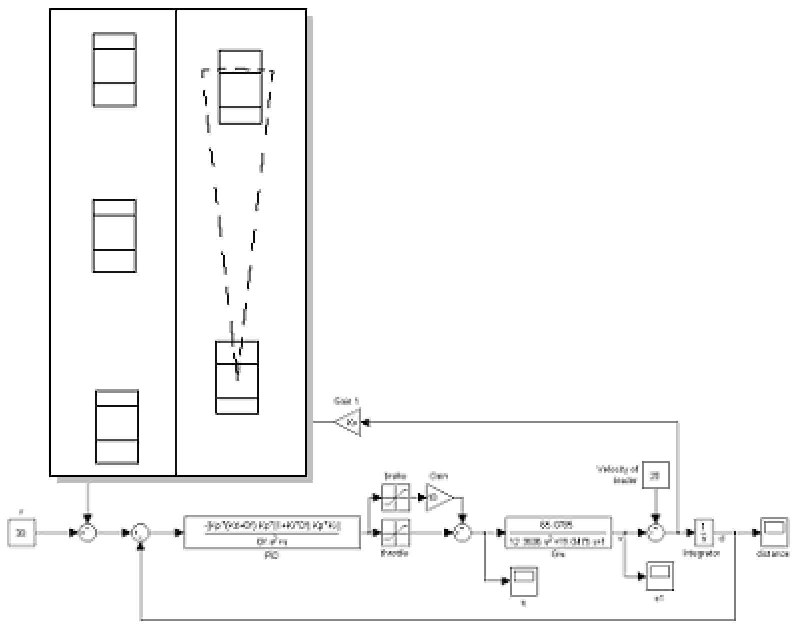


Autonomous Cruise Control for Chalmers Vehicle Simulator

Design and Implementation

PÄR BERGGREN

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2008

Master Thesis: EX047/2008

MASTER THESIS: EX047/2008

Autonomous Cruise Control for Chalmers Vehicle Simulator
Design and Implementation

Author: Pär Berggren

Examiner: Senior Lecturer Jonas Fredriksson

Department of Signals and Systems
CHALMER UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2008.

Autonomous Cruise Control for Chalmers Vehicle Simulator
Design and Implementation
PÄR BERGGREN

Master Thesis: EX047/2008
Department of Signals and Systems
Chalmers University of Technology
SE - 412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
A combination of a simulation model of the closed loop system when using the suggested control approch, see page 34, together with an illustration of the accepting range of the virtual radar sensor developed in this paper, see page 19.

Department of Signals and Systems
Göteborg, Sweden 2008.

Autonomous Cruise Control for Chalmers Vehicle Simulator
Design and Implementation

PÄR BERGGREN
Department of Signals and Systems
Chalmers University of Technology

ABSTRACT

In this thesis an Autonomous Cruise Control for a vehicle simulator is designed and implemented. Autonomous Cruise Control is a type of cruise control based on distance keeping as opposed to velocity keeping. In order to establish realtime communication over an existing CAN-bus the program dealing with this was remodelled and changes to the program was implemented. In order to allow feedback of the distance to and velocity of a leading vehicle, a plug-in to the vehicle simulator´s graphics program was created. A cascaded PID controller was designed and compared with a controller taken from litterature. Simulations suggest that these controllers are comparable in certain simulated safety aspects.

Keywords: Autonomous Cruise Control, Chalmers Vehicle Simulator, radar, gray-box model, step response, residual analysis, PID controller, emergency stopping, cut-in situations, CAN-bus, realtime communication.

Table of Contents

List of Symbols

k		Sample index.
M		Sample horizon.
N		Number of samples.
s		Laplacian variable (Complex frequency).
t	[s]	Time.
τ	[s]	Time; variable.
ω	[s^{-1}]	Frequency.

Control:

a(t)	[m / s]	Acceleration.
A_m		Gain margin.
b($\dot{x}$, $\ddot{x}$)		The uncontrollable part of the acceleration dynamics of a car.
c		Constant; D_f over (K_P K_D).
c(t)		Designable part of the AICC controller.
C_P		Feedback gain of distance for AICC.
C_V		Feedback of vehicle velocity.
d	[m]	Distance variable.
D_f		Derivation filter time.
d_m($\dot{x}$)	[kg m / s^2]	Mechanical drag of car.
e		Control error.
e_D		Distance error.
e_V		Velocity error.
K_a		Feedback of vehicle acceleration.
K_D		Derivation gain.
k_d	[kg / m]	Aerodynamic drag of car.
K_I		Integration gain.
K_P		Proportional gain.
K_v		Velocity gain.
K_v		Feedback gain of velocity (difference).
m	[kg]	Mass of vehicle.
N		Number of circulations around minus one.
P		Number of poles in the right half-plane of the complex plane.
R		Radius of halfcircle and refence value in Laplace plane.

r	[-]	Radius of halfcircle and reference value.
S_0	[m]	Minimum safe distance, used in AICC.
$t_{5\%}$	[s]	Settling time.
$u(t)$		Control signal.
$v(t)$	[m / s]	Velocity.
v_D	[m / s]	Velocity difference.
v_l	[m / s]	Velocity of leading vehicle.
w		Measurement noise.
x	[m]	Position of your car.
$\dot{x}$	[m / s]	Velocity, time derivative of position.
$\ddot{x}$	[m / s^2]	Acceleration, second time derivative of position.
Z		Number of zero crossings in right half-plane of complex plane.
$\alpha(\dot{x})$	[(kg s)$^{-1}$]	Function multiplied to throttle in AICC.
$\delta(t)$	[m]	The safety distance policy in AICC.
θ	[rad]	Angle.
λ	[s^{-1}]	The gain of the velocity when added to the safety distance policy from AICC.
$\tau(\dot{x})$	[s]	Time constant of engine.
φ_m	[°]	Phase margin.

Transfer functions:

$\bar{F}$	The controller without integration.
F	The controller.
G	General process model.
G_{du}	Model of the process d over u.
G_{dvl}	Model of the process d over v_l.
G_m	Model of a "real" process.
L	The loop of a feedback control system.
L_d	The loop; broken up at the distance signal.
L_u	The loop; broken up at the control signal.

Modelling:

P_r	Variance of predicted crosscovariance function.
$R_u(k)$	Covariance function of input signal.
$R_\varepsilon(k)$	Covariance function of predicted error.
$\hat{R}_{\varepsilon u}(\tau)$	Predicted crosscovariance function.
u(t)	Input signal.
y(t)	Output signal.
$\hat{y}(t\|\hat{\theta}_N)$	Prediction of system output signal.
ε(t)	Prediction error.
$\hat{\theta}_N$	Estimated model parameters.

Automata:

Q_P	The finite set of state-names of automaton P.
Σ_P	The alphabet of automaton P.
i_P	The initial state of automaton P.
δ_P	The transition function of automata P.

Table 1: Prefixes for Binary Mulitiples. According to (Nordling, Österman, 2004) in part CU, chapter 2.6 on page 23.

Factor	Name	Symbol
2^10	kibi	Ki
2^20	mebi	Mi
2^30	gibi	Gi
2^40	tebi	Ti
2^50	pebi	Pi
2^60	exbi	Ei

Chapter 1

Introduction

THE FIRST PART of this introductory chapter consists of a short introduction to Autonomous Cruise Control and then an introduction to the Chalmers Vehicle Simulator (CVS) itself is presented. Finally the goal and scope of the thesis are presented. The two next coming chapters deals with signal routing (the CAN-bus) and sensor creation. The two thereafter following chapters deals with modelling and control design. The chapter coming after those presents and discusses the implementations done in the Simulink model used in the vehicle simulator and in the last chapter discussions, future work and conclusions are brought up.

1.1 Autonomous Cruise Control

AUTONOMOUS CRUISE CONTROL or ACC for short is a term used in this paper to describe a cruise control based on distance keeping. In the litterature this has almost as many names and variations as there are researchers in the field. Examples of the variations of names can be Adaptive Cruise Control and Autonomous Intelligent Cruise Control.

The name used here is chosen to minimize the confusion of the reader and in order not imply that the ACC designed in this paper is of adaptive type (no parameter is changed by the control algorithm) or some type of machine intelligence.

An ACC is basically a Cruise Control taken a step further, so that instead of using a velocity feedback (from the speedometer) to maintain a preset speed, feedback of distance to and velocity of a leading vehicle is created in order to maintain a safe distance, see for instance (Iuanno, 1993). The feedback needed for an ACC is usually created using a radar but other sensors can also be used. For instance one can use GPS and communicate information about velocity and position to other vehicles over a wireless connection.

1.2 Thesis goal

THE GOAL OF this thesis is primarily to design an Autonomous Cruise Control for use on the Chalmers Vehicle Simulator. The purpose of which is to investigate a couple of control strategies. More specific goals of the control design are: elimination of persistent errors, the closed loop must be stable, the controller needs to provide a comfortable ride (none or small overthrows) it should be able to handle emergency stopping and cut-ins, and it needs to be somewhat unsensitive against measurement noise.

In order to enable use of the ACC and cruise control from the driving seat, some form of communication between the steeringwheel key panel and the Control computer needs to be established.

There is a need to have appropriate sensors mimicing the behavior of existing senors used in determining distance and velocity of vehicles in the proximity of a car.

There is an ongoing goal of increasing the realism of the simulator, associated with all work on the CVS, this is considered a secondary goal in this work.

1.3 Thesis scope

THE WORK IS centered around the design and implementation of an ACC on a vehicle simulator (more specifically Chalmers Vehicle Simulator) and tools deemed necessary to use this in conjunction with said simulator.

Realtime communication between the controls in the simulator car and the software model needs to be assured. The communication between the car body and the ACC is limited to the existing CAN-bus.

A sensor providing feedback of distance and velocity of appropriate leading vehicles needs to be developed and implemented. The sensory equipment used is limited to a software based radar sensor implemented using the plug-in capabilities of the graphics program.

The ACC needs a control law in order to work and the design and analysis of this requires a simple model of the controlled process. The controllers investigated are limited to simple PID controllers with different types of negative feedback and the model is limited to maximally second order process models.

1.4 Chalmers Vehicle Simulator

The simulator itself consists of three PC computers: one controling the simulation marked Control Computer on which the user interface is launched, one running the model called xPCTarget named after the operating system running on it and one computer running the graphics program named Graphics Computer. The Graphics computer has a program that handles the graphical environment in which the simulated vehicle can be driven in realistic traffic situations. This program has capabilities to handle plug-in programs which are ment to send information to the simulink model running on the xPCTarget Computer. The plug-ins that are currently avaliable handles things like measuring wheel positions.

Furthermore the simulator consists of a hexapodic platform with parts of a car body attached to it. In order to display the graphics a backprojector is also mounted on the platoform, as can be seen in figure 1.1. The link between the car body and the rest of the simulator is the car´s original CAN-bus, which has been connected to the Control Computer. In order to use the CAN-bus the computer has a program that handles the hardware interface named CanCom.

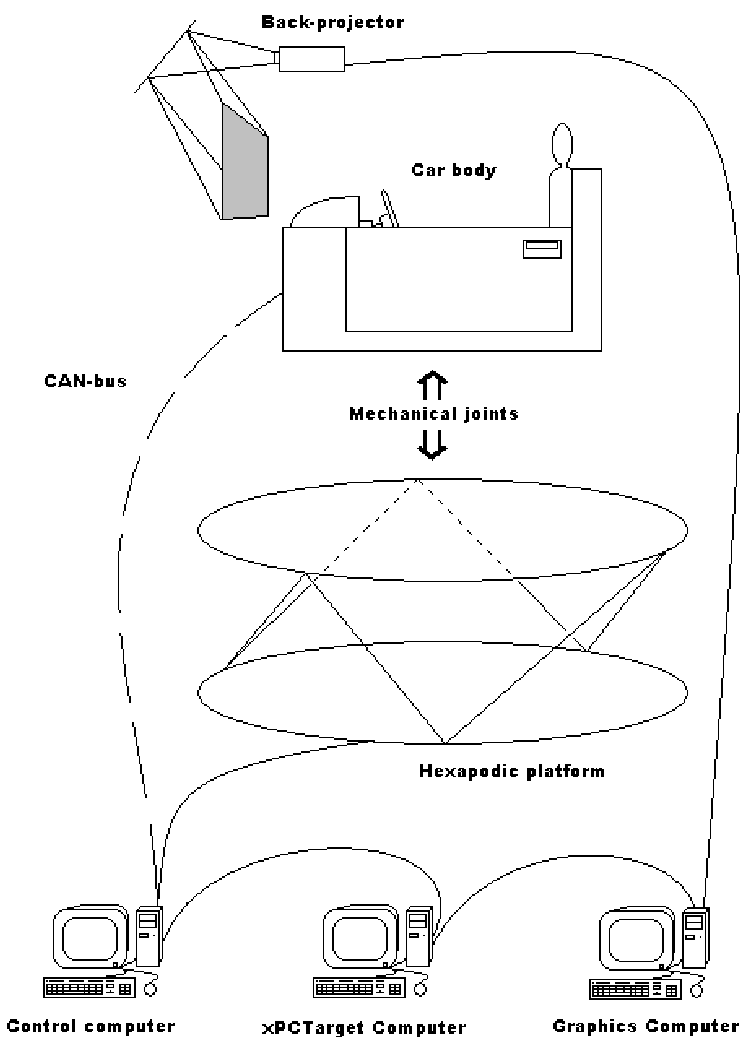


figure 1.1: An overview of Chalmers Vehicle Simulator, or at least the parts that are mentioned in this paper. The lines represent different types of communications, where the CAN-bus is dashed as it is the only one considered in the paper.

The vehicle dyamics is modelled in Simulink and this model is maintained on the Control Computer and at startup a precompiled version of this is loaded into the xPCTarget Computer and the model runs on that computer while the simulator is being used. The Simulink model is organized according to figure 1.2.

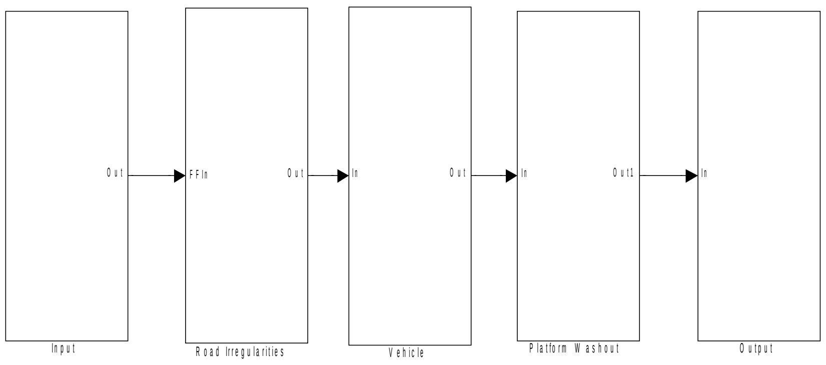


figure 1.2: The top view of the Simulink model associated with the vehicle dynamics. From left to right the blocks contain: all the inputs to the model (from hardware-based like throttle and steeringwheel to software-based like the wheel position generated in the graphics computer) , road irregularities represented by adding noise, the vehicle dynamics is computed in the Vehicle block, the motions of the platform is calculated in the Platform Washout block and the outgoing signals are communicated to their recipients in the Output block.

The block dealing with the vehicle dynamics is named Vehicle in figure 1.2. The contents of this block can be seen in figure 1.3.

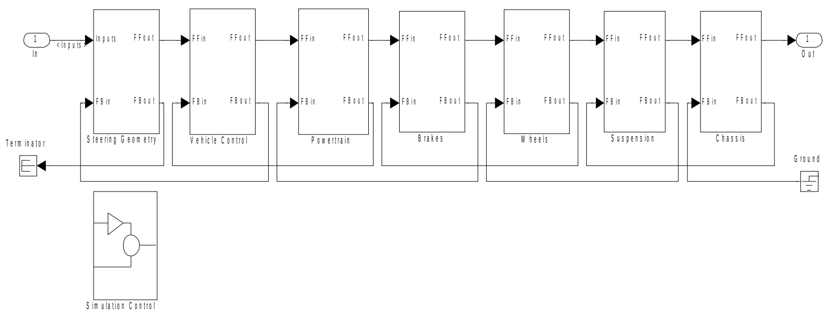


figure 1.3: The block Vehicle from figure 1.2.

