_send() respectively.

Task slot 5, execution window 9-12 ms:

- Runs some fault detection functions that are of no concern in this project. See [13] for information about the fault detection functionality.

Task slot 6, execution window 12-16 ms:

- Receives data sent from the environmental node over TTCAN. This is done in the get_data_ENV() function. This data contains threat information from the Locust sensor unit.

Task slot 7, execution window 16-21 ms: Idle.

Task slot 8, execution window 21-23 ms: Idle.

Task slot 9, execution window 23-26 ms: Idle.

Task slot 10, execution window 26-28 ms:

- Used for extra debug printouts that do not fit in the execution windows of the tasks creating them. The printouts use serial communication that is very slow compared to the execution windows of the RTOS, so in order not to breach a deadline some printouts may need to be done here.

Task slot 11, execution window 28-32 ms: Idle.

5.2.2.2 RTOS Schedule for Wheel Nodes

Task slot 1, execution window 0-1 ms:

- Synchronizes the RTOS with the TTCAN controller, this is the same as for the master node described above.
- Distributes output data from the local control algorithms of the wheel node. This is sent to all the other nodes so that they can be aware of one another's states. This is done in the function save_data_LC_own().

Task slot 2, execution window 1-4 ms:

- Gets sensor feedback data from the other wheel nodes, this is used in the fault tolerance functions. This is done in the function get_data_sensor_OWN().
- Receives speed, steer and brake reference signals from the master node, these are used in the local control algorithms. This is done in the function get_data_HMI().
- Runs a fault detection function, see [13] for details.
- Distributes failure flags that are needed by the fault tolerance functions. This is done in the function save_data_FD_own_node_and_PN().

Task slot 3, execution window 4-7 ms:

- Runs a fault detection function, see [13] for details.

Task slot 4, execution window 7-9 ms: Idle.

Task slot 5, execution window 9-12 ms:

- Gets failure flags from the other nodes, to be used in the fault tolerance functions. This is done in the function get_data_FD_global().
- Runs some fault detection functions, see [13] for details.

Task slot 6, execution window 12-16 ms:

- Calculates output signals for its partner node, in case it has stopped working these control signals will be used instead. This is done in the function local_control_partner_node(). The two front wheel nodes are each other's partner node and the same for the two rear wheel nodes.

Task slot 7, execution window 16-21 ms:

- Runs the global control algorithms for all nodes, not just the own node. This is done in the function perform_global_control().

Task slot 8, execution window 21-23 ms:

- Reads feedback data from the wheel angle, speed and brake sensors needed for the local control algorithms. This is done in the function read_data_from_sensors().

Task slot 9, execution window 23-26 ms:

- Runs a voting fault detection function.

Task slot 10, execution window 26-28 ms:
- Used for extra debug printouts, same as for the master node described above.

Task slot 11, execution window 28-32 ms:
- Runs the local control algorithms for the own node using the output of the global control. This is done in the function local_control_own_node().

5.2.2.3 RTOS Schedule for Environmental Node

Task slot 1, execution window 0-1 ms:
- Synchronizes the RTOS with the TTCAN controller, this is the same as for the master node described above.
- Sends threat data received from the Locust sensor system to the master node via the TTCAN bus. This is done in the function save_data_LC_own(). Note that the data is only saved in the TTCAN RAM and not sent over the TTCAN bus until the TTCAN controller triggers the sending of the message object where the data is stored (which is message object 6).

Task slot 2, execution window 1-32 ms:
- Gets and prints reference data sent by the master node, used for debug purposes. This is done in the function get_data_HMI(). These debug printouts are relatively long and are therefore given a large execution window.

There is no explicit task in the above schedule to handle incoming radio communication from the HMI to the model car, only the other way around. That is because the schedule only handles functions that are suitable to run periodically, which works fine for communication from the model car to the HMI, but not the other way around since the model car and HMI do not share a global time. Incoming radio transmissions are handled directly by an interrupt routine, radio_receive_data().

Another thing that is worth noticing is that if a task breaches its deadline the system does not take any action, except printing a debug message to notify that there has been a deadline breach and in which node. So in effect all of the tasks are implemented as soft deadlines, but some of them could arguably be considered important enough to be regarded as hard deadlines. The execution windows are large enough for all tasks to finish in time, with some buffer time, so if a deadline is breached it is a clear sign that something is not working properly and therefore restarting might be a good option, since there is currently no other way to get the system back on track. A simple solution would be to send out restart messages to all nodes once a deadline is breached, this would of course be a rather crude method but it would probably enhance the model car's stability, which is not very good. Every time something goes wrong the car has to be restarted manually. Another approach would be to restart only the node that breached its deadline, and in the meantime rely on the fault tolerance functionality. None of these have been tested though since it is not part of this thesis project. The stability problem of the model car is something that cost us a lot of time, both in software debugging and searching for faults in the hardware. This is something that we feel should be addressed in the future before further expansion of the model car.

Task slot 10 in the schedule above is reserved for extra debug printouts. The reason for this is that all the printouts use RS-232 serial communication with a PC with a baud rate of 57600. This means that only a limited number of characters can be printed within the scope of 5 ms, which is the largest execution window in the schedule. As can be seen in the schedule above most tasks have even shorter execution windows, typically around 2-3 ms. Each character

sent is ten bits in RS-232 (8 data bits, 1 start bit and 1 stop bit is used in the model car), which means that about 5,76 characters/ms can be printed. In other words, a 2-3 ms window is limited to a maximum printout length of 11-17 characters, disregarding the overhead time for the CPU to use the serial communication module. It is easy to imagine that a complex system like the distributed system in the model car may need longer debug printouts than 15 characters.

5.2.2.4 TTCAN Schedule

As mentioned before, when messages are sent over the TTCAN bus is controlled completely by the TTCAN controller in each node. The only thing the get and send functions in the RTOS do is that they save the data in a certain message object. That object in turn is coupled with a time trigger in the TTCAN controller on the RTCOM board that determines at what time in the TTCAN matrix cycle that message object is to be sent over the bus. Receiving of messages (i.e. reading on the TTCAN bus) is also controlled by triggers in the TTCAN controller. This is worth thinking about when scheduling the sending and receiving of messages to avoid using data that is older than it has to be. In the worst case bad scheduling can mean that data is used up to just short of 64 ms (2 times the 32 ms RTOS loop time) after it is measured (or otherwise received), not taking into account any other delays that might occur before the data arrives at the sending node. The model car uses TTCAN level 2, with 800 kbit/s. The Network Time Unit (NTU) is 1.25 μs and the matrix cycle consists of 32 basic cycles, each with the length of 800 NTU:s, or 1 ms. In other words every TTCAN matrix cycle takes 32 ms to complete and corresponds to one RTOS task-loop (remember that these two are synchronized as well, so they start and end at the ‘same’ time). As mentioned before, during one matrix cycle (or RTOS task-loop if you will) each message object is transmitted once over the TTCAN bus, and is then readable by any of the other nodes. This also means that all the control algorithms have a period time of 32 ms. Each basic cycle is started with a reference message at time 0, measured in NTU:s, and then up to three other messages can be sent in that basic cycle. These messages are sent at time 200, 400 or 600 NTU:s. Each TTCAN message holds 8 bytes of data, and the payload size cannot be changed. The TTCAN message schedule used in the model car is shown in figure 31 below.

Time (ms)	Function running by RTOS	TTCAN at 200	TTCAN at 400	TTCAN at 600
0	Sync. TTCAN and RTOS + error check			
1	Fault detection own node	FL (2)	FR (3)	RL (4)
2		RR (5)	Env.(Locust) (6)	
3		HMI/Com. (7)		
4	Fault detection partner node			
5		FL (8)	FR (9)	RL (10)
6		RR (11)		
7	Reserved for future functions		FL (12)	FR (13)
8		RL (14)	RR (15)	
9	Fault detection node, radio			
10				
11				
12	Local control partner node			
13	Get Locust threat data (master node)			
14		FL (16)	FR (17)	RL (18)
15		RR (19)		
16	Global control			
17				
18				
19		FL (20)	FR (21)	RL (22)
20		RR (23)		
21	Read data from sensors		FL (24)	FR (25)
22		RL (26)	RR (27)	
23	Voting			
24				
25				
26	Reserved for future functions		FL (28)	FR (29)
27		RL (30)	RR (31)	
28	Local control own node			
29				
30				
31				

Figure 31: The TTCAN message schedule in the model car. To the left in the table the functions running in the RTOS at the same time are shown. These functions are the union of all functions running on all nodes, so keep in mind that not all the nodes run all the functions.

One thing to note is that the handling of incoming Locust data is not shown in the schedule. This handling is interrupt driven and is not subject to the periodization like the other tasks. However the distribution of Locust data is done only once per matrix cycle (RTOS task-loop), i.e. at best once every 32 ms.

5.2.3 Communication Interface to the Locust Sensor System

The Locust sensor system outputs signals using the RS-232 serial communication standard. These signals are not sent in a predictable time-triggered manner like the TTCAN messages, but can arrive at any time with no predefined time interval between two consecutive messages. Since the model car has six nodes, where one node is specifically designated as environmental node, we decided to use this node for the communication interface between the serial connection to the Locust sensor system and the TTCAN network in the model car. When this thesis project started, it was still unclear exactly what kind of information would be received from the Locust sensor system. However the internal structure of the model car's nodes left us basically with two design options. We could read the data from the Locust sensor system and run the CA algorithms in the same node and then send the resulting reference values over the TTCAN network. Another way would be to simply receive the Locust data in the environmental node and retransmit it over the TTCAN network to either the wheel nodes directly to run the CA algorithms there, or to the master node and let the CA algorithms run there. At first we wanted to run the CA algorithms directly in the environmental node because that node was originally not doing anything. This option was later discarded because we thought that the bottleneck of this system was not the control calculations running on the microprocessor, but rather the speed (or lack of speed) of the serial communication.

After some testing we came to the conclusion that the serial communication of the G1 board was not sufficiently reliable for baud rates over 38400. Note that for the debug printouts mentioned earlier the baud rate is set to 57600, but they are also sometimes be scrambled, which is acceptable for a debug printout, but hardly for a control system. A baud rate of 38400 translates to 3,84 chars/ms (10 bits per char, 1 start bit, 1 stop bit, and 8 data bits are used), not counting any overheads in the communication between the microprocessor core and the serial communication module. Since one outer control loop has to finish within 32 ms, as described before, this means that a maximum of 122 chars could be received during one outer control loop, assuming that they are all sent immediately after one another. Then we also have to take into account that the control calculations will take some time. Because the length of the Locust data transmissions was unknown at the start of the project we decided that it is better to let the environmental node only handle the receiving of Locust data, and then transmit it over the TTCAN network to another node. The next question that arose was which node this should be. From a control point of view it would be best to send the Locust data directly to the wheel nodes themselves, and incorporate the CA algorithms into the local control algorithm there. This is because reference values are only sent once every matrix cycle over the TTCAN bus, which means that if the data is transmitted from the environmental node directly to the wheel nodes there can be a maximum delay of 32 ms. If instead the data is transmitted to the master node first and later the reference values calculated in the control algorithm there is sent to the wheel nodes, the delay can become one matrix cycle longer, for a maximum delay of 64 ms. From a control point of view we would like the delay to be as short as possible to improve system performance and robustness. However we opted to put the decision algorithm in the master node, for two reasons. The first reason is that the system architecture becomes more obvious and the code easier to read and understand, if all global control is put into a single node that just distributes reference values to the wheel nodes. The second, and for us most important reason, is that we want to avoid tampering with the local control code in the wheel nodes. As mentioned before we did not have any documentation at all at the start of the project, and had to read the C code ourselves to get an idea of how the model car software worked. We felt that the risk of introducing errors in the local control code was too high given the relatively short scope of the project, so for practical reasons we decided that it would be safest to put the CA algorithms in the master node, where the existing code was much less complex.

Because the Locust data does not arrive in a predictable manner, it has to be handled in an interrupt routine. We incorporated this into the existing interrupt handler for the radio receiver (the function radio_receive_data(void) in radio_drv.c), because as mentioned in the beginning of the model car software section, the code for all node types are put into the same files and macros are used throughout the code to distinguish which part to compile for which node type. So we thought it was best to continue this strategy and add another such macro condition in the existing radio interrupt handler (the radio also uses the same serial communication module, but the radio code is only compiled for the master node). For this to work correctly we had to make some additions to the serial communication drivers, to be able to handle interrupts on both serial ports (denoted SCI0 and SCI1 in the code).