In figure 1.3 one can see how information is transmitted through the model using one forward directional bus and one backward directional. The most interesting block, at least in regards to this paper, is the Vehcle Control block. The contents of this block can be seen in figure 1.4.

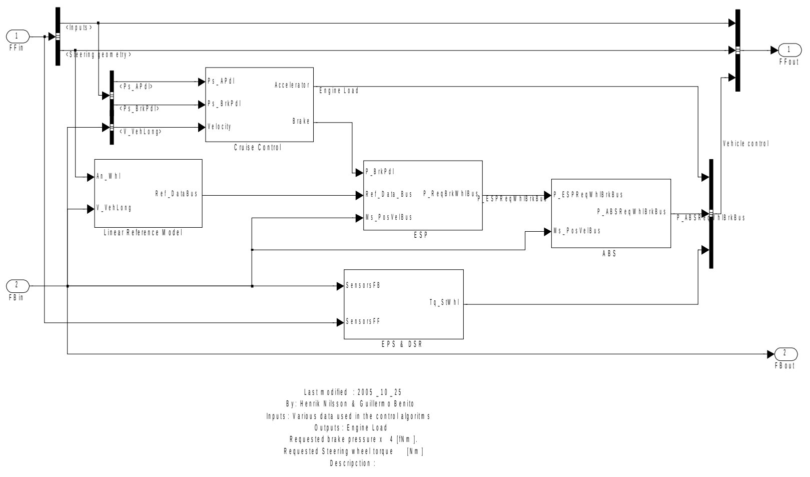


figure 1.4: The vehicle control block from figure 1.3. The linear reference model is used in the ESP, which handles stability of the vehicle. The ABS block prevents the brakes from locking up. The EPS & DSR block contains both the force feedback calculation associated with the Electronic Power Steering and a steeringwheel torque associated with a Driver Steering Recommendation system implemented in a previous master thesis.

In figure 1.5 the implementation of the Cruise Control can be seen. The Cruise Control is allowed to be enabled when the longitudinal velocity is high enough. If the throttle or brake signal is higher than what the controller dictates, the input signal is taken as outputs.

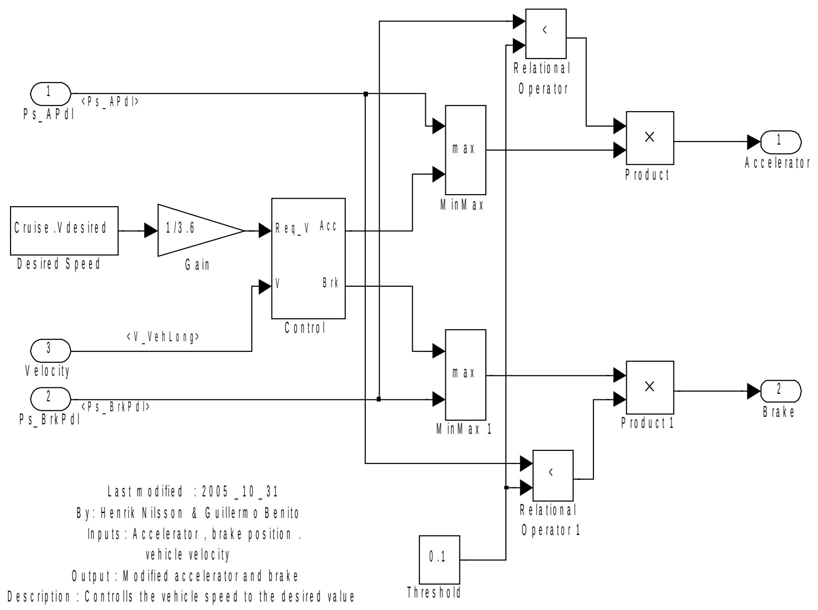
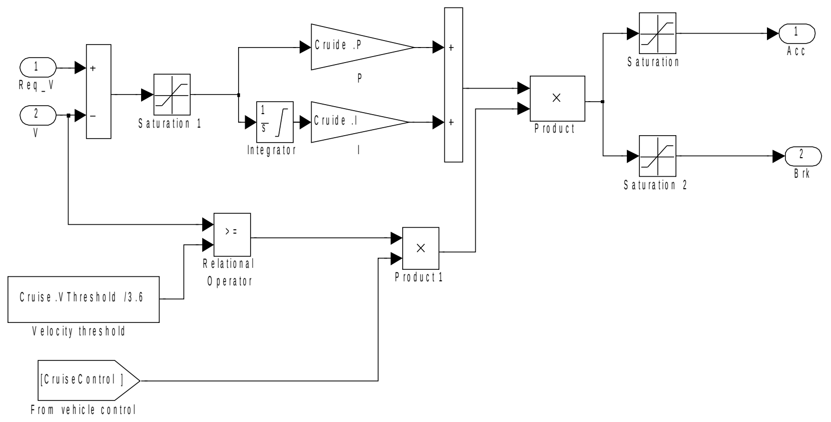


figure 1.5: The Cruise Control block from figure 1.4 showing the implementation of the previosly existing Cruise Control.

The Control block from figure 1.5 can be seen in figure 1.6 where the PI-contoller used for Cruise Control is shown. The lower speed limitation and the activation signal are also implemented here.



Last modified : 2005_10_31
By: Henrik Nilsson & Guillermo Benito
Inputs : Requested and current vehicle velocity
Output: Modified accelerator and brake
Description : A PI-Controller controlls the vehicle speed to the desired value

figure 1.6: The block named Control in figure 1.5 showing how the PI-controller is implemented.

There is completely software based version of the simulator that in this work is called offline simulator. The offline simulator only consist of a Simulink model which has the same layout as the online version.

Chapter 2

CAN-bus communication

THE PRESENTLY USED program for communication over the CAN-bus already has all of the features that is required for using it with the current applications. Features that allows usage of the existing driving information system and warning messages. It has however one distinct disadvantage in regards to some of the applications presented in this paper; a complete lack of realtime processing of incoming CAN-messages.

Little or no attention has previously been given to the realtime problem. Only a single incoming CAN-message is considered in the current application: the position of the ignition key. As one of the objectives of this work was to enable use of the existing keyboards associated with Cruise Control, which at least to some degree increases the realism of the simulator. Increasing the realtime capabilities of the CAN-bus communication program was given high priority.

2.1 A brief introduction to the CAN-bus

THE CONTROLLER AREA network or CAN for short, is a serial communications protocol mainly used within the automotive industries. Due to the cost effectiveness and security of the CAN-bus it has become commonplace in modern cars. Cars equiped with such a bus use them in a wide range of applications, from engine control units to anti-skid systems.

The CAN-bus protocol built into the simulator car is of type 2.0B, see (Robert Bosch, 1991), allowing for multiple messages with the same priority and longer message identifiers. The car also has two separate CAN-buses with differing bandwidths (125 [Kibit/s] and 250 [Kibit/s]).

The CAN-bus basically consists of a wire pair connecting two or more computers. These wires can either have the same electric potential or have different ones, corresponding to the bus communicating a binary zero or one. The wires have, if left unaffected differing potentials and has to be “written to” in order to be given the same potential. This allows the bus to prioritize between different messages in accordance with low number – high priority.

Every connected unit has one writing unit and one reading unit, these are capable of running simultaneous. The reading unit is usualy always active, in order to keep track of transmitted messages and to know when the bus is occupied. The writing unit only needs to be active when outputing a binary zero.

The original idea with the CAN protocol is that every message to be sent has a unique identification code corresponding to the message´s priority. Every message sent begins with writing the corresponding identification code. During this the bus is monitored by the reading unit and as soon as one of the readings differ from the written output; the bus is assumed to be required by a higher priority message and the output is stopped for a period of time corresponding to the length of a message. This is repeated until the message has been sent. Since the bus needs to be free in order to write a one; the messege with the lowest identification code (the highest priority) is the first to be transmitted.

2.2 Probem description

THE FIRST PROBLEM encountered with the CAN-bus was associated with wire routing. The contact mechanism that was originally placed inside the steering-wheel to allow electrical contact between the keys placed on the steering-wheel and the rest of the car, was broken and had to be repaired.

Once the contacts had been repaired, it became obvious that there was a problem with the software handling the CAN-bus communication. The problem appered as an almost twenty seconds long time delay when using the keypad associated with Cruise Control as opposed to the radio controls wich were also placed on the steering-wheel but was unaffected by any noticable time delay.

2.3 Current algorithm

IN ORDER TO analyse the software related problem, an algorithm describing the behaviour of the source code was written. The existing software solution was designed with the main goal of allowing the use of outgoing CAN messaging capabilities sending commands to the vehicle. Little attention has been given to the opposite directional data traffic, messages received from the vehicle. This meant that the software needed to be remodelled in order to allow the realtime communication required to use the cruise control keypad.

The software problem described in the section above is not exacly a design flaw, since no critical process is dependent on realtime information from the car´s CAN-bus. The usage of the cruise control keyboard however changes that; if the signals can not be received fast enough, then the use of the original keyboard would be an obstacle to the realism of the simulation.

In figure 2.1 the discussed algorithm is shown as a flow chart, together with the algorithm for the main program thread, wich mostly displays the simulated velocity and engine speed. The algorithm is based on the framesAndSignals object called fas which is a database that describe the state of all signals transmitted over the CAN-bus.

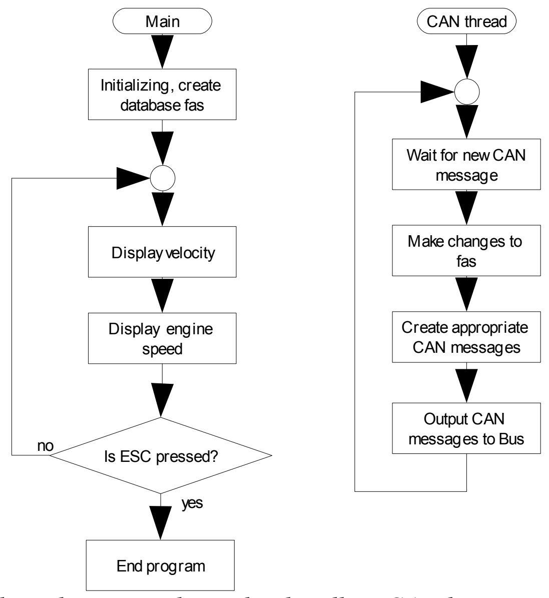


figure 2.1: Algorithm currently used in handling CAN-bus messages in the program CanCom.exe. The object of type framesAndSignals called fas seen in the second square in the CAN thread handles the current messages, both read from the CAN-bus and to be written to the same. The thread called "Main" displayed on the left is the programs main thread.

The fas object consists of all the messages that can be sent, together with timing intervals and all the messages that can be read from the CAN-bus and their last received status. The idea with the object fas is to organize the messages so that a picture of the current status of the car is easily avaliable. Creating the CAN messages is also simplified by the use of the fas, since the program only needs to cycle through it and given the time intervals, create new messages to be sent.

The CAN thread seen in figure 2.1, works very well in sending messages, since this is totally deterministic and everything associated with that is time based. However that is not the case with receiving messages from the CAN-bus, here everyting is more or less random, at least event based, and even if the messages were to be renewed on specific time intervals by an external source, they would be delayed by messages of higher priority on the CAN-bus.

2.4 Remodelled CAN communication

THE BEST SULUTION to the problem of insufficent realtime communication was deemed to be remodelling the excisting software. In order to distribute the processor time more evenly the CAN communication thread is divided into two threads. In figure 2.2 an illustration of how the remodelled program will transport information to and from the CAN-bus is shown.

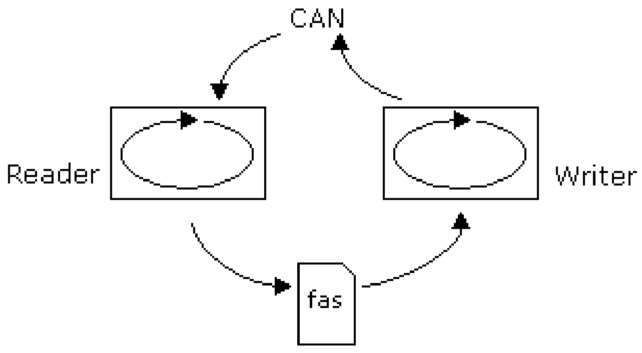


figure 2.2: Illustration of how the reader and writer threads allocates information. The fas, a framesAndSignals object, is a database of all, or at least most, signals and their last known status.

The type of system shown in figure 2.2 is known as a Discrete Event System (DES) and is easily modelled using Automata in for instance Supremica, which is a program that is used to work with DES. For a formal definition of Automata see (Fabian, 2006) page 14, here it is sufficient to know that a Automaton is a 4-tuple given by equation (2.1).

$$P = \langle Q_P, \Sigma_P, i_P, \delta_P \rangle \tag{2.1}$$

where:

Q_P Is the finite set of state-names.

Σ_P Is a finite set of event-labels called the alphabet of the automata.

i_P Is the initial state that the automaton occupies from start (belongs to the set Q_P).

δ_P Is the transition function which, for a specific event and a specific occupied state, describes which state the automaton is in after the event has taken place.

In figure 2.3 on page 15 the processes called reader and writer are shown as automata modelled as consumer and producer processes in Supremica. The reader starts at the RestRead state and when a message is received, which is marked as an uncontrollable event, it enters a state called Read in wich the fas can be updated by the event named updFas, which is a controllable event and when no more messages occupies the incoming message queue (the event queueEmpty occurs) the reader goes back into the resting state. The automata correspoding to the Reader is shown below.

$$\text{Reader} = \langle Q_R, \Sigma_R, i_R, \delta_R \rangle \tag{2.2}$$

$$\begin{aligned}
Q_R &= \langle \text{RestRead}, \text{Read} \rangle \\
\Sigma_R &= \langle \text{msgRcvd}, \text{queueEmpty}, \text{updFas} \rangle \\
i_\text{R} &= \text{RestRead} \\
\delta_R &= \langle \langle \langle \text{RestRead}, \text{msgRcvd} \rangle, \text{Read} \rangle \\
&\quad \langle \langle \text{Read}, \text{updFas} \rangle, \text{Read} \rangle \\
&\quad \langle \langle \text{Read}, \text{qeueuEmpty} \rangle, \text{RestRead} \rangle \rangle
\end{aligned}$$

The writer is modelled in a similar way. It starts in a resting state in which the event named updFas, which is the same event as in the reader, is allowed to occur and the controllable event srcCAN, in which the fas database is searched for messages to output on the CAN-bus, can occur. Most messages needs to be output with individual time intervals. When srcCAN occur, the writer changes to the Write state where the uncontrollable events outputMsg and srcCpl may occur.

$$\text{Writer} = \langle Q_W, \Sigma_W, i_W, \delta_W \rangle \tag{2.3}$$

$$\begin{aligned}
Q_W &= \langle \text{RestWrite}, \text{Write} \rangle \\
\Sigma_W &= \langle \text{outputMsg}, \text{srcCAN}, \text{srcCpl}, \text{updFas} \rangle \\
i_\text{W} &= \text{RestWrite} \\
\delta_W &= \langle \langle \langle \text{RestWrite}, \text{updFas} \rangle, \text{RestWrite} \rangle \\
&\quad \langle \langle \text{RestWrite}, \text{srcCAN} \rangle, \text{Write} \rangle \\
&\quad \langle \langle \text{Write}, \text{srcCpl} \rangle, \text{RestWrite} \rangle \\
&\quad \langle \langle \text{Write}, \text{outputMsg} \rangle, \text{Write} \rangle \rangle
\end{aligned}$$

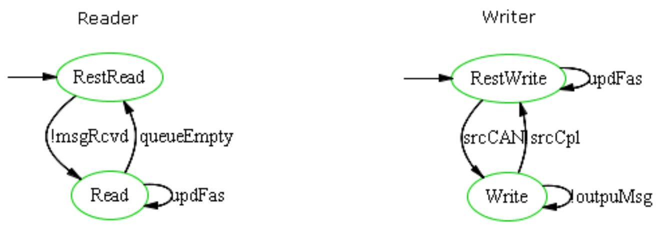


figure 2.3: The reader and writer from figure 2.2, modelled as Automata. It is important to know that they have separate alphabets. The "!" character means that the event is uncontrollable.