Originally when the thesis project started we were supposed to get measurements of position and velocity of the detected possible threats. In the first agreement on the communication protocol to be used between the Locust sensor system and the model car, there were indeed such message types, and there were also supposed to be estimates of the kind of object that had been sensed. The specified objects were pedestrian, bicycle/motorcycle, regular car/medium sized vehicle, and truck/large sized vehicle. However, delivery of the Locust sensor system was delayed about five months, and in the end the only output was a threat parameter, which is basically a warning flag telling when something dangerous enough has been perceived. So the final protocol only contains one message type, the threat level, of the following format: a header tag ‘t’ (the ascii char t), followed by the threat level (a number between 0 and 5, sent as ascii chars rather than actual numbers), terminated by new line and carriage return chars. I.e. a typical threat message looks like “t5\n”, where 5 is the threat level (the highest danger threat) and \n represents new line and carriage return. These messages are parsed in the interrupt handler using some global state variables and when a threat message has been fully received it is stored in the TTCAN memory using the function save_data_LC_own(void) in get_and_save.c. The message is then sent when the TTCAN trigger for the corresponding message is fired. We made a new message object for the environmental node and placed it in basic cycle 2 in the TTCAN matrix cycle, at trigger time 400 NTU:s. The environmental node messages use TTCAN message object number 6.

Because the stability of the TTCAN network was not very good we made some changes to the TTCAN drivers, by adding some extra functions. We added a function called fatalError() that checks for CfE ApW and WTr interrupts as specified in the TTCAN User’s manual [4]. A CfE interrupt indicates that there is an error in the trigger list. An ApW interrupt indicates that the watchdog was not served in time. A WTr interrupt indicates that a reference message in the TTCAN matrix cycle has been missed for some reason (indicating that there might be a problem with the timing). We also enabled the watchdog timer in the TTCAN drivers and made a function serveApplicationWatchdog(), which reloads the TTCAN watchdog timer, and put that into the RTOS task-loop. Note that this function is called just before a new task-cycle is entered, so it is not designated a task number in the RTOS schedule described earlier (meaning it does not reside in the file rtos.c, but in main.c). After this change the stability of the TTCAN performance, in particular the startup of the model car, improved. Another TTCAN issue that appeared was a timing problem when reading and writing to the TTCAN message RAM. According to other people with experience of working with the G1 boards TTCAN stability can often be improved by using double read cycles, which means that first a dummy read is done, before the correct value can be retrieved. After extensive testing we came to the conclusion that the TTCAN network stability (or lack thereof) can be improved further by using even more such dummy reads. We had huge problems with reading and writing speed messages over the TTCAN bus, and managed to solve this by using in total six read cycles (five dummy reads) when retrieving data from the corresponding TTCAN

memory location. For the writing of data to the speed locations in the TTCAN memory we introduced a dummy double read before finally writing the message to the RAM. For all other memory locations using quadruple read cycles (three dummy reads) appears to give the best results for retrieving data, no dummy reads are necessary for writing data to these locations. All this is in the C code file TTCAN_messages.c. These modifications were done purely through trial and error, which was very time consuming, and hardly part of the original project plan. Because of this we did not have time to further improve the stability of the TTCAN communication, since the above mentioned modifications are enough to create sufficiently stable communication for our purposes in this thesis project. However it should be possible to analyze these issues in more detail, and perhaps come up with better solutions to improve the stability of the model car's TTCAN communication for future projects.

The stability of the serial communication module interrupts, that are used for both radio communication and communication with the Locust sensor system, also had stability problems. Sometimes we would lose radio contact for no apparent reason, and the Locust communication suffered the same problems. We found that for some reason the serial communication interrupts sometimes just did not work after system startup. From our experiences of timing problems with the TTCAN controller that could be solved by multiple register reads (in the above paragraph called RAM, since all registers are memory mapped, they appear to the programmer to be locations in the G1 microcontroller's RAM) we decided to try a similar approach to this issue. After lots of testing, we found by trial and error, that this problem can be solved by doing a dummy read of the serial communication module data registers after system initialization. This only needs to be done at model car startup, and it solves the problem completely, even though we are still unsure exactly why the problem occurs.

In the protocol that we agreed on with the developers of the Locust sensor system, the model car also needs to send back measurement values of the actual speed and wheel angle. The format for this is simply a header char to denote the type of data ('a' for wheel angle and 's' for speed), followed by the measured value, and finally a new line (new line char followed by carriage return char). However there is a problem with noise in the data coming from the TTCAN network (this is a problem only in the environmental node). In the original model car code there were already filters applied to the measurements in the various wheel nodes, before sending it over the TTCAN network. So we assume that most of this noise is related to the TTCAN problems mentioned earlier. The modifications we made to the TTCAN drivers mentioned before are however not enough to reduce the noise in the output data to acceptable levels. Instead we apply a filter to try and reduce the noise. What we do is to take the weighted average over several samples, to get rid of high frequency variations. We assume that the noise in the TTCAN communication is probably of higher frequency than the output from the relatively sluggish speed and wheel angle actuators of the model car. We recorded speed and wheel angle data read from the TTCAN network and tried various values for the filter constants using Matlab, until we were satisfied with the output of the filter. The resulting filter is a FIR-filter that takes the weighted average over the 8 latest samples. The filter equation is given below, where y[k] is the output to the Locust sensor system at discrete time k and u[k-n] is the value read from the TTCAN bus n samples earlier. The same filter is used for both wheel angle and speed filtering. The sample time is the same as the RTOS loop time, 32 ms.

Environmental node TTCAN filter equation:
$y[k] = (1.5/8)u[k] +(1.5/8)u[k-1] +(1.25/8)u[k-2] +(1.25/8)u[k-3] +(0.75/8)u[k-4] +(0.75/8)u[k-5] +(0.5/8)u[k-6] +(0.5/8)u[k-7]$

The idea is that the latest samples should be weighted higher and the earlier samples lower and lower. The values of the filter coefficients above are a trade off between noise suppression and the ability to follow changes in the speed and wheel angle. The wheel angle changes faster than the speed, because the model car is rather heavy (about 40 kg) so that is the rate of change the filter has to be able to follow. Turning the wheels from center position to maximum left/right position can be done in approximately 0.5 seconds at the fastest turning rate of the wheel angle actuator. With this turning speed the filter can follow the real angle values with a delay of about 0.1 seconds, with a standard deviation from the real values of about 10-15% of the real value. We think this is an acceptable trade off since the TTCAN measurements are very noisy, and we cannot accept the filter being too slow to follow changes in the signals. Figure 32 and figure 33 shows plots of the original wheel angle signals and the filtered signals when the wheels are in the center position and maximum left position respectively. The center position corresponds to an average value of 4500 and the maximum left position to an average value of 1100. These plots show how noisy the signals read from the TTCAN bus are. Figure 34 shows plots of the original wheel angle signals and filtered signals when the wheels are turned at maximum turning rate from the center position to the maximum right position (the first plot), and when the wheels are turned at maximum turning rate from the center position to the maximum left position and then immediately back through the center position to the maximum right position (the second plot). The maximum right position corresponds to an average value of 7900. Note that in these turning plots the real signals are simulated with no noise, to make it easier to see how the filter is able to follow the changes in the signals.

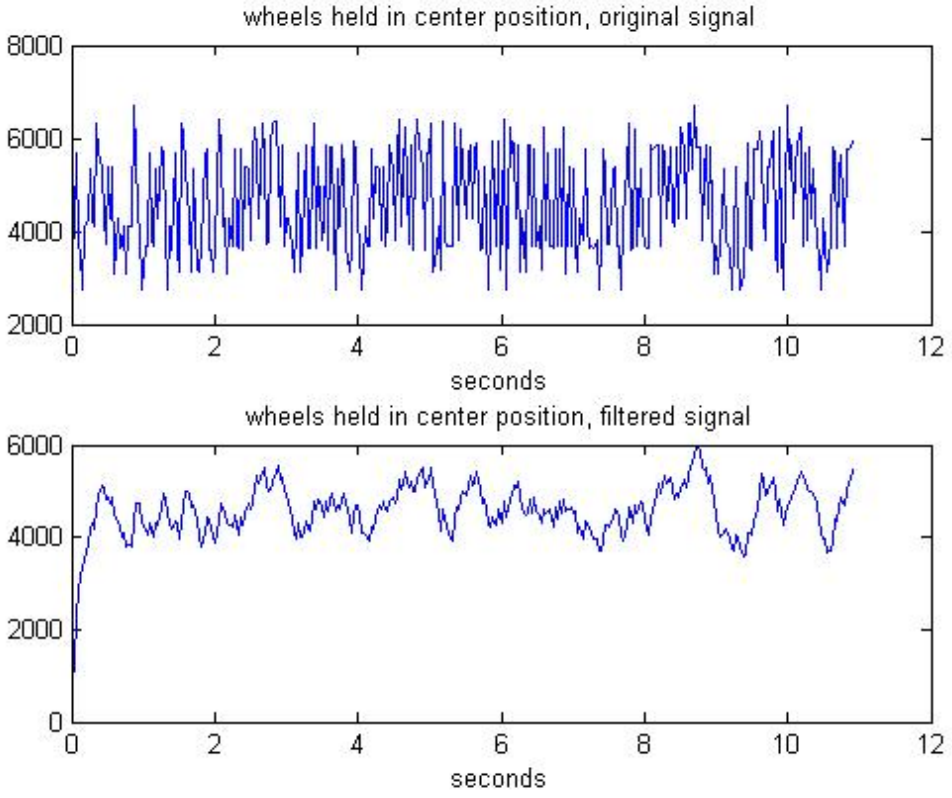


Figure 32: Plots of the wheels being held still in the center position, original and filtered signal.

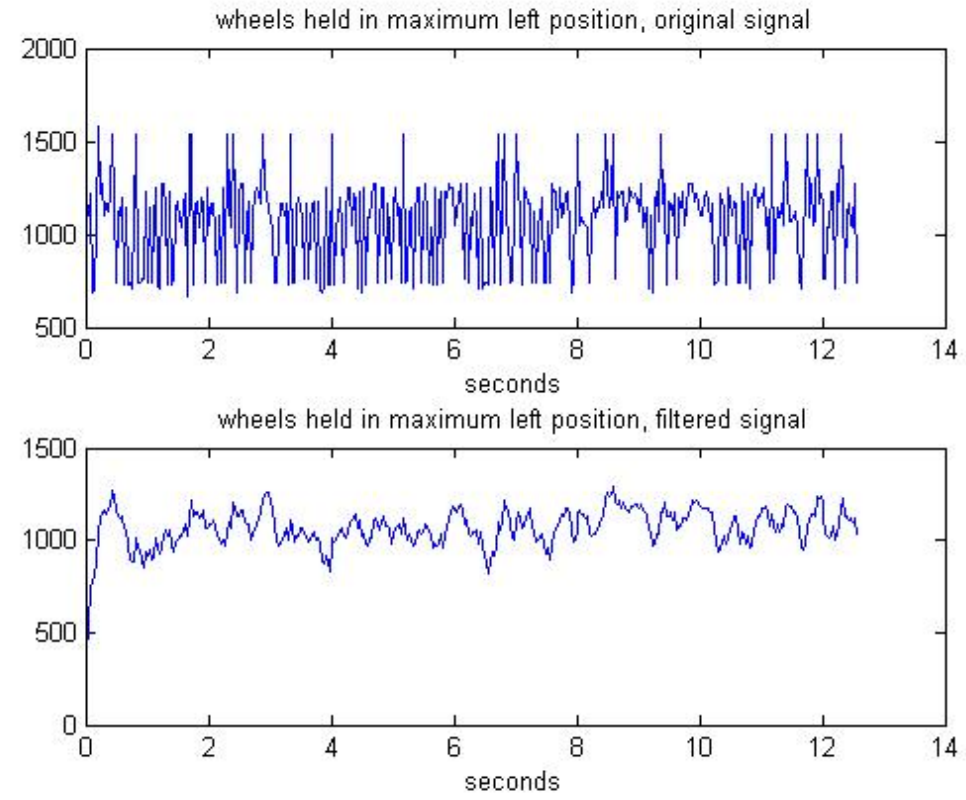


Figure 33: Plots of the wheels being held still in the maximum left position, original and filtered signal.