In order to see what happens when the reader and writer automata work in pseudo-parallel with each other, the automata are synchronized. The synchronous compostition for automaton A and B is denotet A || B and is defined by equation (2.4). The result from when synchronization was done on the reader and writer automata can be seen in figure 2.4 and equation (2.7).

$$A \| B = \langle Q_a \times Q_b, \Sigma_A \cup \Sigma_B, \langle i_A, i_B \rangle, \delta_{A\|B} \rangle \tag{2.4}$$

where:

$$\delta_{A\|B}(\langle q_A, q_B \rangle, \sigma) = \begin{cases} \{\delta_A(q_A,\sigma)\} \times \{\delta_B(q_B,\sigma)\} & \sigma \in \Sigma_A \cup \Sigma_B \\ \{\delta_A(q_A,\sigma)\} \times \{q_B\} & \sigma \in \Sigma_A - \Sigma_B \\ \{q_A\} \times \{\delta_B(q_B,\sigma)\} & \sigma \in \Sigma_B - \Sigma_A \end{cases} \tag{2.5}$$

$$C \times D = \{(c,d); c \in C \wedge d \in D\} \tag{2.6}$$

Cartesian product (Råde et al., 1998) p. 16

$$\text{Writer} \,\|\, \text{Reader} = \langle Q_{\text{W}\|\text{R}}, \Sigma_{\text{W}\|\text{R}}, i_{\text{W}\|\text{R}}, \delta_{\text{W}\|\text{R}} \rangle \tag{2.7}$$

$$\begin{aligned}
Q_{\text{W}\|\text{R}} &= \langle \text{RR.RW}, \text{RR.W R.RW}, \text{R.W} \rangle \\
\Sigma_{\text{W}\|\text{R}} &= \langle \text{outputMsg}, \text{srcCAN}, \text{msgRcvd}, \\
&\quad\ \text{queueEmpty}, \text{scCpl}, \text{updFas} \rangle \\
i_{\text{W}\|\text{R}} &= \text{RR.RW} \\
\delta_{\text{W}\|\text{R}} &= \langle \langle \langle \text{R.RW}, \text{queueEmpty} \rangle, \text{RR.RW} \rangle \\
&\quad \langle \langle \text{R.RW}, \text{srcCAN} \rangle, \text{R.W} \rangle \\
&\quad \langle \langle \text{R.RW}, \text{updFas} \rangle, \text{R.RW} \rangle \\
&\quad \langle \langle \text{R.W}, \text{outputMsg} \rangle, \text{R.W} \rangle \\
&\quad \langle \langle \text{R.W}, \text{queueEmpty} \rangle, \text{RR.W} \rangle \\
&\quad \langle \langle \text{R.W}, \text{srcCpl} \rangle, \text{R.RW} \rangle \\
&\quad \langle \langle \text{RR.RW}, \text{msgRcvd} \rangle, \text{R.RW} \rangle \\
&\quad \langle \langle \text{RR.RW}, \text{srcCAN} \rangle, \text{RR.W} \rangle \\
&\quad \langle \langle \text{RR.W}, \text{msgRcvd} \rangle, \text{R.W} \rangle \\
&\quad \langle \langle \text{RR.W}, \text{outputMsg} \rangle, \text{RR.W} \rangle \\
&\quad \langle \langle \text{RR.W}, \text{srcCpl} \rangle, \text{RR.RW} \rangle \rangle
\end{aligned}$$

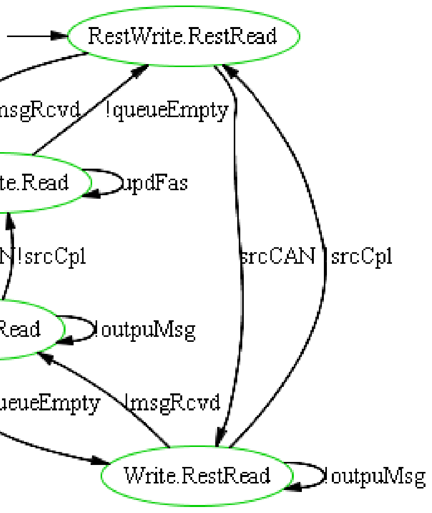


figure 2.4: The syncronization between the reader and writer found in figure 2.3. Mutual exclusion is achieved by not allowing updFAS to occur when the automaton is in the Write.Read state.

The automaton in figure 2.4 describes the desired behaviour of the reader and writer threads. There are no reachable places that are forbidden and it is easy to make sure that the threads will not get stuck in either deadlocking or livelocking using built in functions in Supremica.

The reader and writer threads were implemented under the names CANCom_thread and CANWrite_thread and can be seen in Appendix A. These threads also handles other tasks but these do not affect the CAN-bus communication. The implementation is done using a critical section which is a synchronization primitive that stops the switching of threads within the process. This is used around the events updFAS and srcCAN in figure 2.3 which are controllable events. This ensures mutual exclusion the same way as in figure 2.4.

Chapter 3

Sensor development

IN THIS CHAPTER a way of sensing the simulated vehicles in the graphics environment is developed. This is done in the graphics program using the plug-in capabilities of the program. First the graphics environment is introduced, then the algorithm used is presented. Finally the idea behind the algorithm is discussed.

3.1 The graphics environment

CHALMERS VEHICLE SIMULATOR has a graphics program developed by thesis workers. It has been equiped with plugin capabilities, which allows for measurements and computations done in the graphics program to be sent to the other computers using the UDP protocol. The graphics program is based on Open Scean Graph, (OSG Community, 2007) which is an open source graphics engine. The advantages of OSG is mainly economical (it is free to use) but also that it has extensive, easily accessed online documentation.

The online documentation of OSG has, in some manner of speaking, made it possible to work with the program since the documentation of the graphics program left much to be desired. Due to that more than one source code exists and few people know which one is actualy in use, focus was put on creating plugins to the program, instead of changing the program itself.

The existing plug-ins provide the model running on the xPCTarget computer with the position of the tires of the simulated car. The basic structure of this plug-in was used in the creation of the radar plug-in. Changes was however necessary regarding the information flow to the plug-in. In the existing plug-ins, the only apparent information accessable to the plug-in is a few variables sent from the core program. Fortunately there are no restrictions to the direct access of most objects in the program, which allows the plug-in to gather all information necessary to compute for instance the distance to, speed of and angle to any vehicle displayed by the graphics environment.

3.2 The radar sensor

ONE WAY TO sense a leading vehicle is to use a radar mounted to the front of your car. Such a sensor can give you both the distance and the velocity of a detected object. A radar sensor was implemented using the plug-in capabilities of the graphics program. An illustation of how a real radar find leading vehicles is shown in figure 3.1.

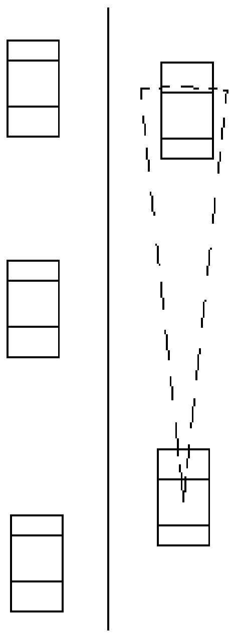


figure 3.1: Traffic situation illustrating how a radar selects leading vehicle. If a car appears within the dashed area it will be discovered by the radar and its distance and velocity is measured. If no car is present within the dashed area the maximum range and zero velocity are returned. The illustration is done off-scale.

The radar sensor implemented works in a very different way as compared to a real radar. The leading vehicle is selected using a quadratic criterion expressed in the angle and distance relative to the simulated car. The angle is calculated as the angle from a line going through both the simulated car and the leading vehicle, to a plane defined by the global z-axis and the simulated car´s direction. Thereby the angle is limited to positive values below 90º, which is acceptable since only cars in front of the simulated car, are considered and the selection is only sensitive to the angle squared. The quadratic criterion is shown in figure 3.2.

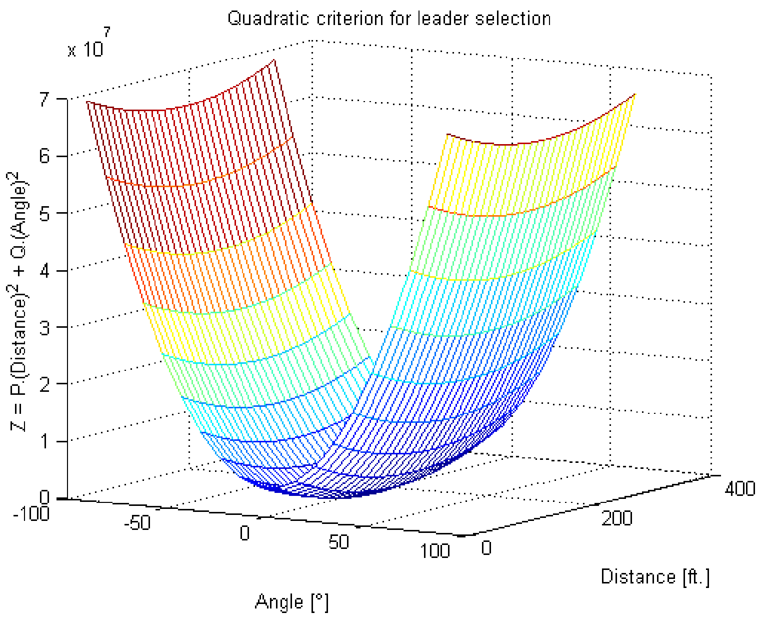


figure 3.2: The quardratic criterion used when selecting a leading vehicle. The criterion is mirrored in the plane where the angle is zero and centered around an assumed reference distance of 50 meters (150 feet).

The realism of this criterion is questionable since vehicles appearing between the leader and the simulated car recieves a lower value than the vehicle being followed. A possible solution to this problem could be to use a criterion that is cubic in the distance variable. Such a criterion would however be more sensitive to measurement noise, since the leading vehicle would not be close to a minimum. Therefore a quadratic criterion was deemed sufficiently realistic and was implemented in the source code of the radar sensor, which can be found in Appendix B.

A measurement series was taken in order to find out how much noise the signal is exposed to and the resulting noise is shown in figure 3.3. The distance noise shown would probably not be much of a problem, especially if low-pass filtered. However, the noise of the measured velocity of the simulated cars are equal to the noise of the numerical derivation of this signal, which is shown in figure 3.4.

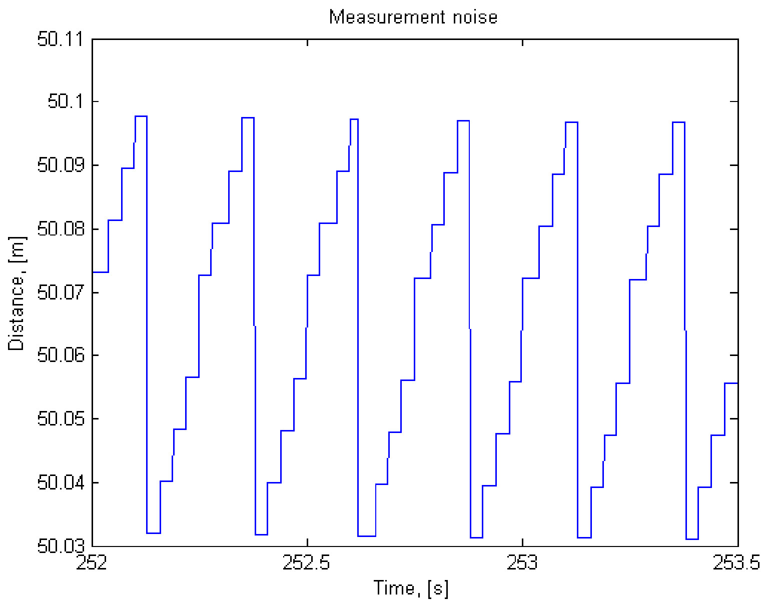


figure 3.3: The noise of the distance signal measured with the software based radar. The reason for the periodicy of the noise comes from the source of the noise, which is the pixelation of the position of the car.

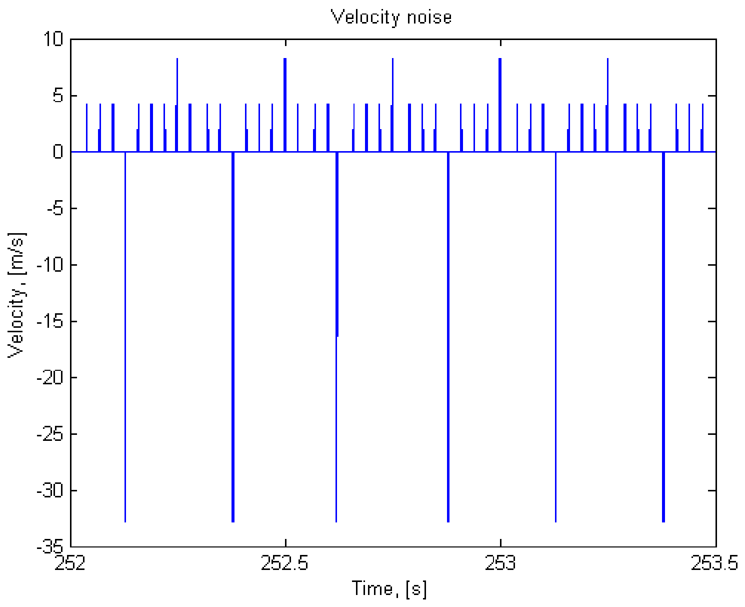


figure 3.4: The numerical derivation of the signal shown in figure 3.3. Illustrates the noise of the measured velocity signal.

A reasonable way of increasing the realism of the sensor would be to add noise to the measurements. It could for instance be more realistic to add noise with a variance of half a car length to the distance measurement, thereby simulating different reflective surfaces, see the implementation done for the offline version of the simulator in figure 6.1 on page 56.

Chapter 4

System modelling

IN ORDER TO perform and analyse the control design a simple model describing the linear parts of the simulator is required. Therefore a collection of mathematical models representing the process: car velocity over level of acceleration requested, were created using a step response. Thereafter the models were evaluated against each other, in order to find the best approximation of the simulation result. Finally the model is modified to represent the distance between a leading vehicle and the simulated car, thereby creating a grey-box model suitable for control design and analysis.

4.1 Model synthesis

THE GOAL OF this modelling is not to create a “perfect” model, just to create a model good enough to design a controller from. Therefore and for other reasons, like computational efficiency and to keep the model generally applicable, no model with order higher than second will be considered. Moreover the process is initially assumed not to need any zeros in order to accurately describe the model.

4.1.1 Experiment and data collection

In order to allow experiments outside of the simulator, there is an offline version of the simulink model which was used to carry out the above mentioned step response. The main difference between the offline and the online versions is that all external communication blocks have been replaced with signal builders and constants for all inputs and scopes for all outputs.

The experiment was designed with a step in the signal builder representing the throttle, at 25 seconds after start, when the throtte was set to full. All the rest of the signal builders was set to appropriate values and a switch was added to allow use of the automatic gearbox. After the experiment was completed, the result was recorded in an iddata object as displayed in figure 4.1.