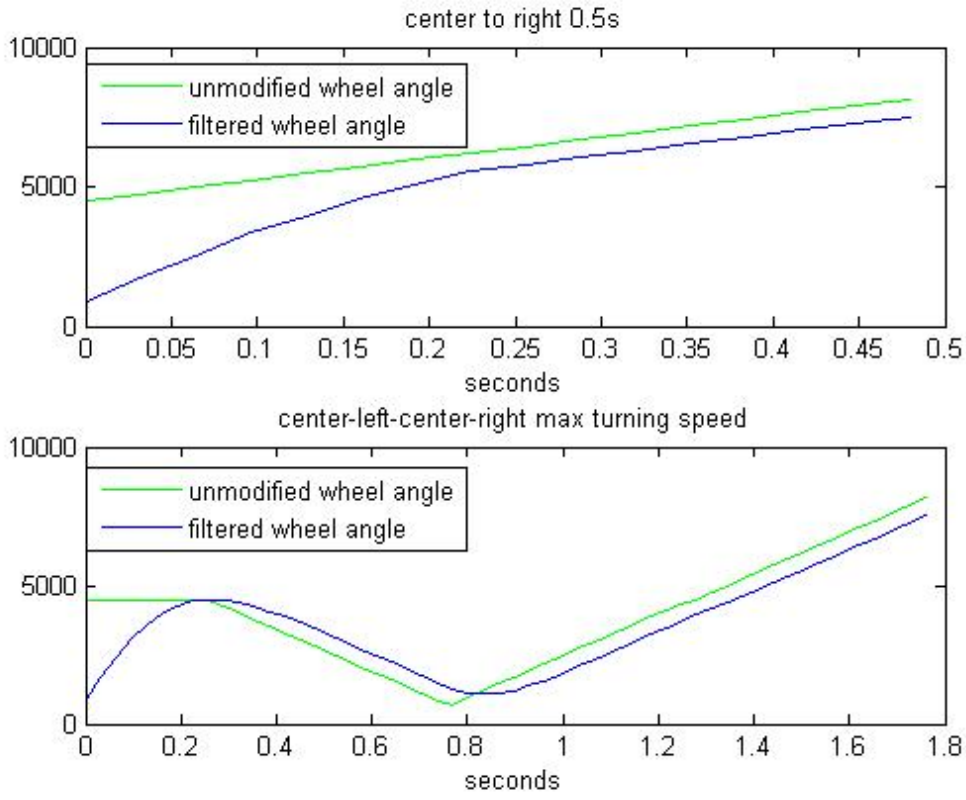


Figure 34: Plots of the wheels being turned, from center to maximum right (first plot), and from center to maximum left and then back through the center and to the maximum right position (second plot).

5.2.4 Model Car Collision Avoidance Strategy

Originally we planned to implement a tracking algorithm with the techniques described in chapter 4 and use it to test different decision functions like the ones described in chapter 4. The whole idea with the model car was to test some actual collision avoidance maneuvers (breaking, steering, or a combination of both) since this is too risky to do with a full size car, given that the scope of our project was only 20 weeks. However this could never be done, for two reasons. When we started the project it was assumed that the model car was fully functional. It was clear after the first month of the project, that this was not the case. As described in the previous section the TTCAN network in the model car had huge stability problems. Also there was no documentation of the model car code, which we found out only when it was time to start programming. This meant that we had to spend a lot of time to read the existing code just to understand how the model car works, and later also to fix the most important TTCAN issues. The Locust project was however delayed, so there would still have been time to implement at least some of the simplest CA methods described in chapter 4. When the Locust sensor system finally arrived (the first time, which was in April 2006), it did not meet the agreed on signal specification, in which it was supposed to provide at least position measurements (and according to contact we had with its developers in Hungary earlier probably also a rough velocity measurement). As described before the only output was a warning parameter, so the only possible solution for us was to just react to that and apply breaking accordingly, since we had no information about a threat's position relative to the host vehicle. This is done in the following way:

- **In function radio_receive_data(void), in file radio_drv:**
  This executes in the environmental node. When a threat message has been received the global variable threat_level will be assigned the threat value, and the global variable threat_updated is used to signal that a new threat has arrived.

- **In function save_data_LC_own(void), in file get_and_save.c:**
  This executes in the environmental node. If the threat_updated variable has been set, the value of the threat_level variable is written to the TTCAN message RAM for message object 6, to be sent over the TTCAN bus next time the send trigger for that message object fires. Then the threat_updated and threat_level variables are reset.

- **In function get_data_ENV(void), in file get_and_save.c:**
  This executes in the master node. The TTCAN RAM is read to obtain the data for message object 6. If the field corresponding to the threat level is at the highest value (which is 5) a global variable take_over is set, to signal that control should be taken away from the driver and given to the emergency breaking.

- **In function timerIRQ(void), in file RTOS.c:**
  This executes in the master node. A counter variable take_over_duration is used to keep track of how long control has been in the hand of the emergency breaking. The take over time is set to 2 seconds, since the model car has very efficient brakes in comparison to its weight and speed properties, and this is judged enough time to bring the car to a complete stop is virtually any situation. As long as the take_over_duration variable has a value which corresponds to a time lower than 2 seconds, the take_over variable is kept set, and then when the take over time has reached 2 seconds the take_over variable is reset, which gives control of the model car back to the driver.

- **In function save_data_HMI(void), in file get_and_save.c:**
  For as long as the take_over variable is set (which as mentioned above is 2 seconds) near maximum breaking power is applied. The reason not to apply maximum breaking power is because the break reference messages are sent over the TTCAN bus to the wheel node actuators, and the TTCAN bus sometimes has overflow problems, meaning that a value that is supposed to be the maximum value instead becomes a very small value, which would be unfortunate in the case of an emergency situation.

6 XC90 Warning HMI

6.1 Overview

As can be seen in the problem specification for this thesis work, a warning HMI for a full scale car, a Volvo XC90, was also to be developed. We did this in cooperation with one of our Hungarian partners in the Locust project, Istvan Petras, who was responsible for integrating the Locust sensor chip into a compact vision system that they call the Bi-i, as described in chapter 2.

As a communication bridge between the internal distributed computing environment in the XC90 and the Locust sensor system, a G1 board is used. The XC90 engine bay uses a CAN bus to connect its nodes to each other (actually there are two CAN networks in the car, and also several other types of networks, but all the engine bay functions use the CAN network we refer to here), and the G1 board is equipped with a CAN module which is used to snoop information sent between the nodes in the XC90. The information retrieved in this way is the speed, wheel angle, and yaw rate of the car, which is needed by the Locust sensor system. The input and output from the Locust sensor system is sent using the RS-232 serial communication standard. So in short, the G1 board is used to read CAN messages from the XC90 and translate these to RS-232 messages and forward them to the Locust sensor system. The G1 board is also used to generate the visible and audible warning signals for the HMI. In figure 35 below is a schematic of the XC90 Warning HMI.

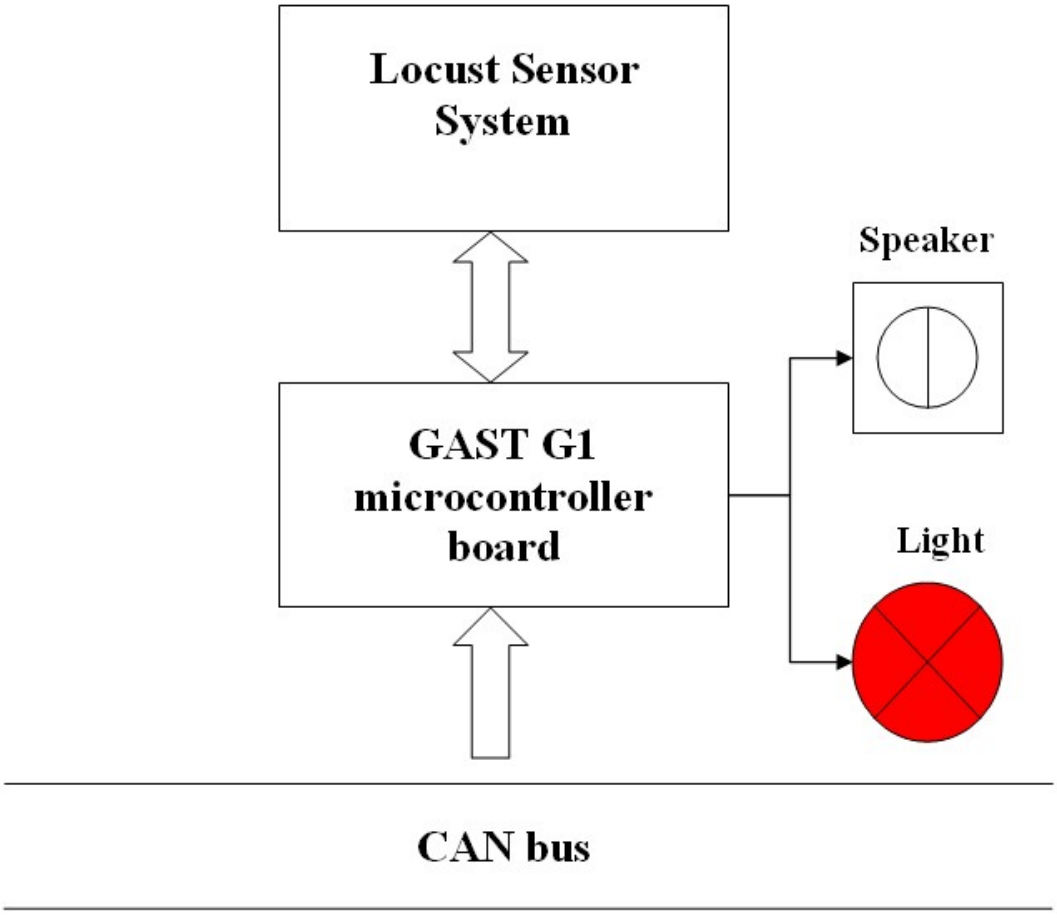


Figure 35: Schematic of the XC90 Warning HMI.

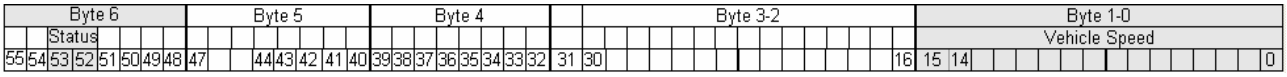
The CAN messages are sent using high speed CAN which can be handled by CAN module 0 and CAN module 4 on the G1 board. In the HMI code we use CAN module 0 to listen to the internal messages of the XC90. The messages relevant for this project are the ones called VehicleSpeedABS, YawRateActual, and SteeringAngle. Below is a description of what these signals mean and also their bit format. In total a message is 8 bytes long but only 7 bytes are shown in the message format below. The 8th byte that is not displayed here holds the ID-information to determine which type of message the currently read message is. This information has been left out in this paper for security reasons. Also note that the signal descriptions below are not complete, also for security reasons.

VehicleSpeedABS

Signal definition

Description:
Purpose:
The purpose of the signal is to supply vehicle speed information to other electronic systems. Vehicle speed is defined as the average speed of the two front driven wheels, together with information from CEM about the tyre circumference.
Requirements on transmitter:
The signal VehicleSpeedABS shall be available once on power-up, 150 ms ± 150 ms after power-up and then according to the update period. The signal shall be transmitted according to the signal database specification. If an ABS sensor signal input is disturbed by abnormally high signal frequency input then the VehicleSpeedABS signal shall not exceed the maximum allowed speed value and it shall return to correct speed value after the disturbance has vanished.
Requirements on receiver:
The receiver shall filter the VehicleSpeedABS signal information to suit the individual application.
Special requirement:
The signal VehicleSpeedABS is the average speed of the two front driven wheels during the previous two calculation loops (14 ms) before signal transmission.
The signal shall be grouped together with the signal VehicleSpeedABSQF. The signal VehicleSpeedABS status shall be reflected in the signals VehicleSpeedABSQF and MechSensorCheckStat. The accuracy of the signal is ± 2% of the value related to tolerances in the tyre dimensions, larger deviation of accuracy will occur when driving with a smaller spare wheel (tampa spare).
OBSERVE: If an application is using the VehicleSpeedABS signal information for 'long time' average vehicle speed calculation but not reading and using every VehicleSpeedABS signal information then real vehicle speed information will be lost. The result may in special road situations lead to a faulty average vehicle speed calculation done by the application (sampling in phase with road disturbances). The signal range is dependent on the tyre adaptation parameter and may be different in different vehicles.