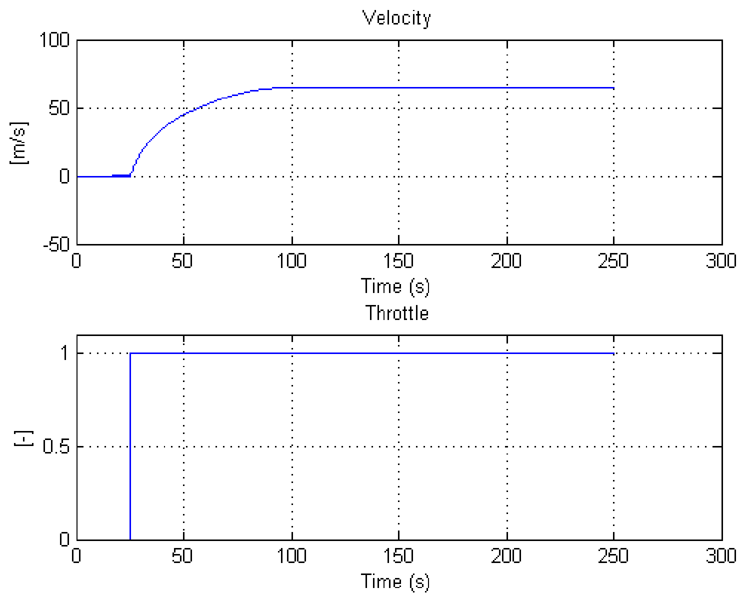


figure 4.1: An oveview of the data stored in an iddata object containing the resulting velocity (and throttle) from a step in throttle level. Experiment carried out on the offline version of the simulink model, using the same automatic gearbox as used in the online version.

4.1.2 Modelling

Two different modelling techniques were used in the model design. Mostly to benchmark the best model, but also to make sure the selected technique is better then at least one other. The techniques used was: Matlab´s System Identification Toolbox, or SIT for short, and step response identification for processes with two time constants, as presented in (Thomas, 2001), or BT for short.

SIT is a toolbox in Matlab which uses an itterative identification process to identify models. It is easy to use and offers a wide range of model types, such as Box-Jenkins type parametric models and process models. SIT is designed with the intent to allow evaluation of different model types, however the only models considered here is of process type, so only a part of SIT´s capabilities are used. The models estimated in SIT is presented in equations (4.1) and (4.2).

$$G_{SIT,2\mathrm{nd}\ order} = \frac{65.07}{2.364s^2+19.05s+1} \qquad (4.1)$$

$$G_{SIT,1\mathrm{st}\ order} = \frac{65.07}{31.07s+1} \qquad (4.2)$$

The BT modelling technique is based on manual calculations and measurements in two graphs. This requires a little bit more work than SIT and do not offer as much in the way of evaluation tools, so this has to be done manually. The BT method however has the advantage of not requiring any software licence.

The BT method consists of the four steps:

1. The first step in the method is to divide the final value with the step size to achieve the static gain, called K, of the system.

2. The next step is to measure the $t_{1/3}$ and the $t_{2/3}$ times, the times at which one third and two thirds of the final value is achieved.

3. The ratio $t_{2/3}$ over $t_{1/3}$ is then computed and given the name Q. The constant Q is then used to aquire valus for the constants P and a from two graphs presented in figure 4.2.

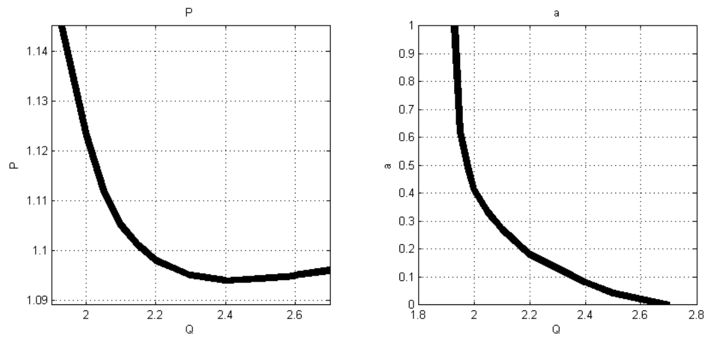


figure 4.2: The graphs used to measure the constants a and P. See (Thomas, 2001), on page 113.

4. The final step is to calculate the largest time constant T using equation (4.3) and to put all the calculated constants (K, a and T) into equation (4.4) which then becomes the model.

$$T = \frac{t_{2/3}}{P(1 + a)} \tag{4.3}$$

$$G = \frac{K}{(1 + Ts)(1 + aTs)} \tag{4.4}$$

Table 4.1: Measured and calculated constants related to the BT modelling process. the constants a and P is taken as extreme values (due to high Q) from figure 4.2.

Constant	***Measured value***
K	65.08
$t_{1/3}$	7.28
$t_{2/3}$	23.66
Q	3.25
a	0
P	1.096
T	10.33

Due to the low constant “a”, the model estimated with the BT method becomes a first order model, which is presented in equation (4.5).

$$G_{BT} = \frac{65.08}{21.59s+1} \tag{4.5}$$

In figure 4.3 a collection of step responses on the designed models given by equations (4.1), (4.2) and (4.5) is shown together with the data collected from the experiment. This figure suggests that either the model represented by the dashed black line, equation (4.1), or by the dotted red line, equation (4.5), gives the the best approximation of the real system based on the step response, here represented by a solid blue line.

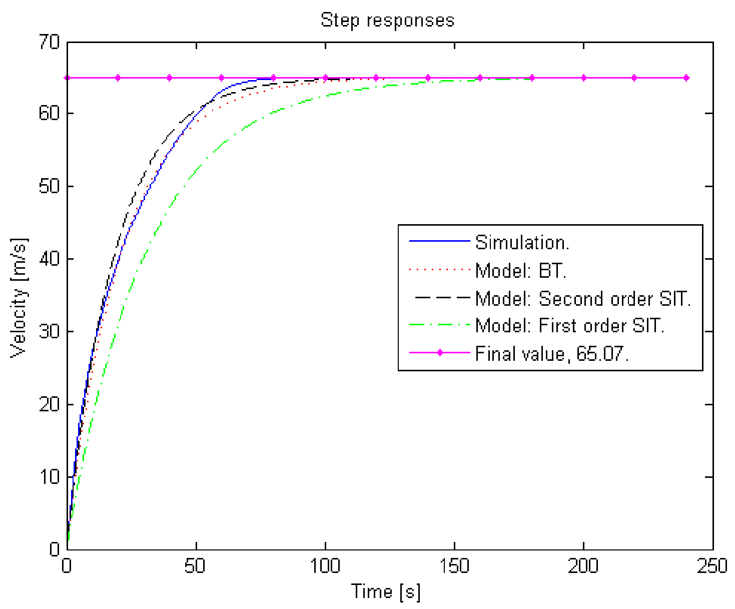


figure 4.3: Step responses of the different models designed with the data shown in figure 4.1. It is hard to tell which model is the best one from this graph, so the choise of model will be decided in the verification section.

4.2 Model verification

In order to get a measurement on the quality of the model, it is important to verify the model. This is best preformed on a different set of data than that which the model is designed on. This is very important especially true if model order selection is considered. However the specific model quality is of little importance here and the model order is fixed to max two, while the verification of the choise between models is more important.

The verification is therefore focused primarily on benchmarking the models and it is preformed on the same data set as the models was constructed from. This is mostly for practical reasons, since there is a high degree of non-linearity in the process, a different step may indiacte that a generally good model is insufficient to describe the system and a frequency experiment will not likely work due to non-linearities.

4.2.1 Model benchmarking

The verification done here consists of residual analysis as presented in (Ljung, 2004), on page 367, the residuals are the predition errors of the model output, see equaiton (4.6). This consists of investigating the correlation between the model prediction error and the input signal, see equation (4.7). Ideally these signals are totally uncorrelated, in which case the correlation would consist of normally distributed values with a mean of zero and a variance given by equation (4.8).

$$\varepsilon(t) = \varepsilon(t, \hat{\theta}_N) = y(t) - \hat{y}(t|\hat{\theta}_N) \tag{4.6}$$

$$\hat{R}_{\varepsilon u}(\tau) = \frac{1}{N}\sum_{t=1}^{N} \varepsilon(t-\tau)u(t), \quad |\tau| \le M \tag{4.7}$$

$$P_r = \frac{1}{N}\sum_{k=-\infty}^{\infty} R_\varepsilon(k) R_u(k) \tag{4.8}$$

In figure 4.4 the residuals for the models are plotted in a comparative way. This figure supports what was suggested in figure 4.3. However, the residuals are supposed to be normally distributed with a mean of zero, which implies that the second order model designed in System identification toolbox is the one producing best results, at least with reguards to the data used in the modelling process.

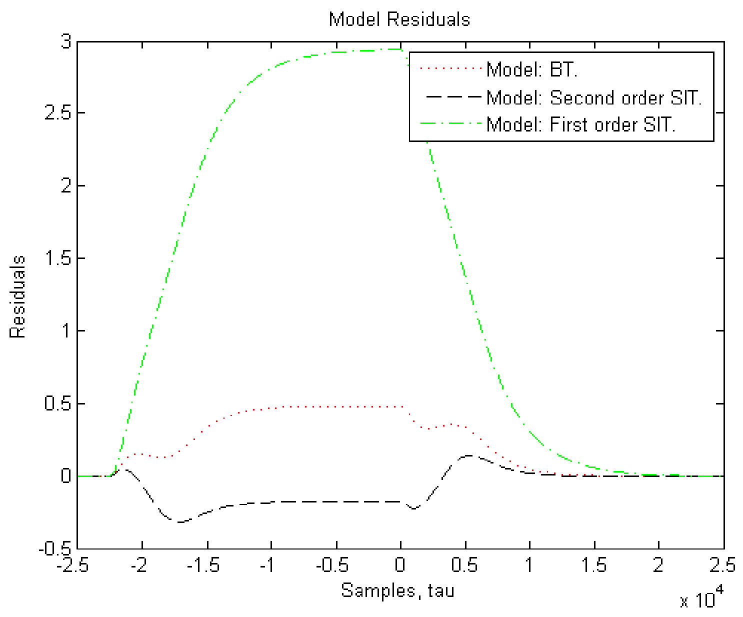


figure 4.4: The residuals of the designed models. The closer they are to zero the better the model prediction is. The residuals are taken from the same data set as the models are design from.

Usually the residuals are plotted together with the allowable variance limits, given by three times the square root of equation (4.8), in order to show a measure of acceptable deviation from zero. In figure 4.5 the second order SIT model is plotted in such a fashion. According to (Ljung, 2004) page 367, high deviation from zero for negative τ (tau in figure 4.5) indicates that data was collected under feedback and not that the model is unable to capture the system dynamics.

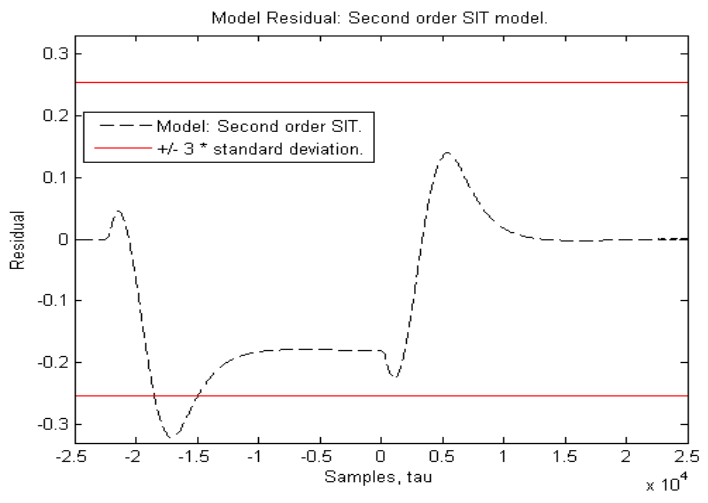


*figure 4.5: Residuals of the second order model designed with SIT, plotted together with the lines +/- 3 * √(P_r). where P_r is given by equation (4.8). The high level of deviation for negative tau comes from that the data was collected under feedback.*

4.3 Model completion and noise sources

SINCE THE MODEL only describes the velocity dynamics of the simulated vehicle, the model now needs to be put into proper physical context. Then it will instead descride the distance to a leading vehicle. In order to achieve this, the velocity of a leading vehicle also needs to be included.

To describe the change of distance between two points, the difference between the speed of the farthest point and the speed of the closest is computed and integrated. This is applied to the models as shown in figure 4.6.

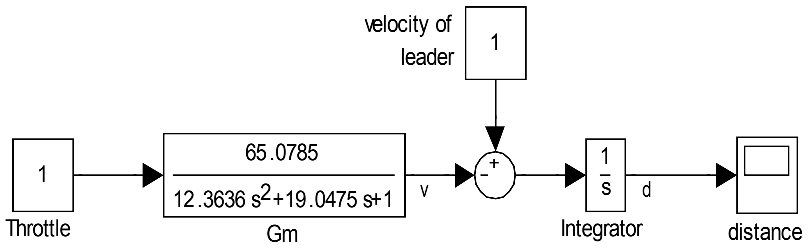


figure 4.6: Block diagram representation of the extension of the velocity model. Implementation done in Matlab/Simulink.

The model now becomes a two input, two output grey-box model, described with the transfer-functions given in equations (4.9) to (4.12).

$$G_{du} = \frac{distance}{thottle} = -\frac{G_m}{s} \tag{4.9}$$

$$G_{dvl} = \frac{distance}{velocity\ of\ leader} = \frac{1}{s} \tag{4.10}$$

$$G_{dvl} = \frac{velocity\ difference}{throttle} = -G_m \tag{4.11}$$

$$G_{dvl} = \frac{velocity\ difference}{velocity\ of\ leader} = 1 \tag{4.12}$$

This model only describe the linear distance and velocity and do not takes into account the effects of horizontal direction or velocity of the cars, see figure 4.7. The sensor presented in chapter three measures the norm of the vector v_l when returning the velocity of the leading vehicle. That velocity is not always equal to $\dot{d}$. The difference between v_l and $\dot{d}$ (below called measurement error) depends on the curvature of the road, which is deterministic (given by the road) but unkown and therefore treated as stochastic. The curvature is assumed to be a random-walk process, meaning that low frequencies will dominate the measurement error caused by the curvature of the road.

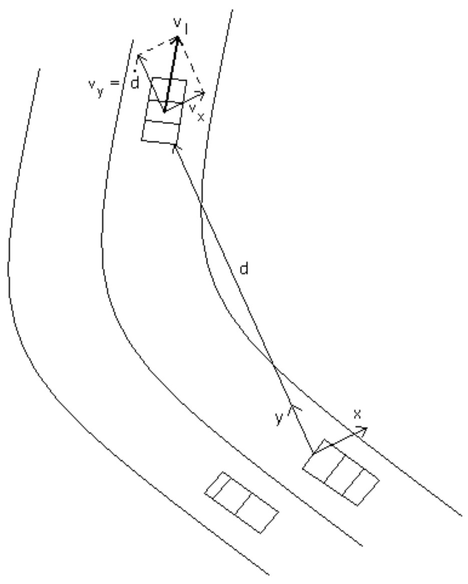


figure 4.7: An illustration of how the road curvature impact the measured velocity with what is assumed to be random-walk noise. Here d is the scalar distance to the leading vehicle, y always points in the direction of the leading vehicle, v_l is a vector representing th e speed of the leading car.

The same type of measurement error occurs when the curvature of the road becomes three-dimensional, which happens when the simulated environment has a hilly characteristics.

Chapter 5

Control Design

IN THIS CHAPTER the ACC controller is to be computed and analysed. The controllers considered are of PID compensator type with feedback of distance or a combination of distance and velocity difference. First the control loops are formulated, after which the controllers are computed and tested, both on the model and on the offline simulator. Finally a controller taken from litterature is presented and compared with one of the designed controllers.

5.1 Control loop formulation

WHEN FORMULATING A control loop it is important to consider the physical model. In the considered case this is presented in figure 4.6 on page 31. In formulating the control error, it was deemed logical to use negative feedback with a positive reference value, see figure 5.1.

The distance reacts with opposite sign to the speed of a following car. For a case of equal initial velocity between leader and follower, an incrementation of the velocity of the follower will, naturally, decrease the distance. This means that in order to have negative feedback , the contoller needs to react with opposite sign to the formulated distance error.

One limiting factor to the controller is that in reality, the vehicle has a very small control range, from minus one to plus one. This comes from that the velocity of the modelled vehicle is controlled using brake and throttle levels respectively. Since no model of the braking process has been done and it is known that the braking is a much faster process than accelerating, this was simulated by simply muliplying negative parts of the control signal by ten.

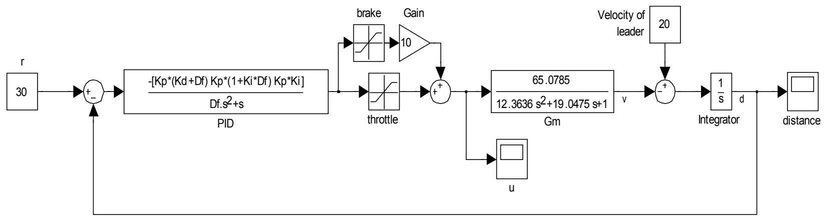


figure 5.1: Simulink model used for simulations of simple PID controllers. Braking simulated by taking negative part of control signal and multiplying it by ten. Limits to brake and throttle signals are set to 0.3 and 0.7 respectively in order to achieve comfort.

One way to add dynamics and increase stability is to add feeback of the velocity difference between the leading vehicle and the simulated car. Thereby making the de facto reference value sensitive to velocity in accordence with equation (5.1). This is simulated with the simulink model shown in figure 5.2.

$$r_v = r - K_v v_D \tag{5.1}$$

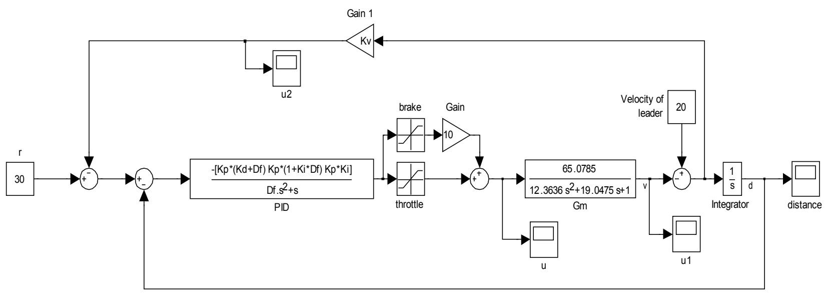


figure 5.2: Simulink model of the control loop using external feedback to make the reference value sensitive to the speed difference.

The closed loop system in figure 5.2 is a simplification (where the noise is ignored) of equation (5.3) below. The controller, the PID block in the figure, is assigned to the transfer function F and the system model is assigned to G.

$$V_D = sD \tag{5.2}$$

$$D = \frac{1}{s}(V_l + GF(R - D - V_D - W)) \tag{5.3}$$

(5.2) in (5.3) $\Rightarrow$:

$$D\left(1 + \frac{GF}{s}(1+s)\right) = \frac{1}{s}(V_l + GF(R - W)) \tag{5.4}$$

$\Leftrightarrow$

$$D = \frac{1}{s + GF(1+s)}V_l + \frac{GF}{s + GF(1+s)}(R - W) \tag{5.5}$$

In equation (5.5) the transfer-functions corresponding to the sensitivity and the complementary sensitivity functions are visible. They decide the performance of the control system but they do not have the same simple connection to stability in this controller as they do in one with a single feedback loop. This comes from that the loop is no longer a single expression, see equations (5.20) and (5.22) on page 45.

There is a few requirements that needs to be fulfilled for the closed loop system: The closed loop needs to be stable, which will be checked after the control synthesis using the Nyquist criterion. The ride also needs to be comfortable for the passengers, in order to allow for user confidence in the ACC, which will also be investigated after the control synthesis, by the use of simulations.

The goal of persistent error elimination is centered around elimination of the distance error, defined in equation (5.6).

$$E_D = R - D \tag{5.6}$$

$$F = \frac{\bar{F}}{s} \quad \text{where: } \bar{F} \neq 0 \text{ when } s \rightarrow 0 \tag{5.7}$$

(5.6) in (5.3) with $W = 0 \Rightarrow$:

$$D = \frac{1}{s}(V_l - GF(E_D - V_D)) \tag{5.8}$$

(5.2) and (5.7) in (5.8) $\Rightarrow$:

$$\begin{aligned} E_D &= \frac{1}{GF}V_l - \frac{s}{GF}D + V_D = \\ &= \frac{s}{G\bar{F}}V_l + s\frac{G\bar{F} - s}{G\bar{F}}D \end{aligned} \tag{5.9}$$

In order to calculate the persistent error the final value theorem, see (Lennartson, 2002) p. 40, is used.

$$\begin{aligned}\lim_{t\to\infty} e_D(t) &= \lim_{s\to 0} \; sE_D = \\ &= \lim_{s\to 0} \; s\left(\frac{s}{G\bar{F}}V_l + s\frac{G\bar{F}-s}{G\bar{F}}D\right) = 0 \end{aligned} \qquad (5.10)$$

The presistent distance error is, according to equation (5.10) eliminated by the controller used in figure 5.2 on page 34.

5.2 Control synthesis and approach analysis

THE CONTROL SYNTHESIS is of intuitive type, based on the control signal limits, first formed for a single feedback PID controller and then for a PID controller with feedback of both distance and difference of velocity. This controller approach will have the advantage of actually considering the limits of the simulator (the non-linear, simulated “reality”), but it may not be as good (regarding performance, robustness etc.) as a controller designed from a frequency based approach. The algorithm is exemplified with a distance feedback controller.

The algorithm of the intuitive PID approach

1. Using the physical control signal limit (plus one to minus one) to decide on how big the control range of a proportional controller should be. An appropriate maximum error was chosen to 12 meters, see figure 5.3, corresponding to a $K_P = 1/12 \approx 0.08$. This is by no means a hard limit, only a theoretical limit to were full throttle will be applied without regards to integratory and derivatory outputs. The throttle and brake limitations are tightened further in order to achieve a more comfortable ride.

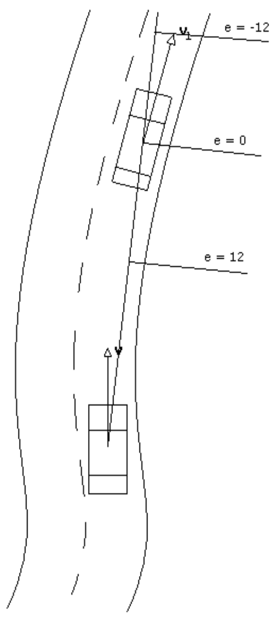


figure 5.3: Illustration of how the proportional control parameter was selected. At an error of minus 12 meters, a control signal of minus one (full brake) is chosen. Because of the negative feedback, the error increases in the opposite direction of the vehicles. (units in meter)

2. Now a derivative gain is to be selected, generally this is done in order to achieve desired control speed. However since this is to be filtered, a filter time constant is also to be selected or more precisely a factor c according to: $D_f = c * K_D * K_P$ is to be selected (a filter with a filter time expressed as a factor of the total derivative gain). I order to select the gain a simulation of the model in figure 5.1 with initial leader velocity of 20 m/s and initial distance of 60 m, was executed. An appropriate derivative gain was selected from the simulation to: $K_D = 2$. The constant c is chosen to be 0.5 thereby achieving a filter cutoff frequency of $(Kp)^{-1} = 12.5$ rad/s.

3. Finally, the Integration gain K_I is selected as small as possible, to suppress the magnitude of overthrows, while till having acceptable $t_{5\%}$. The choise of $K_I = 0.0015$ gives a simulated $t_{5\%} = 35$ s (under brake and throttle limits of 0.1 and 0.7 respectively and under the same initial conditions as the simulation in step 2).

5.2.1 Distance feedback

WHEN FOLLOWING THE above presented algorithm in creating a controller with negative feedback of the distance, a controller of PID structure was synthesied. The resulting controller will have the apperence of equation (5.11) and the nummerical approximation of equation (5.12):

$$F_{PID}(s) = K_P\left(1 + \frac{K_I}{s} + \frac{K_D s}{D_f s + 1}\right) \approx \tag{5.11}$$

$$\approx \frac{0.1664 s^2 + 0.08001 s + 0.00012}{0.08 s^2 + s} \tag{5.12}$$

This controller gives a loop expression according to equation (5.13):

$$L(s) = F(s)\, G_m(s)\, \frac{1}{s} \approx \tag{5.13}$$

$$\approx \frac{10.83 s^2 + 5.207 s + 0.007809}{0.1891 s^5 + 3.887 s^4 + 19.13 s^3 + s^2} \tag{5.14}$$

In order to investigate the closed loop stability, the loop expression is to be checked against the Nyquist criterion. Due to the fact that the loop is subject to more than one integration, the simplified criterion can not be used. Instead the complete criterion is to be applied.

According to the Nyquist criterion, see (Lennartson, 2002) on page 240, a system is stable if equation (5.15) is fulfilled.

$$Z = P + N = 0 \tag{5.15}$$

Where:

Z = number of transmission zeros in right half of the complex plane for 1 + L(s).

P = number of poles in right half of the complex plane for L(s).

N = number of clockwise rotations around the point (-1, 0) in the complex plane for the depiction of L(s) when s is assigned the Nyquist contour.

The Nyquist criterion requires the Nyquist contour to be applied to the process. The contour is defined in four sections: the positive complex axis from zero to complex infinity, see equation (5.16), a half circle infintly far away from complex infinity to minus complex infinity (5.17), the negative complex axis from minus complex infinity to zero (5.18) and finaly a half circle infinitly close to zero from negative (complex) zero to positive (complex) zero (5.19) , see (Lennartson, 2002) page 239.

The contour is applied to a process by letting the Laplacian operator s be replaced with equation (5.16) to (5.19) and plot the results in a Nyquist diagram.

$$s = j\omega,\ 0 < \omega < \infty \tag{5.16}$$

$$s = Re^{j\theta},\ (clockwise\ rotation) \\ \frac{\pi}{2} \geq \theta \geq -\frac{\pi}{2},\ R \to \infty \tag{5.17}$$

$$s = j\omega,\ -\infty < \omega < 0 \tag{5.18}$$

$$s = re^{j\theta},\ (counterclockwise\ rotation) \\ -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2},\ r \to 0 \tag{5.19}$$

This was done in Matlab and in figure 5.4 the resulting Nyquist diagram can be viewed. Equation (5.16) is represented by the bottom line, going towards zero. Equation (5.17) becomes an infinitly small circle (two integrations) around zero. Equation (5.18) is shown as the top line, growing towards minus infinity and plus complex infinity. Equation (5.19) is not possible to see, since it becomes an infinitly large circle from the “end-point” of the top line, to the “start-point” of the bottom line in the clockwise direction (due to integration).

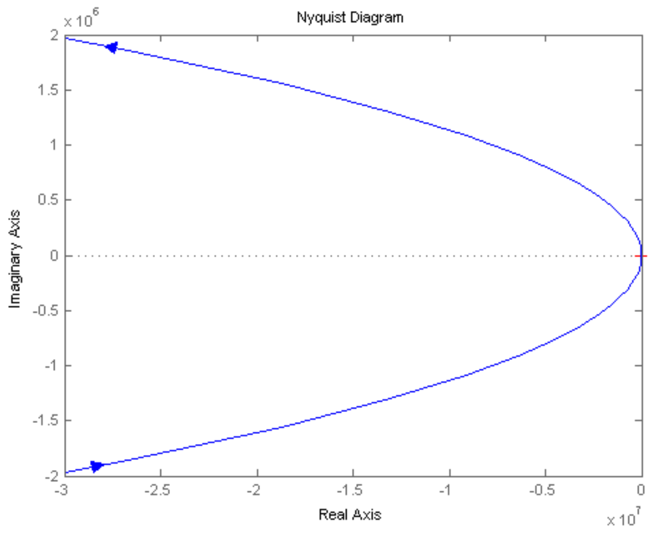


figure 5.4: The Nyquist diagram of the circuit loop of the control loop, in a general overview.

In figure 5.5 the Nyquist diagram in close up around the minus one point is shown. In this figure it can clearly be seen that no small scale rotations around the minus one point occurs.

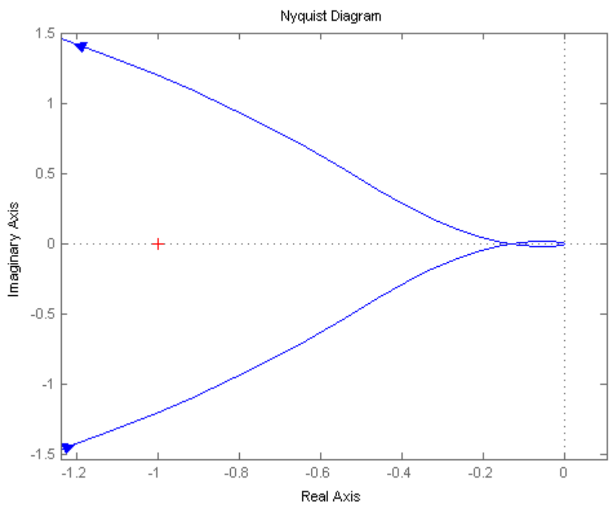


figure 5.5: The Nyquist diagram in close-up around minus one

In Table 5.1 the data needed to chek the citerion is presented. Z and P are both given by equation (5.14) and computed in Matlab. N is given by figure 5.4 and (5.5), following the text explaining them. Since the collected data satisfy equation (5.15) the linear approximation of the system is concluded to be stable and some of the sulutions to the nonlinear system is therefore assumed to be stable as well.

Table 5.1: *The constants used in the Nyquist criterion. They obviously satisfy equation (5.15)*

Constant	*value*
Z	0
P	0
N	0

This controller was tested in the Simulink model in figure 5.1 which produced the results shown in figure 5.6. The plot suggests that the controller is most likely fast enough, but it also shows a "slinky" effect around 30 seconds into the simulation which is not acceptable under the comfort requirement. No one would use an ACC unless it provides a more comfortable ride than what is suggested.

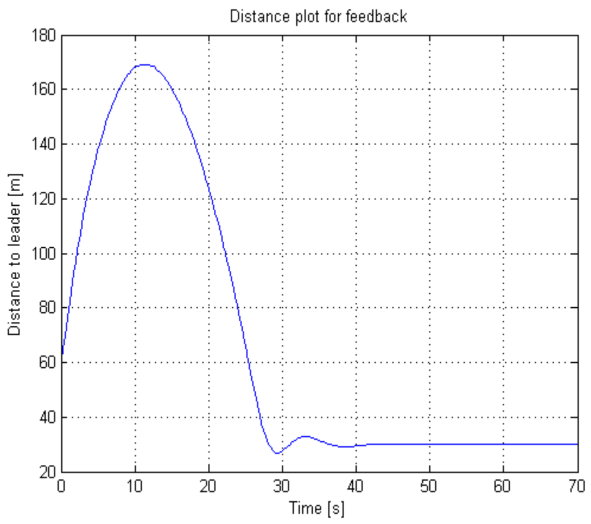


figure 5.6: Plot of the distance response to a leading vehicle speed of 20 m/s with an initial distance of 60 m. The simulated car starts with initial speed of 0 m/s.

5.2.2 Distance and speed difference feedback

IN ORDER TO eliminate the “slinky” effect, the control loop is complemented with a negative feedback of the velocity difference in accordance with the structure in figure 5.2. The three steps of the intuitive control design above is only changed in the first step. This step is complemented with the choise of an appropriate K_V. The new controller constants are shown in Table 5.2.

Table 5.2: *Control constants selected with step 1-3 of the algorithm on pages 36 to 37. With the KV selected as unity.*

Constant	*Assigned value*
K_V	1
K_P	0.08
K_I	0.0015
K_D	3.59
c	1.2

Using this controller gives the experiment presented in figure 5.7 a $t_{5\%}$ of approximately 35 seconds, wich is the same settling time as with the controller without the extra feedback. The reason that the “slinky” behavior is avoided with this controller is that the complex poles otherwise experienced by the closed loop system, are damped.