Can message layout:



General Remarks: -
Revision: -

Group:	VehicleSpe	Update Bit:	Yes
Signal Type:	Unsigned	Information Type:	State_Info
Size:	16	Range:	0..400
Unit:	kph	Resolution:	0.01
Periodicity:	Periodic	Accuracy:	-

Max Upd Period:14 ms
Min Upd Interval:-

Current implementation in source ECU: 14 ms.
Sporadic signals only!

Coding:

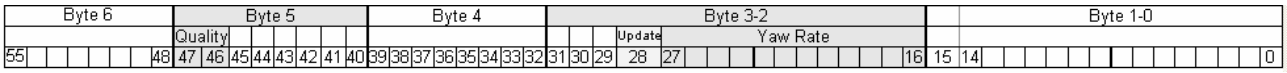
```
signal VehicleSpeedABS_H
{
        size = 16 ;
        type = unsigned ;
        offset = 0 ;
        update bit = 49 ;
}
```

YawRateActual

Signal definition

Description:
The purpose of the signal is to distribute the requested yaw-velocity of the vehicle.

Can message layout:



General Remarks: -
Revision: -

Group:	DSTCGrp	Update Bit:	Yes
Signal Type:	Signed	Information Type:	State_Info
Size:	12	Range:	-75..75
Unit:	deg/s	Resolution:	0.04
Periodicity:	Periodic	Accuracy:	-

Max Upd Period:	14 ms	Current implementation in source ECU: Not specified
Min Upd Interval:	-	Sporadic signals only!

Coding:

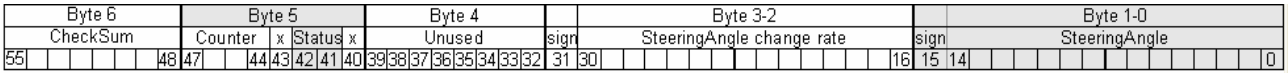
```
signal YawRateActual_H
{
        size = 12 ;
        type = signed ;
        offset = 16 ;
        update bit = 28 ;
}
```

SteeringAngle

Signal definition

Description:
Current angle of the steering wheel measured by Steering Angle Sensor. Signal is unsigned (sign bit is a separate signal).

Can message layout:



General Remarks: -
Revision: -

Group:	SteeringAn	Update Bit:	No
Signal Type:	Unsigned	Information Type:	State_Info
Size:	15	Range:	0..1433.6
Unit:	degrees	Resolution:	0.04375
Periodicity:	Periodic	Accuracy:	1.5

Max Upd Period:	10 ms	Current implementation in source ECU: 10 ms.
Min Upd Interval:	-	Sporadic signals only!

Coding:

```
signal SteeringAngle_H
{
        size = 15 ;
        type = unsigned ;
        offset = 0 ;
}
```

The CAN bus is polled for messages, and when a message is received the message ID is checked (bit stuffing is used for the XC90 CAN messages, so first that is dealt with) and in case it is any of the above messages bit masks are used to filter out the interesting information bits from the message. The most interesting message parts are marked with grey background in the message layout above. The data is then forwarded as raw hexadecimal numbers on the RS-232 connection to the Locust sensor system. Note that messages are only sent to the Locust sensor system if something has previously been received from it. So the filtered out CAN data is stored in the G1 board until an RS-232 message is received from the Locust sensor system, and then the speed, wheel angle, and yaw rate messages are sent once over the RS-232 connection. Since the CAN bus is polled we have to use interrupts for any other input, or else we cannot guarantee that no messages are dropped. In the code the function Generic_Handler() in file main_locust.c takes care of interrupts coming from the serial communication module. The interrupt vector is set in file vector_table_locust.c and the serial interrupt for the module we use is called Generic_VSCI0 in the table. Most of the interesting code is in file main_locust.c. The serial communication interrupt routine sends a message to the main CAN polling loop that the Locust sensor system is ready to receive a new batch of data. This mechanism has to be used for two reasons, first of all because the Locust sensor system does not check its external I/O buffers regularly, and second because its RS-232 buffer is limited and the Locust sensor system might crash if the buffer is overrun. In other words it is not a good idea to output translated CAN messages on the RS-232 line regularly, since that might overrun the Locust sensor system RS-232 receive buffer, so the best way is to simply let the Locust sensor system signal the G1 board when it is ready for a new batch of data. Another issue is that the timing in the Locust sensor system is rather sensitive, so it cannot be allowed to wait for too long without getting an answer from the G1 board, after it has signalled for a new batch of translated CAN data to be sent. This time window appears to be about 20 ms from the time the Locust sensor system starts sending to the time it finishes receiving data on the RS-232 connection. As for the model car the baud rate for the RS-232 communication has to be set to 38400 to ensure robust performance. Each output message from the G1 board is 15 chars long, and the output from the Locust sensor system is 3 chars long, so a total of 18 chars have to be sent each communication cycle, which takes about 5 ms of total sending time, assuming there are no other delays in the sending process. This means that we have to take care not to waste too much time in the G1 board data processing, since we want to have a fairly large time buffer to keep the Locust sensor system from crashing every now and then. Keep in mind that even though 15 ms is quite a lot of computing time, the Locust sensor system has to be up and running without crashing for at least 20 minutes to be able to do any successful demonstration. In this time there is a potentially huge number of messages sent on the RS-232 line, and it is enough that one such time window is breached for the Locust sensor system to crash, so we want the time buffer to be as large as possible. Of course for commercial applications an uptime of 20 minutes is ridiculously short, so we want to make the uptime as long as possible, meaning we want to keep the calculations in the G1 board to a minimum.

6.2 Light and Sound Warnings

To signal to the driver that something dangerous is about to happen we use a flashing head up display of diodes and a simple speaker to send audible warnings. The light display is placed just below the windshield in the XC90 so that the light is reflected in the windshield right in front of the driver, at about eye height. The speaker is mounted on top of the G1 board itself which is placed between the driver seat and the passenger seat, just behind the gearshift. The head up display in the XC90 demonstrator can be seen in figure 36. The G1 board in the XC90 with the mounted speaker can be seen in figure 37. Figure 38 and figure 39 show the

Locust sensor system mounted in the XC90 demonstrator, from inside and outside the car respectively.

Figure 36: The head up display in the XC90.

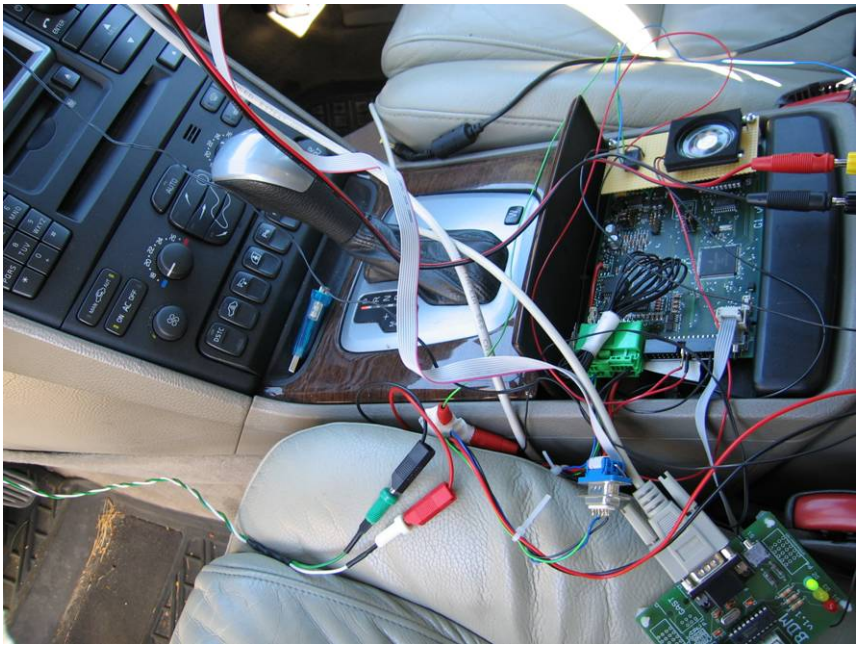


Figure 37: The G1 board in the XC90, with a speaker mounted on top.

Figure 38: The Locust sensor system mounted in the XC90, seen from the inside.

Figure 39: The Locust sensor system mounted in the XC90, seen from the outside.

When a threat signal is received in the G1 board from the Locust sensor system, light and sound warnings are generated. We use a timer interrupt called timer_IRQ() for this, just like in the model car code, that generates an interrupt every millisecond. Every time the interrupt routine is entered it checks if any message has been passed from the serial communication interrupt, that a new threat signal has been received. If so if sets some global variables to keep track of when the current threat level was received, and also to turn on the light and sound warnings. Furthermore it keeps track of the current time and whether a warning is currently in progress. The Locust sensor system also sends signals to let the G1 board know when the danger has passed, so the code in the G1 board does not have to worry about deciding when to turn off the warnings, however we put in a maximum limit of 5 seconds for any one

warning flag from the Locust sensor system. 5 seconds is a very long time and should be enough by far for any dangerous situation. In the case of a false warning being sent from the Locust sensor system, and if during the next 5 seconds it crashes, this would leave the light and sound warnings running in an infinite loop. This of course would be mighty annoying for the driver so that is why we decided to put a duration limit on the warnings. The light warning is set to use two blink frequencies, one for threat level 4 and one for threat level 5. Threat level 4 is given a blink frequency of 5 Hz, and threat level 5 is given a blink frequency of 10 Hz. Other lower threat levels are not reported to the driver by the XC90 HMI, since they are way too sensitive and would be considered false alarms by most people. Lower threat levels along with some other data can be displayed by a special PC program developed by our Hungarian project partners. This is used when tuning the Locust sensor parameters. The light display consists of about 20 light diodes connected in series and is mounted on the panel behind the driver's wheel, just below the windshield. So when the display blinks the light is reflected in the windshield at eye level right in front of the driver. Because the maximum output of the G1 board is some 5V 10mA we have to use a relay to let the light outpin of the G1 board control the 12V available from the car.

The timer interrupt is also used to generate a time base for the sound warning. We use one of the general purpose I/O pins of the G1 board to generate square waves with a frequency of 500 Hz (sound frequency). As with the light only threat levels 4 and 5 from the Locust sensor system trigger warnings in the HMI. When a level 4 threat is received bursts of sound are sent out with a frequency of 5 Hz (we call this beep frequency), to match the blink frequency of the light for that warning level. The same goes for threat level 5, so its beep frequency was set to 10 Hz. Note that the sound frequency for all beeps is 500 Hz, this is good to keep in mind to avoid confusion when reading the C code for the XC90 HMI. Since there are two frequencies to keep track of for each threat level when generating the sound, some more global variables are needed, but other than that it is pretty much the same thing as generating the light warnings. A thing worth noticing about the square wave generated by the G1 board is that it did not sound like a square wave at all, probably due to delays when the outpins toggle from 0 to 5V and back again, together with some lucky circumstances with the components in the loudspeaker, to help and smooth out the wave form. We tried various sound frequencies and around 500 Hz it sounds more or less like a smooth sine wave (not perfect, but acceptable). With other sound frequencies the sound is very rough and machine like, much more like the typical square wave sound. Were it not for this, we would have had to use D/A converters to create a smoother wave form, but at 500 Hz the wave form is quite acceptable, and we also think 500 Hz is an acceptable note height. The sound is ok to listen to but still rather annoying, which is only suitable for a warning system.

7 Test Cases and Results for the XC90 Demonstrator

7.1 Procedure

Because the delivery of the Locust sensor system was delayed very long, we only had time to test the performance with the XC90, and not with the model car as originally planned when the thesis project started. The testing was done in two sets, the first one in May when the first version of the Locust sensor system arrived, and then a second time in September after the performance had been improved to match the requirements not met in the first testing round. We give the results of both these testing rounds here to give some understanding of the difficulties the Locust sensor system had. The results of the first testing round is an excerpt from the preliminary Locust project test report from Volvo Car Corporation [16]. The second testing round was carried out in September 2006, and has at the time of writing yet to be documented in any project paper at Volvo Car Corporation, however the test cases are a subset of the cases found in [16]. Originally there were even more test cases than presented here, but we had to reduce their number to have time to carry them out. We did the testing both in real traffic and on the test track at Volvo Car Corporation in Gothenburg. Many of the tests are designed to test the performance of the Locust sensor system in real traffic, where there are many situations that can easily trigger false alarms. So the tests are designed to test that the Locust sensor system warns in real dangerous situations, which it is supposed to, but also that it does not warn when there is a situation that would be considered safe by most drivers. All in-traffic tests were carried out in a city driving environment, in Gothenburg. No country road driving was tested.

Because we had time limited access to the Locust sensor system in September, the second testing round does not include as many test cases as the first testing round. However all the important shortcomings of the preliminary Locust system were retested, along with the most important other situations that were more or less working in the first testing round.

7.2 Summary of Test Results

Below is a summary of the test results. For ease of comparison we first present the results summary for the first testing round, for each commented case, followed by the results summary for the second testing round observed after the Locust sensor system had been enhanced and retuned to try and meet some of the most serious shortcomings that were revealed in the first testing round. The results in bold are the results from the first testing round, and the new, and in most cases better, results from the second testing round are in italic following after the first round results for each case. The results of the second testing round were reported to the EU reviewers as the performance of the Locust sensor system at the final review meeting in September 2006. The complete results for the second testing round can be found in Appendix C.

- Low objects by the side of the road, like guard rails, presented no problems at all.

Low guard rails were not reported as collision threats. There were some minor problems with higher fences along the road (about 1,5 m high) that would sometimes trigger false alarms when entering a curve and turning sharply and/or with high speed. The false alarms where of level 4 and always only one short beep.

- Overhead objects like road bridges crossing the host vehicles path presented no problems at all.

At speeds around and above 50 km/h there would sometimes be false alarms if the bridge was very low (about 1 m height margin from the host car). The false alarms were always of level 4 and only one short beep. Normal height bridges and lower speeds presented no problems.

- Tall slim objects by the side of the road, like road signs and posts, triggered false alarms during turning when the host car was about 10-15 m away. This was a huge drawback of the preliminary Locust system performance.

Tall slim objects by the side of the road would trigger false alarms in about 10% of the cases, if the objects came within 1 m of the host car during sharp and/or fast turning. This was a considerable improvement from the preliminary Locust system performance, but is still a problem that needs to be addressed.

- Road tunnels did not seem to affect the performance in any way.

There was unfortunately not enough time to retest the tunnel scenarios in the second testing round.

- In intersections cars approaching perpendicular to the host vehicle triggered false alarms when the distance between them was about 10-15 m. Whether the situation was dangerous or not seemed to have little relation to the triggered alarms. This was a huge drawback of the preliminary Locust system performance.

There were almost no false warnings, and there were warnings if the passing was very close. Generally we feel that the Locust system performance was very good in this test situation.

- Cars driving in the opposite direction in the adjacent lane did not present any problems at all, with regard to false alarms.

Cars driving in the opposite direction in the adjacent lane presented no problems at low relative speeds. However relative speeds of about 100 km/h or more would trigger false alarms about 1,5 s before passing. The threats were mostly level 5 and beeped until the threat car had passed. This was a serious flaw and something that was not a problem in the preliminary Locust system.

- Cars driving in the same direction in the adjacent lane did not present any problems at all, with regard to false alarms. Neither keeping a constant distance, approaching, nor overtaking would trigger false alarms.

There were no performance changes for this case in the second testing round.

- Approaching a threat car in the same lane did not present any problems at all, with regard to false alarms. Neither keeping a constant distance, approaching, nor

overtaking would trigger false alarms. However warnings were generally a bit late if there was indeed a collision or very close to collision (tested only with motionless balloon car). For speeds above 50 km/h the performance was better, but still the warnings were a bit late. On average we got the feeling that the warnings would be triggered at roughly the same time as the driver would have to initiate breaking, in order to just prevent a collision by very narrow margin. So the driver would have needed basically instant reaction to fully avoid a collision.

Speed 20 km/h: No warning, felt like a safe situation (stopped about 5 m away). There were level 5 warnings when stopping closer (1-2 m away). The driver was able to stop the car with some margin to avoid collision after reacting to the warning. The results were generally very satisfying at this speed.

Speed 30 km/h: Level 5 warning about 5-6 m away from collision, felt dangerous. The driver was just about able to stop the car in time to prevent a collision after reacting to the warning. The results were generally very satisfying at this speed.

Speed 40 km/h: Level 5 warning about 6-7 m away from collision, felt dangerous. The driver was just about able to stop the car in time to prevent a collision after reacting to the warning. The results were generally very satisfying at this speed.

For speeds above 50 km/h the Locust system performance decreased significantly with increasing speed, and avoiding collision from reacting to the warnings was no longer possible, but could still be of use for collision mitigation.

- Pedestrians crossing a zebra crossing in front of the host car triggered warnings about 2-3 m away from collision, regardless of speed.

Speed 5 km/h: The pedestrian was reported as a threat at about 1 m distance away from collision, and no false warnings according to our judgement.

Speed 10 km/h: The pedestrian was reported as a threat at about 1.5-2 m distance away from collision, and no false warnings according to our judgement.

In general the test results for this situation were very satisfying, no false warnings and no missed "true" warnings. Also all warnings came in time to completely avoid collision. It seemed that the speed played a much more significant role in the warning mechanism than in the preliminary Locust system testing.

- Pedestrians running out in front of the host car, coming from behind vehicles parked by the side of the road triggered warnings very late (0-1 m away) or not at all. This was a big performance flaw.

Speed 5 km/h: The pedestrian was reported as a threat at about 1 m distance away from collision, and no false warnings according to our judgement.

Speed 10 km/h: The pedestrian was reported as a threat at about 1.5-2 m distance away from collision, and no false warnings according to our judgement.

In general the test results for this situation were very satisfying, no false warnings and no missed "true" warnings. Also all warnings came in time to completely avoid collision. This was a great improvement from the preliminary Locust system, where the cars by the side of the road caused big problems.

- Pedestrians walking by the side of the road with the host car passing by would not trigger warnings at all, even if the passing was very close and felt dangerous. With pedestrians clearly standing in the way of the host car warnings were triggered, but not until about 5m away from collision, regardless of speed. The performance of the preliminary Locust system was better when there was another vehicle standing in collision course.

In the case of pedestrians walking beside the road and the host passing them by the pedestrians were reported as threats, if it was closer than about 1 m. The pedestrians were also reported as threats when they were standing in clear collision course with the host vehicle. The warnings came about 3-7 m away from collision depending on the speed (20, 30 or 40 km/h). The performance was generally pretty good for this test case. This was a significant improvement over the preliminary Locust system tests.

- In most of the test situations there were quite a few false alarms due to shadows covering the otherwise sunny road. This was most disturbing and not acceptable performance at all.

In the second testing round the problems with false warnings due to shadows covering the road were completely eliminated. However another issue that arose was that the Locust system performance varied with lighting condition. It performed best in sunny conditions and worst on very gloomy or rainy days. Warnings could come up to about 0.5 seconds later in the worst conditions, compared to the best conditions.

8 Conclusions

The main objective of this master's thesis has been to construct two demonstrators for the Locust sensor system, which is a bio-inspired visual sensor system that was developed in an EU-project called Locust. These demonstrators were then to be used to evaluate the performance of the Locust sensor system for automotive applications at Volvo Car Corporation. The first demonstrator was the model car presented in chapter 5. Because of delays in the delivery of the Locust sensor system there was however not enough time to use this demonstrator for evaluation. The other demonstrator was a Volvo XC90 with a warning system using the Locust sensor system installed in it, which was described in chapter 6. A summary of the test results was given in chapter 7. Below the results are analyzed with regard to the usability of the Locust sensor system for Volvo Car Corporation based on its overall performance in the tests. Some suggestions for improvements are also given, and finally we give our general thoughts on the handling of large international projects from our experiences within this thesis work.

8.1 Conclusions Regarding Usability of the Locust Sensor System

The test results are generally speaking very satisfying for lower speeds up to about 40 km/h. The tests also show that the Locust sensor system is very useful to detect pedestrians, especially when they are coming from the side. This is a typical scenario when there are children playing in residential areas for example, and at the moment Volvo Car Corporation is lacking such a detection system in their products. The Locust sensor system performance for detecting vehicles directly in front of the host car is also reasonably good, but there are already other systems in use today with similar performance that also work well for higher speeds, so in this regard the Locust sensor system would add nothing. The low speed performance in combination with the very successful pedestrian tests makes the Locust sensor system very useful for driver assistance in cities, residential areas and parking lot situations. The system is not yet fully developed to be used in production, but the low number of false warnings suggests that it should be possible to enhance the system further and thus make the performance good enough for use in commercial products. The only big issue that we can see is that the performance varies depending on weather conditions, with warnings coming up to half a second later in darker lighting conditions (such as rainy and/or very cloudy days compared to sunny days) which is something that would need to be solved in order to create a commercially useable product. Perhaps this can be addressed by using a better type of light sensor or higher image resolution, but we are not sure as our knowledge of image processing is very limited.

There is certainly room for improvement of the Locust sensor algorithms. For example the current version has no target tracking, and does not separate objects from each other, it simply triggers on what it perceives as the most dangerous object and disregards the rest. On the other hand this is also one of the main strengths of the Locust system since it is able to respond more quickly to threats that appear suddenly, like pedestrians emerging from the side of the host vehicle's path of driving. Today's warning detection systems in cars often use radars which need a longer time window to detect possible threats due to the relatively slow sweep speeds, and hence are not as useable for this kind of pedestrian threat detection. However we think the Locust system performance can be enhanced in situations where there are other vehicles in the direct path of the host vehicle, by employing so called sensor fusion. By making use of sensors that are not sensitive to lighting conditions, for example radars, in combination with the Locust sensor system it should be possible to reduce the performance differences caused by weather conditions described earlier. These differences are less

noticeable with the pedestrian tests, but quite large in the test scenarios with other vehicles in the host vehicle's path.

8.2 Practical Conclusions Regarding the Handling of Projects

In this thesis project we got some first hand experience in the handling of projects, and the difficulties that arise. Since it was a part of a large international project we had to deal with communication problems between the various project members, and we feel that through better communication and more clearly defined specifications on exactly what was to be delivered, by whom, and when, would have saved us a lot of time and trouble. In this case it turned out that the delivery of the Locust sensor system was delayed about 5 months, and in the end did not meet the specifications that we were counting on from the start, with regard to what output data the system could provide us with. Also we feel that the decision structure between the various partners in the Locust project was not well defined, and as a result it was hard for us to know who to turn to in order to get some clear answers to the above questions.

Another thing that we would like to stress is the importance of well documented code. The model car we worked with did not have any documentation whatsoever and we had to figure out ourselves how it worked through reading raw C code which was, even though the code was rather well commented, very time consuming. Before anyone else starts a new project with the model car we would suggest that some effort should be put into a technical documentation of its hardware and software.