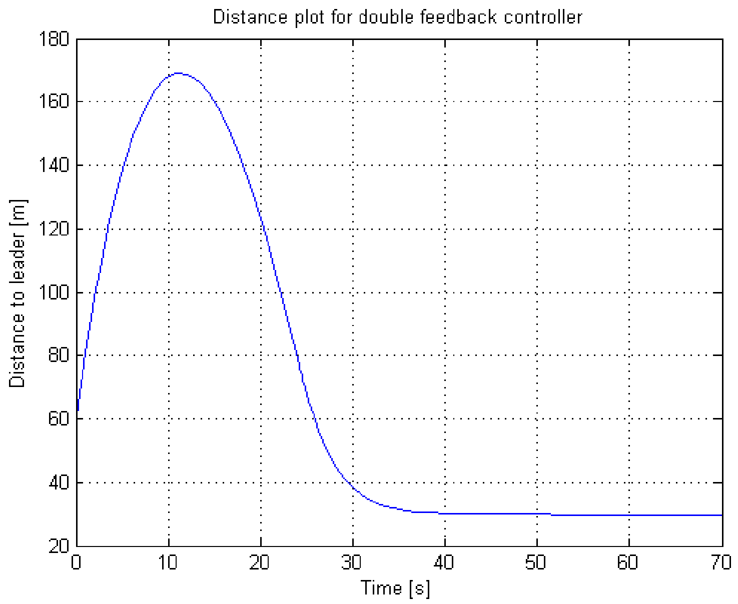


figure 5.7: Plot of the distance response to a leading vehicle speed of 20 m/s with an initial distance of 60 m. The simulated car starts with initial speed of 0 m/s and has a reference value of 30 m. Simulation of the Simulink model in figure 5.2.

The control signal activity is an important factor in control engineering, especially in the case of sensitive actuators that are subject to wear or when a economically or environmentally expensive fuel is used. In the studied case with a software based simulator this is not as important, but may still be worth looking at since it is closely connected with sensitivity to measurement noise. In figure 5.8 the control signal activity of a simulation in the offline version of the simulator with noisy, but filtered measurements. The controller has damped the high frequency part of the added measurement noise, but is unable to damp the low frequency part. If the derivative filter time constant is increased, the control signal activity is reduced at the cost of a slower system.

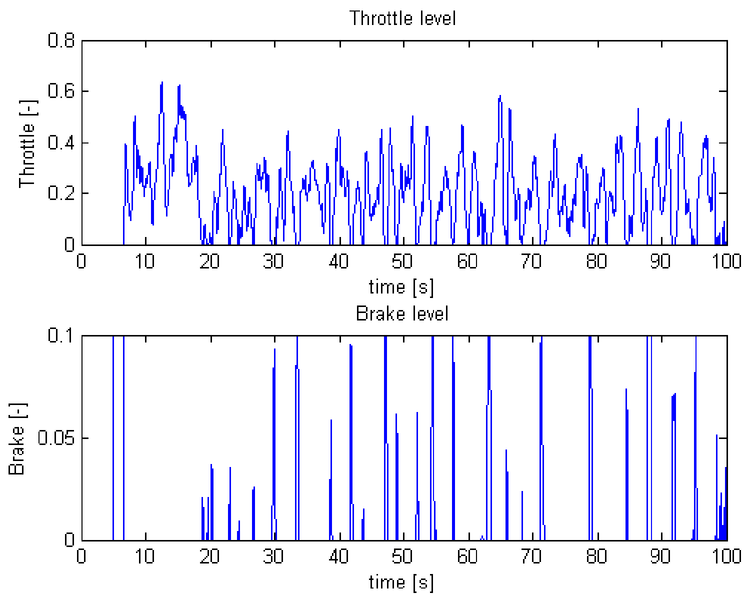


figure 5.8: The control signal activity of the discussed controller under a simulation with a leader starting with 0 m/s and accelerating to 20 m/s (72 km/h) in 16 s and holding that velocity then on. The simulated car is first accelerated to 16.67 m/s (60 km/h) whereafter the ACC is engaged.

In figure 5.9 the control signal from the same simulation but without the added noise, is presented. The signal in that figure seams to be a good approximation to the mean of figure 5.8. The most important difference is the Brake signal (the negative part).

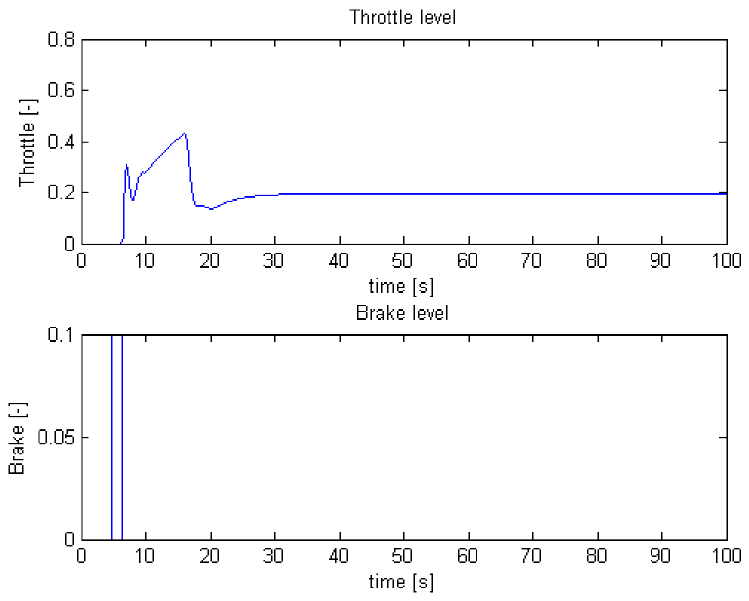


figure 5.9: The same simulation as in figure 5.8 but without the added measurement noise. This illustrates the controllers sensitivity to measurement noise. In general the mean value of the noisy signal in figure 5.8 seams not to deviate much from the signal in this figure.

In figure 5.10 the Bode diagram of the controller can be seen. It has a characteristic PID apperence with the infinite low-frequency response and two fairly constant gain levels.

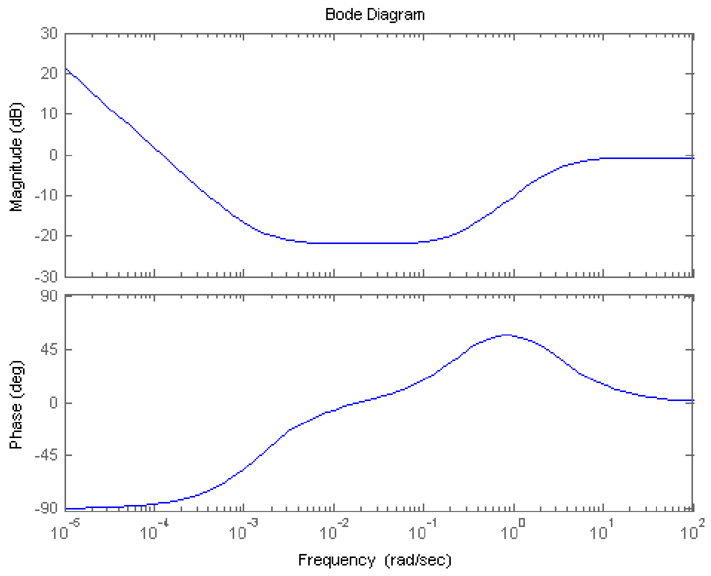


figure 5.10: The Bode diagram of the controller given by Table 5.2.

As can be seen in figure 5.10 the controller has a high-frequency gain of just below one (zero dB) which limits the control signal activity and the sensitivity against high frequency measurement noise, at least for singel loop PID controllers according to (Lennartson, 2002) on page 285.

This controller has two controlled variables, velocity difference as well as distance, and therefore it has no single circuit loop expression. Instead it has two circuit loops, depending on where the circuit is broken up. Equation (5.20) describe the circuit loop if the loop is broken at the control signal u and (5.22) if it is broken at the primary controlled variable d.

$$L_u = F(\frac{1}{s}G_m + G_m) = \frac{F\,G_m}{s}(1+s) \approx \tag{5.20}$$

$$\approx \frac{25.15s^3+31.54s^2+6.404s+0.009587}{s^5+10.96s^4+23.81s^3+1.228s^2} \tag{5.21}$$

$$L_d = \frac{1}{s}\left(\frac{FG_m}{1+FG_m}\right) \approx \tag{5.22}$$

$$\approx \frac{25.15s^2+6.395s+0.009587}{s^5+10.96s^4+48.95s^3+7.622s^2+0.009587s} \tag{5.23}$$

In order for this system to be stable, both L_u and L_d must represent stable systems. L_d can is this case be proven to be stable using the simplified Nyquist criterion (only one integration and no poles with positive real part), while L_u requires the non-simplified Nyquist criterion to be fulfilled. In figure 5.11 the Nyquist diagram for L_u is presented. Since L_u is integratory (equation (5.19) changes direction) the curve does not circulate the point minus one and since all poles and zero crossings have negative real part, the system represented by L_u is stable. The L_u expression is studied in conjunction with robustness against model errors and L_d in the sensitivity against measurement noise. They naturally have different margins and these are given in Table 5.3.

Table 5.3: *The gain and phase margins of the loop expressions.*

Margin	L_u	L_d
A_m, [-]	-inf.	19.8364
φ_m, [°]	99.5752	74.2905

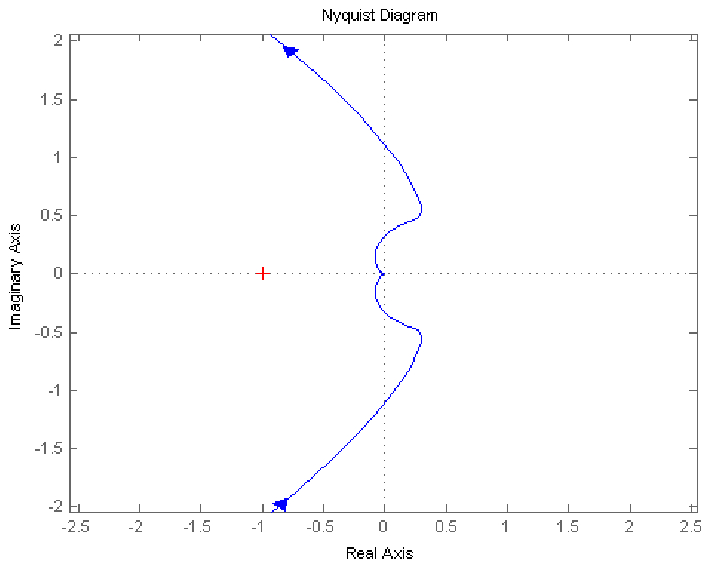


figure 5.11: Nyquist diagram of the L_u circuit loop according to equation (5.21).

The simplfied Nyquist criterion say that a feedbacked system is stable if the point (-1,0) is to the left of (and below) the curve L(jω) for all ω. In figure 5.12 the Nyquist diagram of L_d is shown. As can be seen in the figure, the system represted by L_d is stable according to the simplified Nyquist criterion.

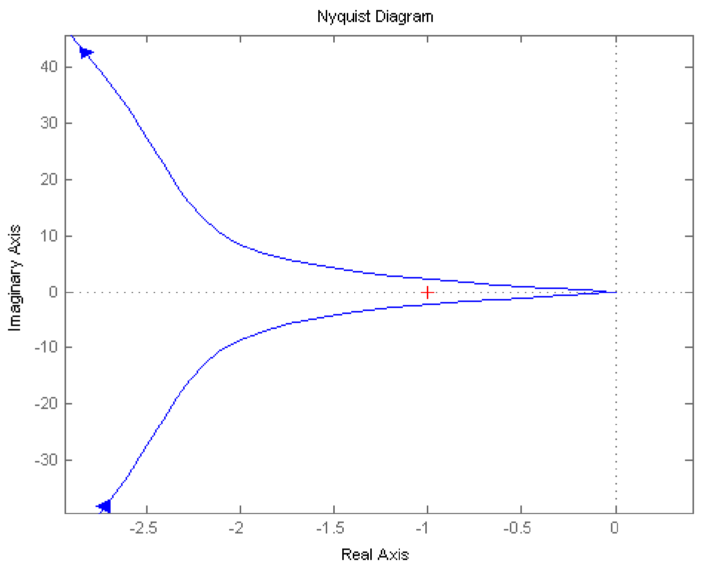


figure 5.12: Nyquist diagram of the L_d circuit loop, according to equation (5.23).

5.2.3 A benchmarking controller

As a comparative with existing controllers the Autonomous Intelligent Cruise Control, AICC, from (Ioannou, 1993) is introduced. This is based on a safety distance policy and will therefore not have formal error elimination. However the stationary “error” will come from the mentioned safety distance policy and allow for closer spacing between cars and thereby a higher traffic flow. The safety distance policy is given according to equation (5.24).

$$\delta(t) = d - (L + S_o + \lambda v(t)) \tag{5.24}$$

Here d stands for distance, L is the car length, S_o is the minimum safe distance (a reference variable), λ is the slope of the velocity dependent part and v is the cars velocity. The δ(t) is the policy deviation, which should be zero under stationary conditions. The control law presented in the article is given by equation (5.25) and is based on feedback linearization.

$$u(t) = \frac{1}{\alpha(\dot{x})}(c(t) - b(\dot{x}, \ddot{x})) \tag{5.25}$$

Where α, c and b are given by equations (5.26), (5.27) and (5.29) and comes from the feedback linerization technique.

$$\alpha(\dot{x}) = \frac{1}{m\tau(\dot{x})} \tag{5.26}$$

$$c(t) = C_p\delta(t) + C_v\dot{\delta}(t) + K_v v(t) + K_a a(t) \tag{5.27}$$

The design constants in equation (5.27) are given in (Ioannou, 1993), as $C_p = 4$, $C_v = 28$, $K_v = 0$, $K_d = -0.04$ and $\dot{\delta}(t)$ is taken as the time derivative of equation (5.24) which becomes equation (5.28), where v_d means the velocity difference.

$$\dot{\delta}(t) = v_d - \lambda a(t) \tag{5.28}$$

$$b(\dot{x}, \ddot{x}) = -2\frac{k_d}{m}\dot{x}\ddot{x} - \frac{1}{\tau(\dot{x})}\left(\ddot{x} + \frac{k_d}{m}\dot{x}^2 + \frac{d_m(\dot{x})}{m}\right) \tag{5.29}$$

Where m is the vehicle mass, k_d is the aerodynamic drag coefficient, $\tau(\dot{x})$ is the engine time constant and $d_m(\dot{x})$ is the mechanical drag. An implementation of this controller can be seen in figure 5.13.

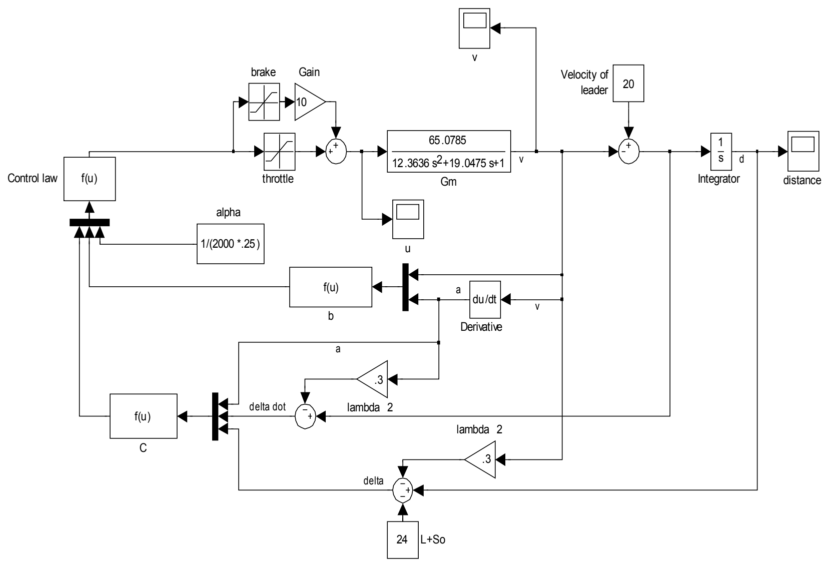


figure 5.13: An implementation of the AICC discussed in the text. Note the derivative block necessary in the controller.