References

[1]. Robert Bosch GmbH - *CAN specification Version 2.0.* 1991.
[2]. Hartwich, Miller, Fuhrer and Hugel - *Timing in the TTCAN Network.* Robert Bosch GmbH, 2002.
[3]. Robert Bosch GmbH - *TTCAN IP Module User's Manual Revision 1.5.* 2002.
[4]. Homepage about CAN - *www.can-cia.org/can/protocol.*
[5]. Holger Zeltwanger, CAN in Automation (CiA) - *Time-Triggered Communication on CAN*, SAE World Congress Detroit, Michigan 2002.
[6]. CAN in Automation (CiA) homepage - *http://www.can-cia.de.*
[7]. Kvaser homepage - *http://www.kvaser.com.*
[8]. Henrik Glawing - *Measurement data selection and association in a collision mitigation system.* Master's thesis at the Department of Electrical Engineering, Linköping University, 2002.
[9]. Jonas Jansson - *Collision Avoidance Theory with Application to Automotive Collision Mitigation.* PhD Dissertation No. 950 at the Department of Electrical Engineering, Linköping University, 2005.
[10]. Andreas Eidehall - *An Automotive Lane Guidance System*. PhD Dissertation No. 1122 at the Department of Electrical Engineering, Linköping University, 2004.
[11]. F. Gustafsson, J. Jansson and J. Johansson - *Decision Making for Collision Avoidance Systems.* Report number 2002-01-0403, SAE 2002, Detroit, 2002.
[12]. Fredrik Gustafsson, Lennart Ljung, Mille Millnert - *Signalbehandling.* Studentlitteratur 2001.
[13]. Jonas Cornelsen and Patrik Dahlqvist - *Development of a Safety Critical Mechatronical System used as Demonstrator for the GAST Project.* Master's thesis at KTH, 2004.
[14]. GAST project website - *http://www.chl.chalmers.se/gast*
[15]. Freescale Semiconductor's website - *http://www.freescale.com*
[16]. Martti Soininen - *Locust Project Workpackage 4: System requirements and Demonstrator Test report Preliminary Deliverable D44.* Volvo Car Corporation, 2006.
[17]. Greg Welch and Gary Bishop - *An Introduction to the Kalman Filter.* Department of Computer Science, University of North Carolina at Chapel Hill, 2004.
[18]. Joel Le Roux - *An Introduction to the Kalman Filter: Probabilistic and Deterministic Approaches.* University of Nice, 2003.
[19]. R.E. Kalman - *A new approach to linear filtering and prediction problems.* Transactions of the AMSE - Journal of Basic Engineering 1960.
[20]. Andreas Kivrikis and Johan Tjernström - *Development and Evaluation of Multiple Objects Collision Mitigation by Braking Algorithms.* Master's thesis at the Department of Electrical Engineering, Linköping University, 2004.
[21]. Richard Stafford (i), Matthias S. Keil (ii), Shigang Yue (i), Jorge Cuadri-Carvajo (ii), F. Claire Rind (i) - *Insect neural networks as a visual collision detection mechanism in automotive situations*. (i) School of Biology, University of Newcastle upon Tyne. (ii) Instituto de Microelectronica de Sevilla (IMSE), Centro Nacional de Microelectronica, 2004.
[22]. Claire Rind (i) - Locust final presentation, 2006. (i) School of Biology, University of Newcastle upon Tyne.

Appendix A:

Special Words and Abbreviations

A/D - Analogue to Digital converter
BDM - Background Debug Mode (a mode of the HCS12 microcontroller that allows advanced debug options)
Bi-i – a compact vision system used to host the Locust sensor chip
CA – Collision Avoidance
CAN - Controller Area Network (a protocol widely used in the automotive industry)
CPU - Central Processing Unit
CRC -Cyclic Redundancy Check
CSMA -Carrier Sense Multiple Access
DC - Direct Current
D/A – Digital to Analogue converter
LGMD – Lobula Giant Movement Detector, a part of the Locust grasshopper's visual system
x – the states in a state-space model (vector)
y – the measured signal in a state-space model (vector)
w – process noise (vector)
v – measurement noise (vector)
Q – process noise covariance matrix
R – measurement noise covariance matrix
P - estimation error covariance matrix, used in the Kalman filter
PDF – Probability Density Function
FAR – Project in which the model car used in this thesis project was originally developed
FSE - Frame Synchronisation Entity
GAST – General Application Development Boards for Safety Critical Time-Triggered Systems (A joint project between Chalmers and Volvo Car Corporation)
HCS12 - a microcontroller from Motorola (now Freescale) that is popular in the automotive industry
HMI - Human Machine Interface
I/O - Input/Output
NTU - Network Time Unit (used in the TTCAN protocol)
PCB - Printed Circuit Board
PWM - Pulse Width Modulation
RAM - Random Access Memory
RS-232 - a common serial communication standard (used by the regular COM-ports on a PC)
RTCOM - Real Time Communication
RTOS - Real Time Operating System
SCI - Serial Communication Interface
Token Ring - a method decide who can use a shared bus at any given time, only the node holding the 'token' is allowed to send, in the model car used in this project a manual wheel switch was used to determine which node should have the 'token', this was used mainly to choose which node got to print its debug messages and which node to program when downloading code from the PC.
TTCAN - Time Triggered CAN (Controller Area Network)
TUR - a ratio between the length of an NTU and the length of the FSE specific time unit
XCC - the C development environment used in this thesis project
XC90 - a Volvo car model

Appendix B:
Overview of the C Code Files in the Model Car

This section gives a short overview of the file structure of the model car code. This was not available to us when we started the work on our thesis project, so we put one together as we slowly 'reverse engineered' the functionality of the model car during the course of our thesis work. This took a lot of time and effort so a short summary of the code structure is presented here to facilitate work for future thesis workers or others who might continue to expand the model car's functionality. In figure 40 below a graphical chart of the file structure can be seen.

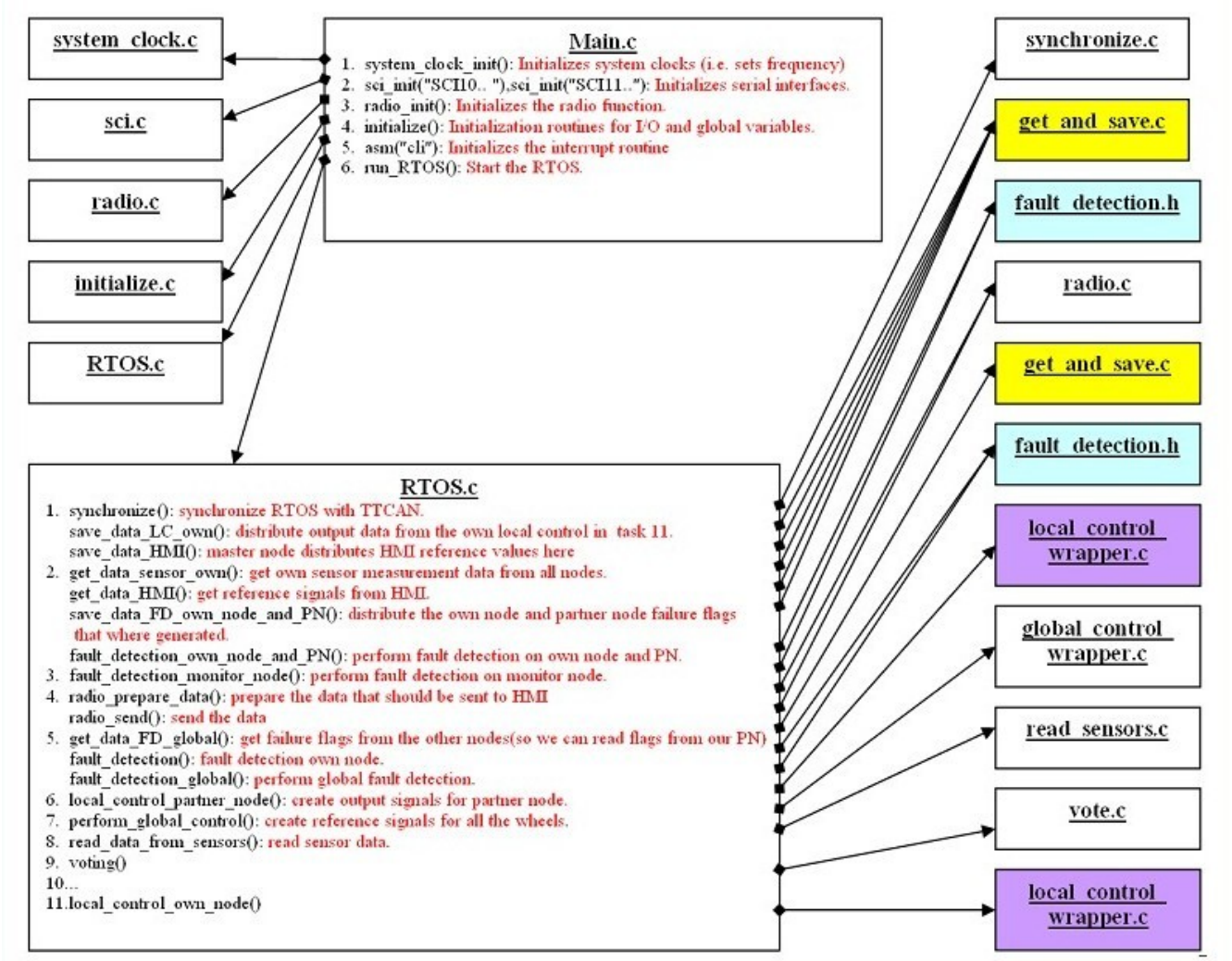


Figure 40: Boxes represent files. Arrows link functions to the files they reside in. Boxes of the same color (except white) represent the same file.

A short summary of what the files do:

Various utility files, copied from third party:
ADV64.H, AUXDEF.H, COMBINED.h, COPY.h, DIV.h, DSFxp.h, FIR.h, FIT.h, MUL.h, NEG.h, SAT.h, SHL.h, SHR.h, SQRT.h, SUM.h, tl_types.h

Files with functions generated by TargetLink:
Fault_Detection_Global.h, GlobalControl.h, FD_Monitor.h, FD_Own.h

Fault_detection.h: contains fault detection functions.
void fault_detection_own_node_and_PN(void);
void fault_detection_monitor_node(void);
void fault_detection_global(void);

Encoder_drv.h: contains driver functions to read data from the encoders fitted on the DC motors.
void inti_encoder(void);
unsigned int read_encoder(void);

Fd.h: contains fault detection functions that are not generated by TargetLink, used instead of the Targetlink functions since the stability is better.
void fault_detection(void);

get_and_save.h: contains functions for inter-node communication (using TTCAN).
void get_data_senor_own(void);
void get_data_HMI(void);
void get_data_FD_global(void);
void get_data_GC(void);
void get_data_sensor_PN(void);
void get_data_voting(void);
void get_data_LC_own(void);
void get_data_ENV(void);
void save_data_FD_own_node_and_PN(void);
void save_data_FD_monitor_node(void);
void save_data_FD_global(void);
void save_data_LC_PN(void);
void save_data_LC_own(void);
void save_data_HMI(void);
void save_data_GC(void);
void save_data_voting(void);

global_control_wrapper.h: contains "wrapper" functions for the TargetLink generated global control code files.
void perform_global_control(void;
void calculate_CRC(void);

global_data_h: contains declarations of all global variables.

HCS12: contains macros defining easy-to-use names for the registers, ports and modules of the HCS12 microcontroller. All ports and modules are memory mapped so from the programmer's model point of view everything looks like regular memory addresses.

Initialize.h: contains functions for initialization at model car startup. Also sets some global macros (i.e. the NODE_TYPE macro defining which node type to compile for among others).
void initialize(void);

local_control_wrapper.h: contains "wrapper" functions for the TargetLink generated local control code files.
void local_control_partner_node(void);
void local_control_own_node(void);

main.c: main program entry doing the following:

1. Initialize system clock: *system_clock_init();*
2. Initialize serial RS-232 communication:
   sci_init(SCI0, BR_57600);
   sci_init(SCI1, BR_38400); //for the environmental node
   sci_init(SCI1, BR_115200);
3. Initialize radio communication: *radio_init().*
4. Initialize I/O, TTCAN etc: *initialize().*
5. Enables interrupts (assembler call): *asm("cli").*
6. Starts the RTOS and lets it loop indefinitely:

```
while(1) {
                run_RTOS();
        }
```

mem_segments.h: defines the various memory segments.