This controller is more complex than the two previously considered and with increased model accuracy the complexity of the AICC will increase further. It requires the model to provide data about the distance to the leading vehicle, the velocity difference, the simulated car´s velocity and acceleration. Since the model, G_m, does not provide the car´s acceleration, the model needs to take the derivative of the velocity which increases the computation time.

In figure 5.14 the results of a simulation of the different control system is presented. The difference is that the AICC do not have any proper reference value, instead the minimum safety distance L + S_0, see figure 5.13, is set to 24 m, which gives a total distance of approximetly 30 m.

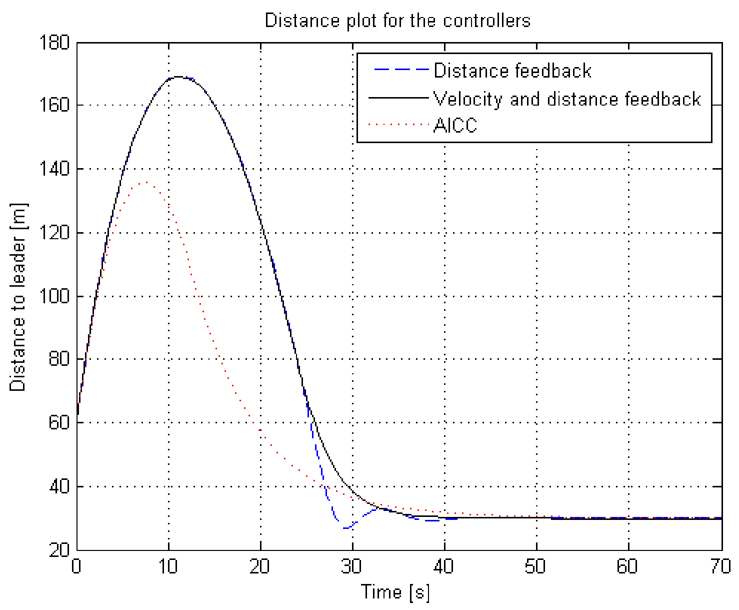


figure 5.14: A simulation of the different controllers.

This controller gives a $t_{5\%}$ of approximately 40 seconds and it completely eliminates the slinky behaviour of the simpler controllers considered.

An interesting control property is the control signal activity especially when the measurement signal is subject to noise. A simulation in the off-line simulator with noise added to the simulated leader was carried out and the resulting control signal is presented in figure 5.15. The control signal activity in this simulation is very high and it is likely that the high activity is rooted in the controller´s built-in model flaws, since the same simulation with the noise removed was done and the high activity remained. If the physical modelling done is improved less constant approximations and so forth, it is likely that the controller will preform better in all aspects including control signal activity.

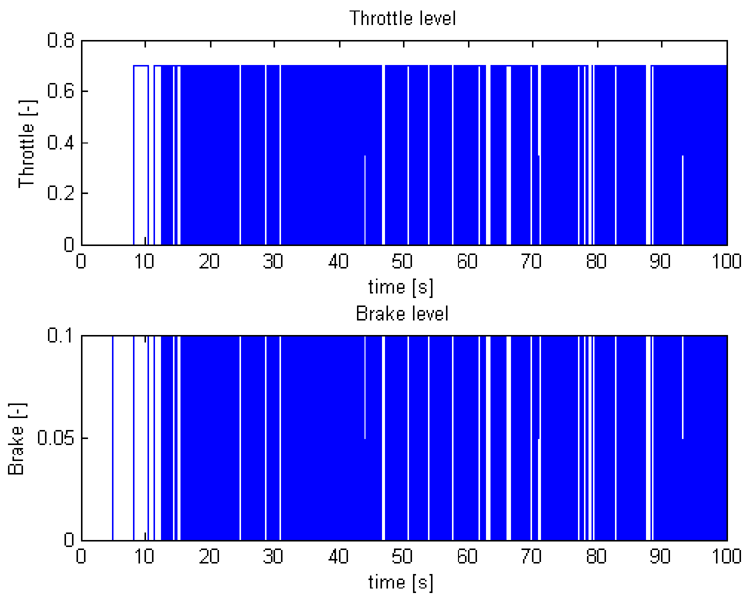


figure 5.15: The resulting control signal from a simulation with a simulated leader and carried out with the same conditions as the simulation in figure 5.8.

5.2.4 Simulations of stopping and cut-ins

Two important traffic situations that can occur in reality, outside of the simulators otherwise considered in this paper, is if the leading vehicle has to preform an emergency stop and if a third vehicle overtakes your car and cut-in directly in front of you. First a simulation of the emergency stopping capabilities of the controller with feedback of distance and velocity (PID) is compared with that of the AICC, from (Ioannou, 1993).

The simulation starts with the leading vehicle at a distance of 100 m at standstill. Then the leader is accelerated to 20 m/s with a maximum acceleration of 0.4 g. After 100 s the leader is deccelerated back to standstill with a maximum acceleration of 0.8 g. The stopping distance is noted with full braking allowed and with braking limited to maximum 30 %. The results of the simulation can be found in table 5.4 on page 52.

Table 5.4: *Results of a simulation of emergency stopping with two different controllers. $S_0 + l$ is the minimum safe distance of the AICC and the λ is chosen to 0.4. The reduced distance is the initial distance minus the distance directly after the emergency stopping is completed, when both vehicles are at standstill.*

Controller	Initial distance, [m]	Braking level, [-]	Reduced distance, [m]
PID	30	0.3	26
PID	30	1	10
PID	10	1	9
AICC	38 ($S_0 + l = 30$)	0.3	29
AICC	38	1	20
AICC	18 ($S_0 + l = 10$)	1	20

The PID controller seams to out-preform the AICC but this is most likely only an effect of the modelling of the AICC not corresponding very well with the simulated vehicle. However the controllers have at least comparable performance in their emergency stopping capabilities.

The cut-in simulations is done with an initial distance to the leading vehicle of 100 m and at standstill. The leading vehicle is thereafter accelerated to 20 m/s and after 100 s a negative step that brings the distance to 2 m is executed. This does not affect the velocity of the leading vehicle, which is assumed to have already adjusted its speed. During this simulation, the controllers are both allowed to use full brake. The results of the simulations are presented in figure 5.16 and figure 5.17

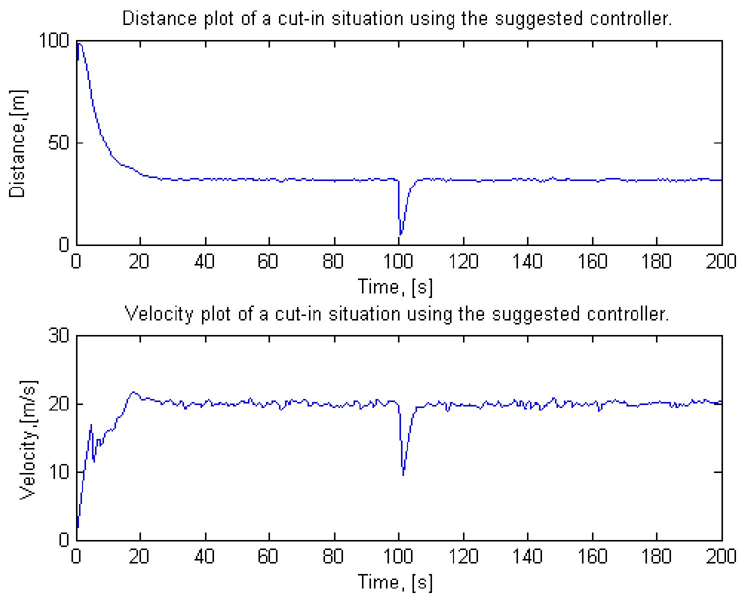


figure 5.16: The cut-in simulation for the PID controller.

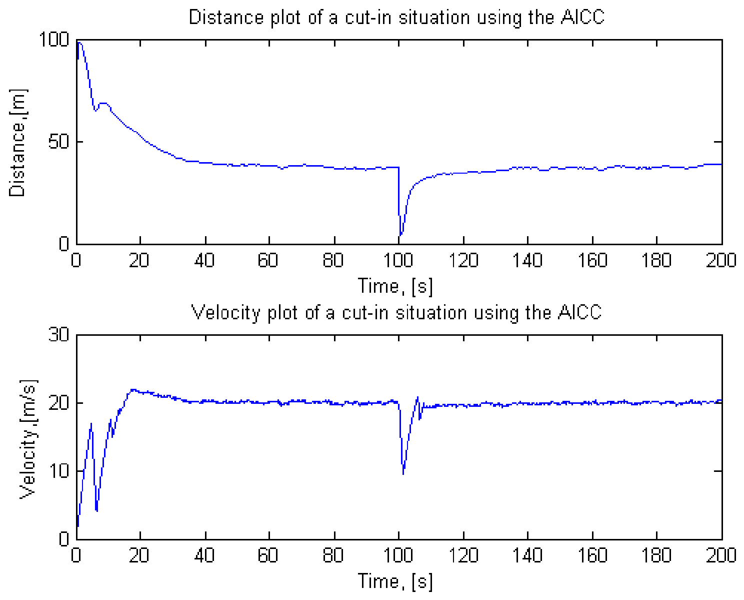


figure 5.17: The simulation of a cut-in situaiton for the AICC.

Both of the controllers handle this situation in a satisfactory way and collision is avoided.

Chapter 6

Implementations

HERE THE IMPLEMENTATIONS done in the different Simulink models are presented and discussed. First the offline version of the simulator model is considered. Thereafter the changes to the online version are presented.

6.1 Offline simulator

THE MOST APPERENT disadvantage of the offline simulator is that there are not any other vehicles, so there is no leaders to follow. In order to do any meaningful simulations with an ACC a leader is required and must therefore be simulated. Such a simulated leader is shown in figure 6.1.

The velocity of the leader is here simulated with a signal builder block in which the velocity can be given any desirable characteristics. To the basic velocity is added a mixture of pseudo-random noise in form of both integrated noise (random-walk noise) and a simple band limited white noise.

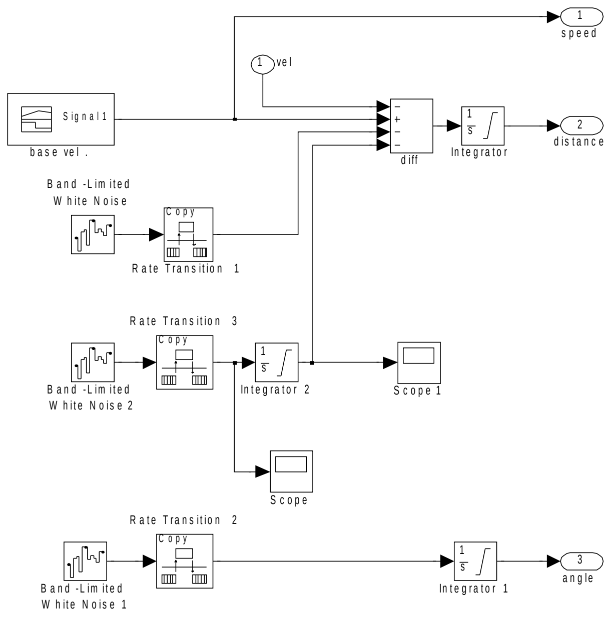


figure 6.1: The block used for simulating the radar output when following a leading vehicle. Implemented in the offline version of the simulink model.

The input simulation shown in figure 6.1 is sufficient for simulating the single loop control approach (the normal PID controller). In the other control approaches the speed-output was also given a additative band limited white noise. The angle is not used for anything, but is still there since the radar algorithm uses it and outputs it.

In figure 6.2 the implementation of how the simulated leader from figure 6.1 connects with the controller can be seen. The distance signal is simply connected to the appropriate input (creates a negative feedback) while the speed signal is used to create the velocity difference between the leader and the simulator car. Thereafter the velocity difference signal is subtracted from the reference value, thereby creating the sought after negative feedback.

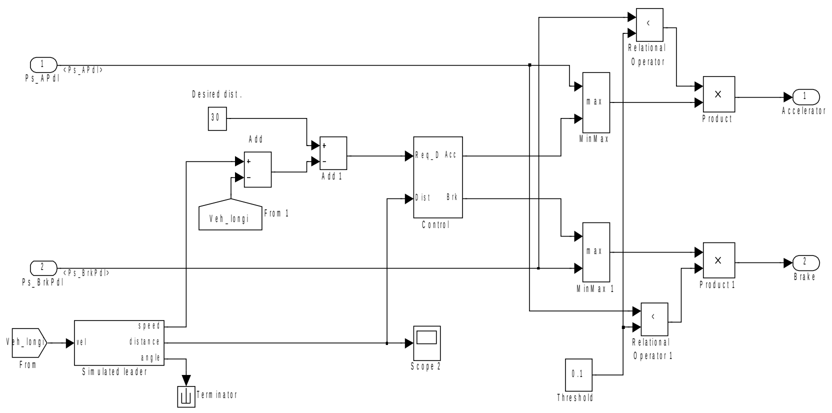


figure 6.2: The simulink model of how the simulated leader is connected with the controller and how they are connected with the throttle and brake.

In figure 6.3 the Control-block from figure 6.2 can be seen. The structure is used in both the single and double loop controllers.

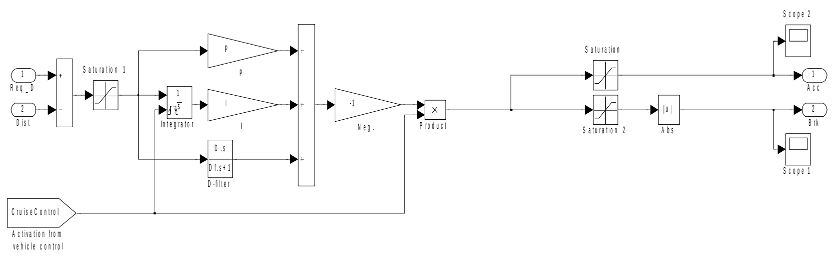


figure 6.3: The simulink model of how the controller is implemented. Corresponds to the control block in figure 6.2.

6.2 Online simulator

IN THE ONLINE simulator there is a few other requirements that needs to be fulfilled, as compared with the offline version. There is obviously no need for any simulated leader, as it is in the offline version, instead the input from the radar sensor in the graphics environment needs to be received and filtered. The most obvious difference to the offline version is however that the ACC must be able to pick up from the previosly existing Cruise Control as soon as a leader vehicle is found.

In figure 6.4 the implementation of the ACC is shown. The most significant changes, as compared with the previously existing Cruise Control model, are marked in gray. The controllers are switched when a leader is found or lost, see variable Leader found in figure 6.6. The blocks ACC Logics and Cruise Control Logics are basicly the same block and the only difference between the boolean signals ACCActive and CruiseControlActive is that the latter is false when the velocity is to low.

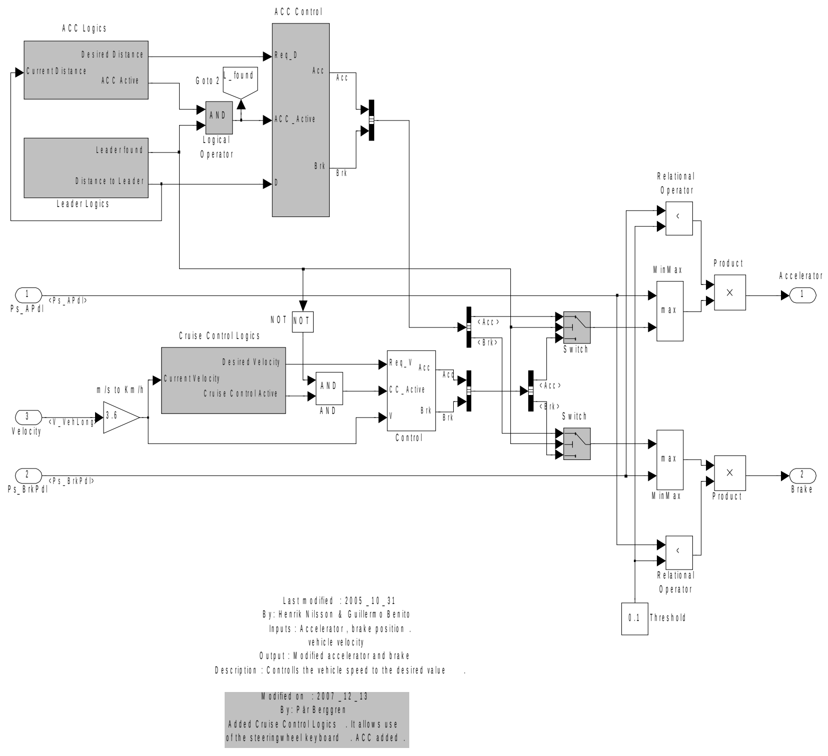


figure 6.4: An overview of the simulink model of the ACC implementation.