Pwm_drv.h: contains drivers for the PWM module.
void init_PWM(void);
void servo_steer_own_set_duty(unsigned int duty_cycle);
void servo_brake_own_set_duty(unsigned int duty_cycle);
void servo_steer_PN_set_duty(unsigned int duty_cycle);
void servo_brake_PN_set_duty(unsigned int duty_cycle);
void motor_set_speed(short value);
void stop_PWM(int channel);

radio.h: contains functions for radio communication.
void *radio_prepare_data(void);*
void radio_send(void);
void radio_handle_byte(char data);

radio_drv.h: contains drivers for the radio communication. For practical reasons the handling of communication between the Locust sensor system and the model car has been integrated into these drivers. See the section "Communication Interface to the Locust Sensor System" for details.

read_sensors.h: contains functions to handle sensor data from the wheels and DC motors.
void read_data_from_sensors(void);
void read_sensors(void);

RTOS.h: contains the real time operating system functions:
void init_RTOS(void);
void run_RTOS(void);
interrupt void timerIRQ(void);
void timer_tick(int microsec);

sci.h: contains drivers for the serial communication interface module (SCI).

smallprintf.h: contains printout functions. Also used for sending return data to the Locust sensor system via RS-232.
*void smallprintf(char *fmt , ...);*
*void smallprintf_SCI1(char *fmt , ...);*

Subsystem.h: used by the TargetLink generated functions.

syncronize.h: contains a function to synchronize the RTOS with the TTCAN controllers global time.
void syncronize(void);

system_clock.h: contains a function that initializes the system clock.
void system_clock_init(void);

TTCAN_drivers.h: contains drivers for the GAST RTCOM TTCAN board.

TTCAN_Init.h: contains files to initialize and debug TTCAN communication.
void init_TTCAN();
void print_TTCAN_register(void);
void print_TTCAN_register_double(void);

TTCAN_messages.h: contains definitions of message structures used in the TTCAN communication, and functions for reading and writing to those structures to the RAM on the GAST RTCOM board.
*extern Temporary_type Temp; /*used to store the message currently being sent or received from the TTCAN RAM*/*
void write_TTCAN_message(unsigned char message_object);
void read_TTCAN_message(unsigned char message_object);
void print_TTCAN_message(int message);

TTCAN_reg.h: contains macro definitions to access the TTCAN RAM and registers (which are all memory mapped).

Vote.h: contains voting functions used in the fault tolerance strategy.
void voting(void);
void vote(void);

Appendix C:

Test Cases and Results, Second Round, Sep 2006

Single Vehicle Tests

Host Approaches Guardrail

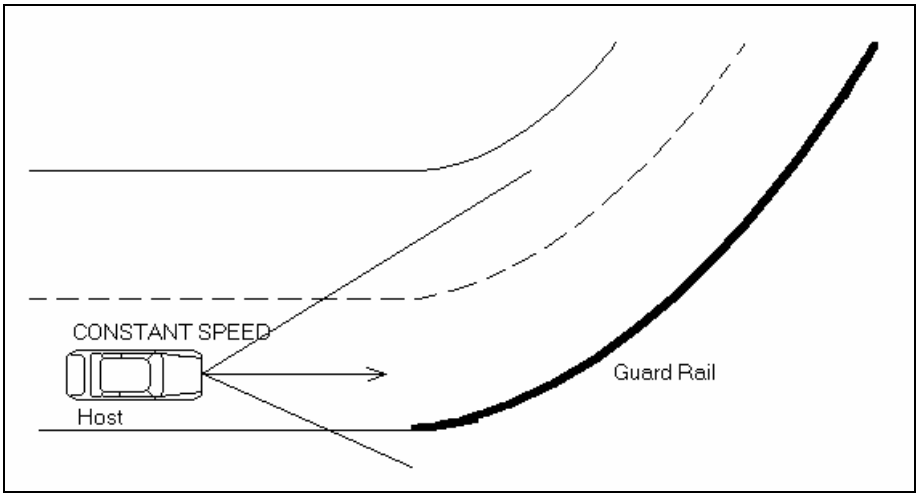


Figure 58: Test 1

Test Procedure:

Drive Host vehicle at a constant 70 km/h speed in the right lane with guardrail on the right side of the vehicle. This test can be repeated with the Host vehicle driving at a constant speed in the left lane with the guardrail on the left side of the vehicle.

Expected result: The guard rail shall not be reported as a collision threat.

Measured result:

Low guard rails were not reported as collision threats. There were some minor problems with higher fences along the road (about 1,5 m high) that would sometimes trigger false alarms when entering a curve and turning sharply and/or with high speed. The false alarms where of level 4 and always only one short beep.

Host Approaches Overhead Object

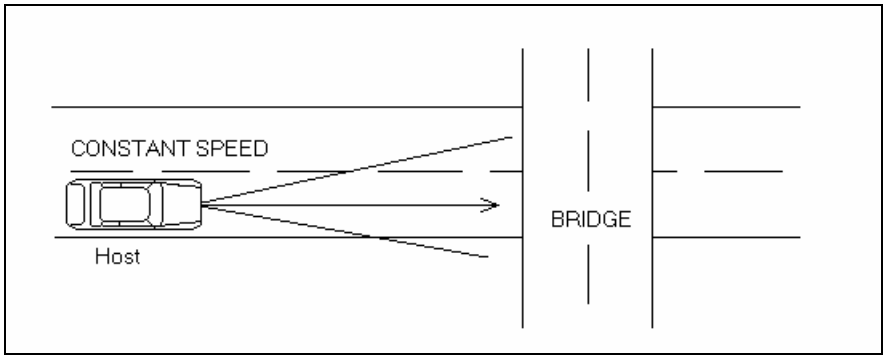


Figure 59: Test 2

Test Procedure:

Drive the Host Vehicle at constant speeds 50, 70 and 90 km/h in the right lane under a bridge. This test begins when the Host vehicle's distance from the bridge is 1.25 times the maximum range of the Host vehicle's range sensor. This test is concluded when the Host exits the other side of the bridge.

This test scenario will be used at model car testing also.

Expected result: The bridge shall not be reported as a collision threat.

Measured result:

At speeds around and above 50 km/h there would sometimes be false alarms if the bridge was very low (about 1 m height margin from the host car). The false alarms were always of level 4 and only one short beep. Normal height bridges and lower speeds presented no problems.

Host Turns with Non-Vehicle Objects Near Road

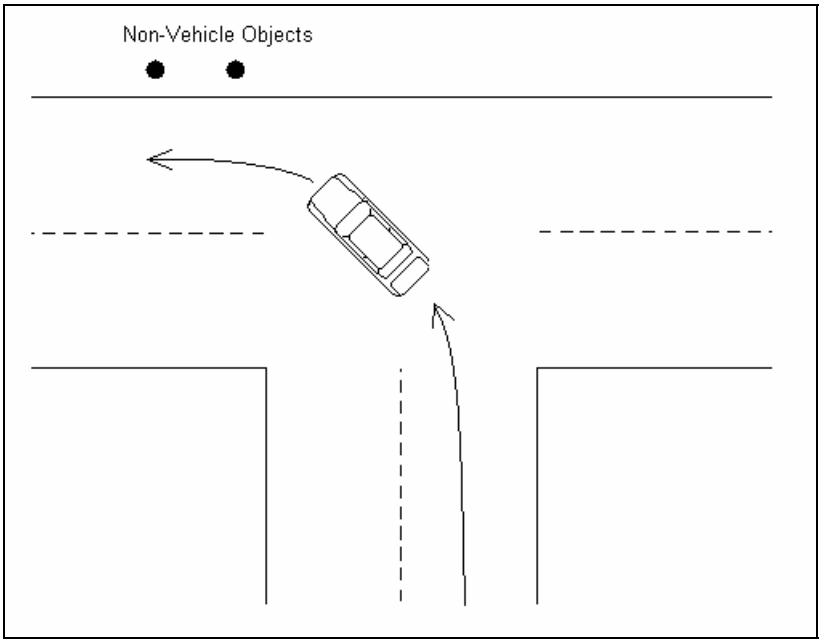


Figure 60: Test 3

Test procedure:

Drive the Host vehicle at a constant appropriately low speed toward intersection. At a safe distance turn the host toward the Non-Vehicle (a piece of stationary cutter) near the road. This test begins when the Host vehicle's distance from the Non-Vehicle object is 1.25 times the maximum range of the Host vehicle's range sensor. This test is concluded when Non-Vehicle object is no longer in the Host Vehicle's range sensor field of view.

This test scenario will be used at model car testing also.

Expected result: The object shall not be reported as a collision threat.

Measured result:

This would trigger false alarms in about 10% of the cases, if the objects came within 1 m of the host car during sharp and/or fast turning. This was a considerable improvement from the preliminary Locust system performance, but is still a problem that needs to be addressed.

Two Vehicle Tests

Host Approaches Opposing Target in Adjacent Lane

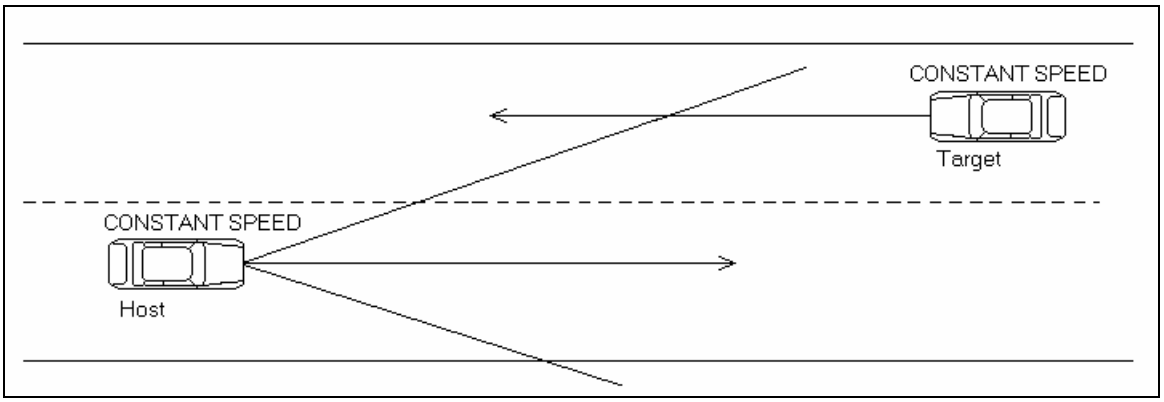


Figure 61: Test 4

Test procedure:

Drive the Host vehicle at constant 70 and 90 km/h speeds in the right lane toward the Opposing Target vehicle driving at a constant speed in the Adjacent left lane. This test begins when the Host vehicle's distance from the target is 1.25 times the maximum range of the Host vehicle's range sensor. This test is concluded when the Target is no longer in the Host vehicle's range sensor field of view.

This test scenario will be used at model car testing also. Another model car will be used as target vehicle.

Expected result: The on-coming vehicle shall not be reported as a collision threat.

Measured result:

No problems at low relative speeds. However relative speeds of about 100 km/h or more would trigger false alarms about 1,5 s before passing. The threats were mostly level 5 and beeped until the threat car had passed. This was a serious flaw and something that was not a problem in the preliminary Locust system.

Host Approaches Stationary Target in Same Lane

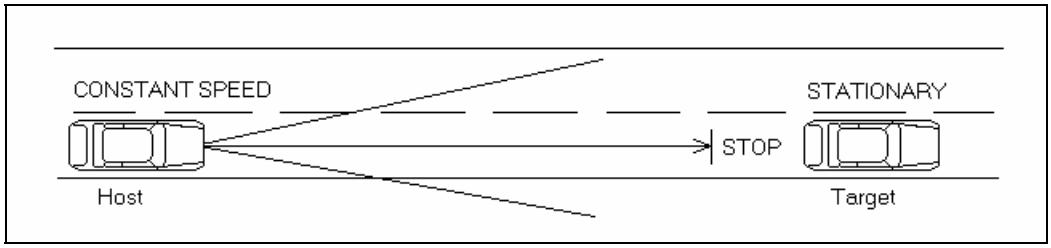


Figure 62: Test 5

Test Procedure:

Drive the Host vehicle at a constant km/h speed towards the Target vehicle in the same lane. Bring the Host vehicle to a complete stop behind the Target vehicle. This test begins when the Host vehicle's distance is 1.25 times the maximum range of the Host vehicle's range sensor. This test is concluded when the Host vehicle comes to a complete stop. Host vehicle speed 20 km/h when a real car as a target vehicle. Host vehicle speeds 20, 30 and 50 km/h when the target vehicle is a balloon car.

This test scenario will be used at model car testing also. Another model car or soft dummy car will be used as target vehicle.

Expected result: The stationary target vehicle shall be reported as a collision threat.

Measured result:

Speed 20 km/h: No warning, felt like a safe situation (stopped about 5 m away). There were level 5 warnings when stopping closer (1-2 m away). The driver was able to stop the car with some margin to avoid collision after reacting to the warning. The results were generally very satisfying at this speed.

Speed 30 km/h: Level 5 warning about 5-6 m away from collision, felt dangerous. The driver was just about able to stop the car in time to prevent a collision after reacting to the warning. The results were generally very satisfying at this speed.

Speed 40 km/h: Level 5 warning about 6-7 m away from collision, felt dangerous. The driver was just about able to stop the car in time to prevent a collision after reacting to the warning. The results were generally very satisfying at this speed.

For speeds above 50 km/h the Locust system performance decreased significantly with increasing speed, and avoiding collision from reacting to the warnings was no longer possible, but could still be of use for collision mitigation.

Host Approaches Target Traveling Perpendicular to Host

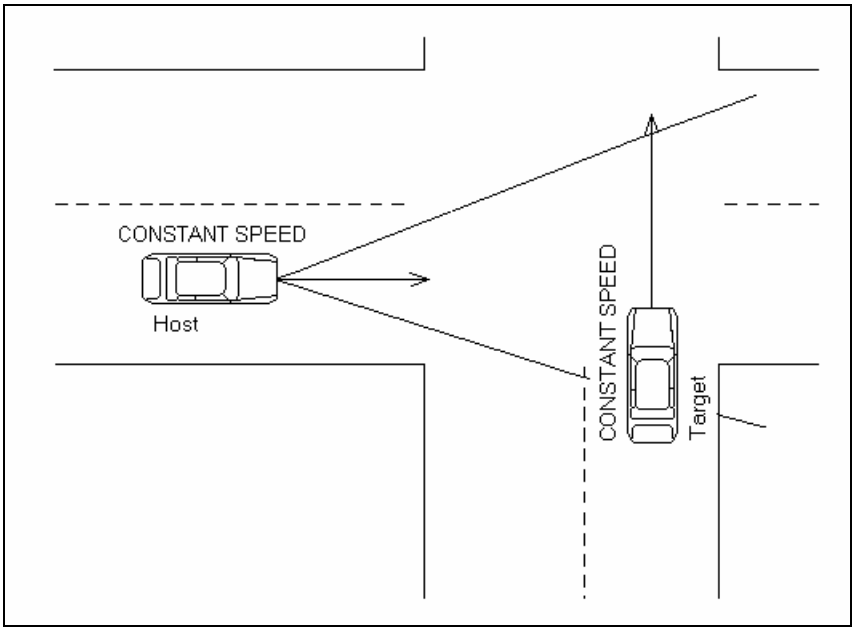


Figure 63: Test 15

Test Procedure:

Drive the Host vehicle at a constant 20 km/h speed while the Target vehicle drives at a constant speed perpendicular to the Host. This test begins when the Target vehicle enters the intersection. This test is concluded when the Target is no longer in the Host vehicle's range sensor field of view. This test can also be done using a balloon car, if the balloon car movement can be arranged.

This test scenario will be used at model car testing also. Another model car or a soft dummy model car will be used as target vehicle.

Expected result: The target vehicle shall be reported as a collision threat, if there would be a collision if the host vehicle wouldn't brake.

Measured result:

There were almost no false warnings, and there were warnings if the passing was very close. Generally we feel that the Locust system performance was very good in this test situation.

PEDESTRIAN TESTS

Pedestrian on a zebra- crossing

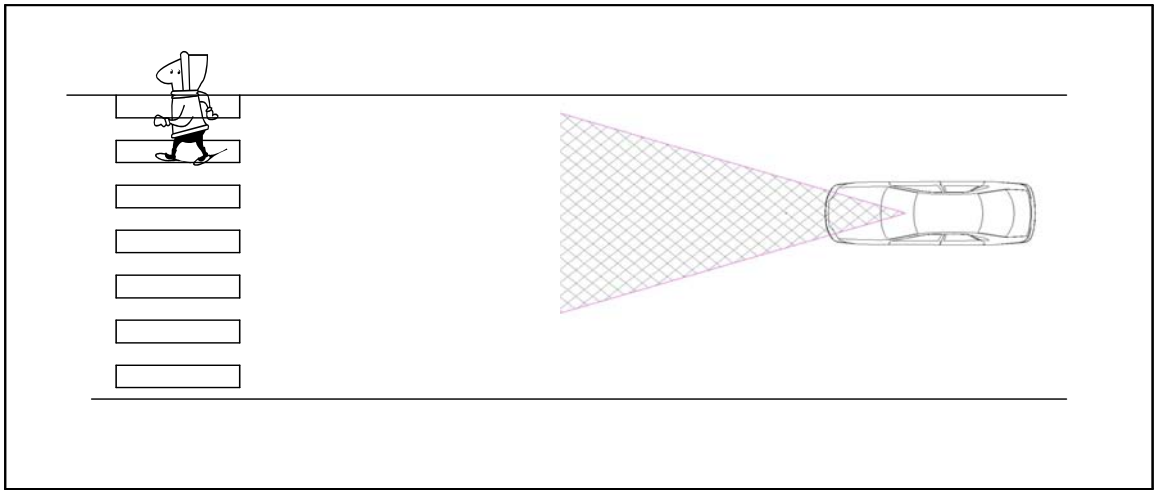


Figure 64: Test 31

Test Procedure:

Drive the host vehicle at a constant low speed while a pedestrian appears suddenly from the side and 1) walks over the street, 2) runs over the street at the zebra- crossing. The test starts just before the pedestrian is within the field of view of the sensor.

The test is over, when the sensor reports a collision threat. Testing speeds 5 and 10 km/h if safe testing can be done, otherwise lower speeds

This test is very risky and shall be done with voluntary personnel. The same test at speeds 10, 20, 30 and 40 km/h shall be done using a soft crash dummy that is mechanically moved over the street.

Expected test result: The pedestrian shall be detected in time to warn the driver or to activate automatic car systems.

Measured result:

Speed 5 km/h: The pedestrian was reported as a threat at about 1 m distance away from collision, and no false warnings according to our judgement.

Speed 10 km/h: The pedestrian was reported as a threat at about 1.5-2 m distance away from collision, and no false warnings according to our judgement.

In general the test results for this situation were extremely satisfying, no false warnings and no missed “true” warnings. Also all warnings came in time to completely avoid collision. It seemed that the speed played a much more significant role in the warning mechanism than in the preliminary Locust system testing.

Pedestrian entering the street between parked vehicles

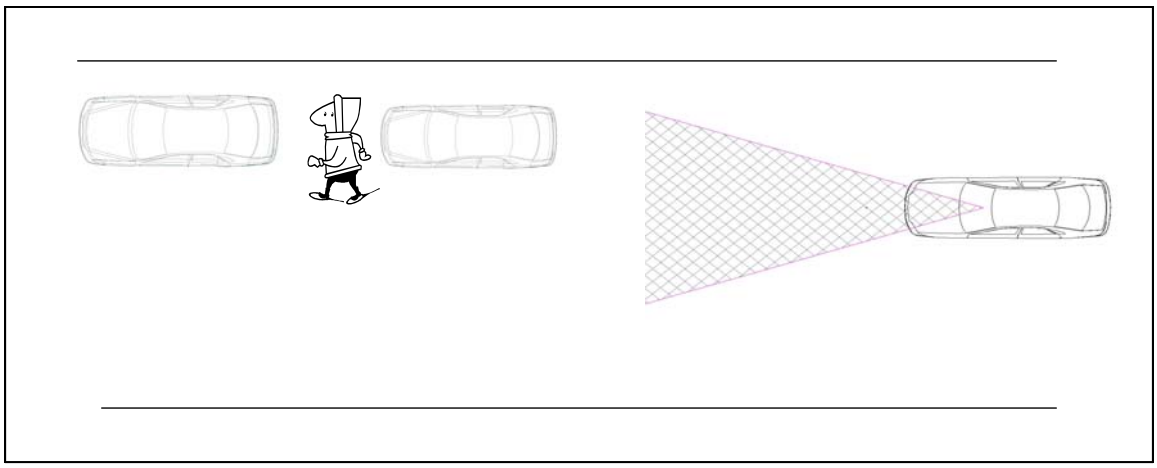


Figure 65: Test 32

Test Procedure:

Drive the host vehicle at a constant low speed while a pedestrian appears suddenly from the side between parked vehicles and 1) walks over the street, 2) runs over the street. The test starts just before the pedestrian hiding vehicle is in the field of view of the sensor.

The test is over, when the sensor reports a collision threat. Testing speeds 5 and 10 km/h if safe testing can be done, otherwise lower speeds.

This test is very risky and shall be done with voluntary Volvo personnel. The same test shall be done at speeds 10, 20, 30 and 40 km/h using a soft crash dummy that is mechanically moved over the street.

Expected test result: The pedestrian shall be detected in time to warn the driver or to activate automatic car systems.

Measured result:

Speed 5 km/h: The pedestrian was reported as a threat at about 1 m distance away from collision, and no false warnings according to our judgement.

Speed 10 km/h: The pedestrian was reported as a threat at about 1.5-2 m distance away from collision, and no false warnings according to our judgement.

In general the test results for this situation were extremely satisfying, no false warnings and no missed “true” warnings. Also all warnings came in time to completely avoid collision. This was a great improvement from the preliminary Locust system, where the cars by the side of the road caused huge problems.

Pedestrian walking along a street and a vehicle is approaching

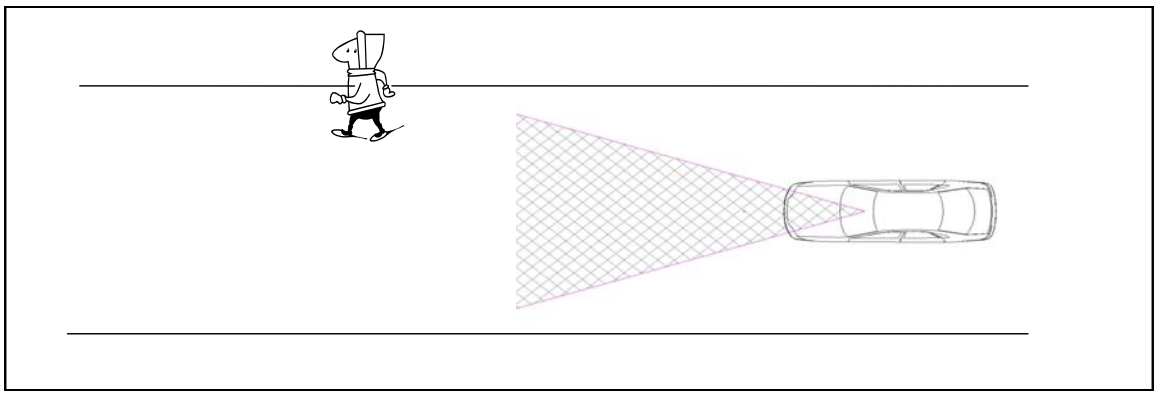


Figure 66: Test 33

Test Procedure:

Drive the host vehicle at a constant low speed while a pedestrian appears walking along the road, close to the road side 1) in the same direction as the host vehicle and 2) walking towards the host vehicle. The test starts just before the pedestrian is in the field of view of the sensor.

The test is over, when the sensor reports a collision threat. Testing speeds 20, 30 and 40 km/h if safe testing can be done, otherwise lower speeds.

This test is very risky and shall be done carefully with voluntary Volvo personnel.

Expected test result: The pedestrian shall be detected in time to warn the driver about a possible collision threat or to activate automatic car systems. The sensor shall not report a collision threat, if the pedestrian is clearly outside the path of the host vehicle.

Measured result:

In the case of pedestrians walking beside the road and the host passing them by the pedestrians were reported as threats, if it was closer than about 1 m. The pedestrians were also reported as threats when they were standing in clear collision course with the host vehicle. The warnings came about 3-7 m away from collision depending on the speed (20, 30 or 40 km/h). The performance was generally pretty good for this test case. This was a significant improvement over the preliminary Locust system tests.