The reason for the regular Cruise Control to require that no leader is present, as opposed to use the ACC_Active input signal, is that the control switching should not be dependent of wether or not the cruise control or ACC is currently being used.

In figure 6.5 the contents of the block ACC logics from figure 6.4 can be seen. The block´s two outputs define the parts that make up the block: ACC Active and and Desired Distance. The state of the ACC Active signal is desided by a D-latch that only allows changes to the output if the brakes are active or if the cruise button is pressed and the brakes are not active at the same time. The D-latch updates the output with the logic complement to the brake active signal, thereby making sure that the brakes are not engaged when the ACC (same for Cruise Control) is activated.

The desired distance signal is computed as the reference value minus the velocity difference, see control approach with feedback of both velocity difference and distance on page 34. The reference value is set on the positive flank of an appropriate starting signal and can be modified by the blocks Incrementation and Decrementation of Desired Distance by pressing the plus and minus buttons on the steeringwheel.

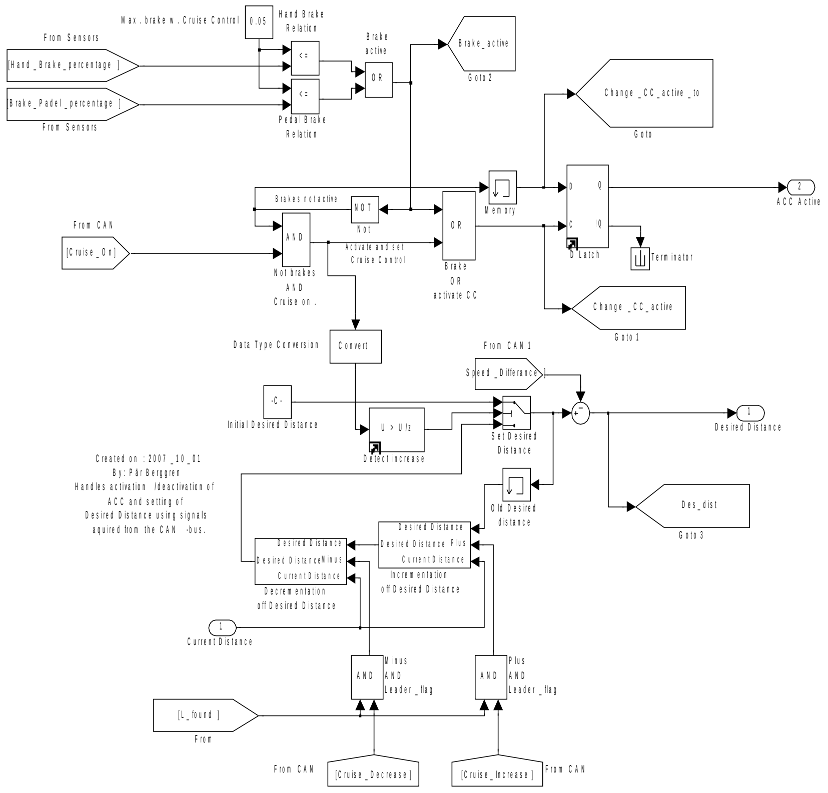


figure 6.5: The simulink model of how the block ACC logics from figure 6.4 is implemented.

The behaviour of the incrementation and decrementation of the desired distance and speed (for ACC and Cruise Control respectively) is given according to: the first possitive flank increases / decreases the desired speed (only applied to the cruise control) with a predetermined amount (set to 1 m/s) after the signal has been held high for a period of time (0.5 second) the desired signal is ramped with a slope of 5 units (m and m/s) per second. Finally when a negative flank is encountered, the desired signal is set to the current value of the measured distance or speed. The implementation of the behaviour can be found in figure 6.9.

In figure 6.6 the block Leader Logics is presented. This block keeps track of when an appropriate leading vehicle is aviable and filters the measured distance to this vehicle.

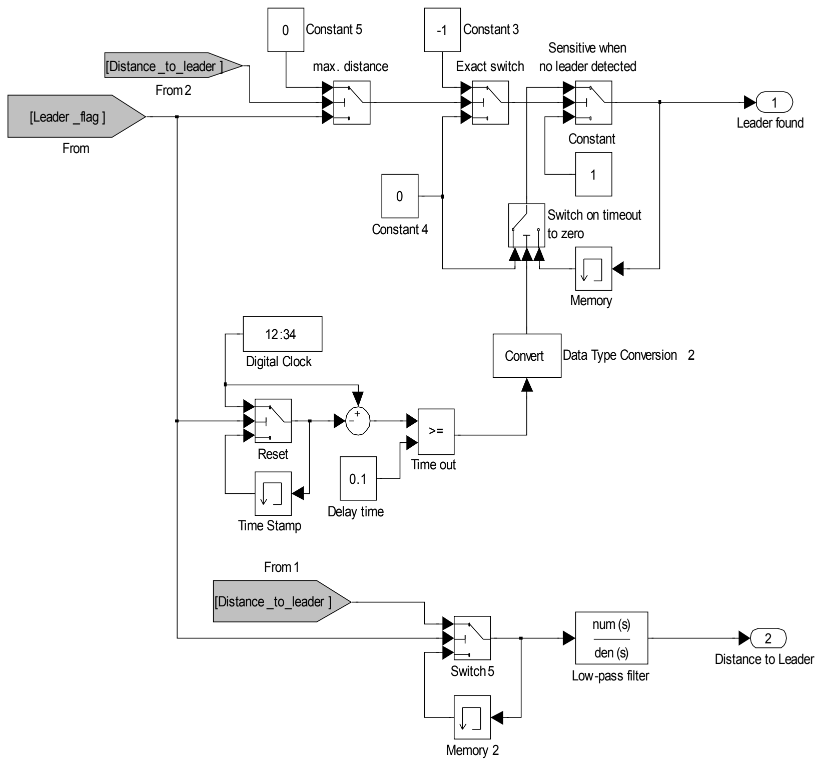


figure 6.6: Leader Logics

In figure 6.7 the ACC controller implementation is shown. It requires the ACC_Active signal to have been high for a period of time (0.5 seconds) before allowing the ACC to activate. This time delay prevents the abnormaly high derivative output associated with leaps in the reference value that often occurs with the activation of the ACC.

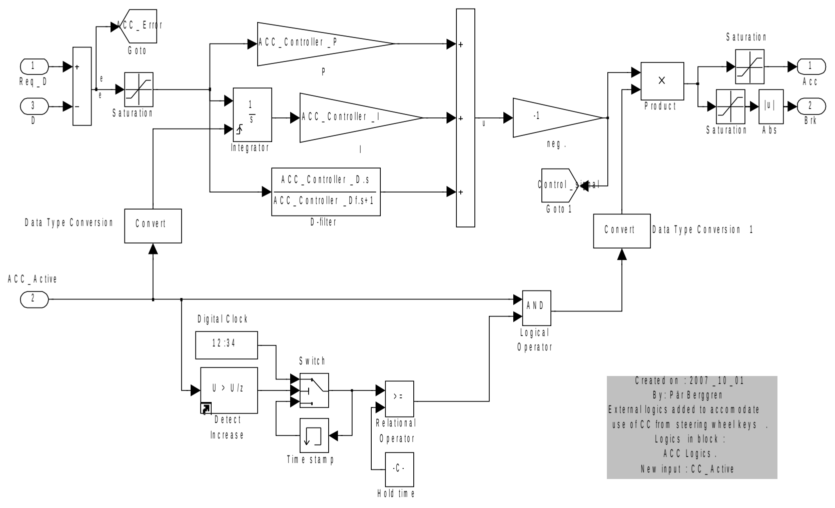


figure 6.7: ACC Controller

In figure 6.8 the Cruise Control Logics block found in figure 6.4 is shown.this block handles the activation of the cruise control and connects the desired velocity with the external CAN-bus controls.

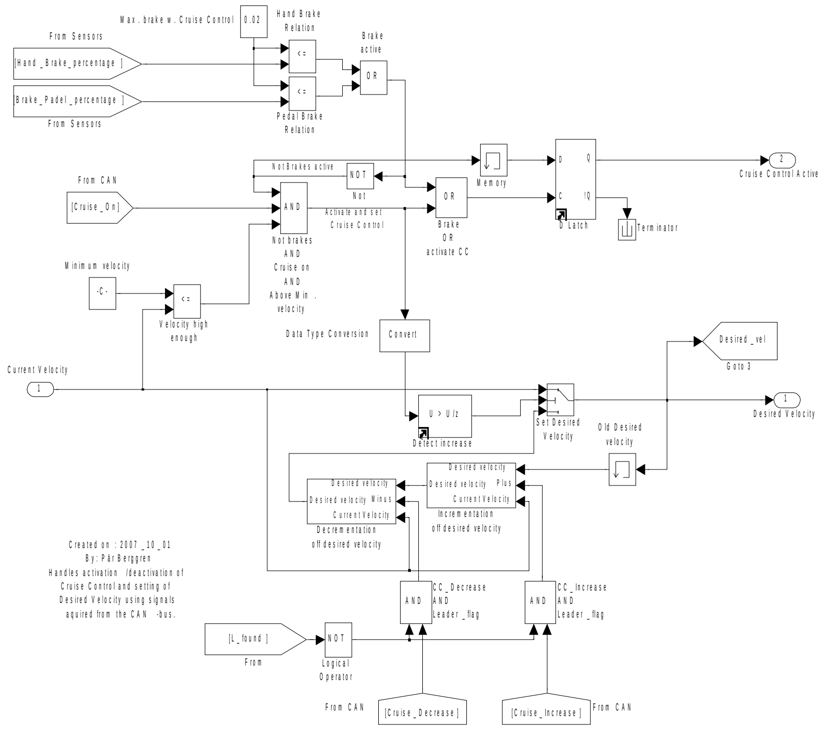


figure 6.8: Cruise Control Logics

The decrementation / incrementation of desired velocity blocks that can be seen in figure 6.8 and decrementation / incrementation of desired distance in figure 6.5 are all illustrated by figure 6.9.

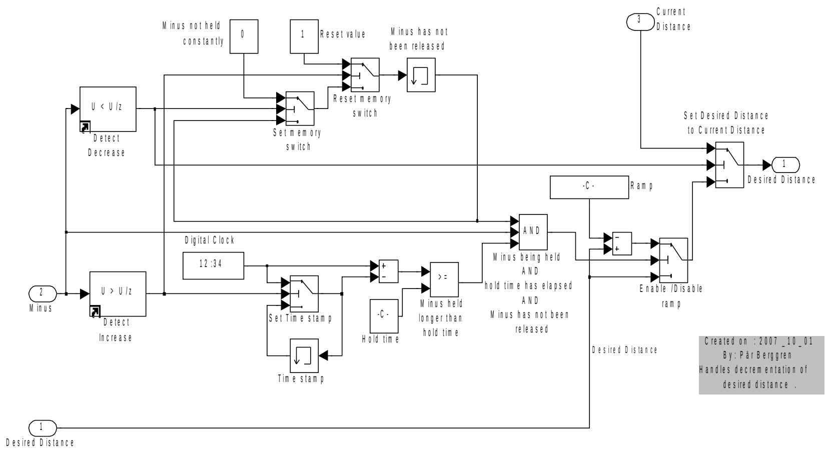


figure 6.9: The block Decrementation of desired distance. All of the decrementation and incrementation blocks have this layout.

The contents of the cruise controller block, see Control in figure 6.4, is shown in figure 6.10. The only difference from the previously existing block is that the integratory state has been given reseting capabilities.

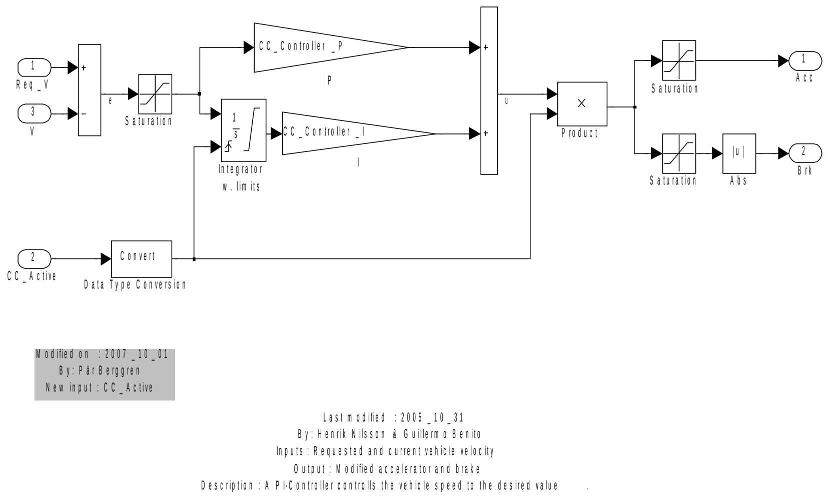

1
Req_V
3
V
e
Saturation
CC_Controller _P
P
1/s
Integrator w. limits
CC_Controller _I
I
u
×
Product
Saturation
1
Acc
Saturation
|u|
Abs
2
Brk
2
CC_Active
Convert
Data Type Conversion
Modified on : 2007_10_01
By: Pär Berggren
New input : CC_Active
Last modified : 2005_10_31
By: Henrik Nilsson & Guillermo Benito
Inputs: Requested and current vehicle velocity
Output: Modified accelerator and brake
Description : A PI-Controller controlls the vehicle speed to the desired value .

figure 6.10: Cruise Controller

Chapter 7

Discussion, future work and conclusions

THIS CHAPTER STARTS with a discussion of the results and other issues related to the paper followed by a presentation of a few ideas regarding future work. Finally some conclusions drawn from the results are brought up.

7.1 Discussion

THE CONTROL APPROACHES have no capabilities to handle low frequency measurement noise, which is a common problem with PID controllers. Fortunately the graphics application of the simulator produces little or no such low frequency noise, so this is not really much of a problem here. The noise that is produced by the graphics program comes from the process of discretization of the cars position as they are being propagated through the environment and this mostly contain high frequency components.

Since the CAN-bus communication program suggested uses what is known as a critical section and some implementations of those may affect the realtime clock, it is important to clarify that the type used here is a kind of synchronization primitive associated with the operating system running on the control computer. This synchronization primitive is local and only blocks the switching of threads within the process, thereby not affecting the realtime clock. It would be unfortunate If that occurred in the discussed application, since the messages sent needs to be updated within a certain time span or the heads-up display would light up with warning messages.

The model of the throttle to velocity process is designed on a step response that during most of the time desribes the highest gear. The ACC however almost never actually uses the highest gear, so if the simulation that created the step response had been formed in such a way that the top gear had not been allowed to engage, the resulting model may have described the actual process better. The only difference between gears however is that they change the gain of the closed loop system, by changing the equivalent mass of the car. Therefore the model errors introduced by higher gears should be countered by the high gain margins produced by the suggested controller.

The intuitive contol design used in selecting the parameters of the PID controller has the drawback of being suboptimal. However, the controller do take into account the stationary nonlinearities in the limitations on the control signal.

The distance plot in figure 5.14 provides information about the rise time, settling time and overthrows. The rise time seams furthermore to have an indicator in the maximum distance in the plot. The settling time is however the most relevant indicator of the controller´s quickness. The overthrows in the plot gives the first failed requirement of a controller as the PID controller with only distance feedback has several large overthrows.

The emergency stopping simulation gives information about minimum stopping distance under different maximum allowed braking levels, as can be seen in table 5.4. That the maximum braking level has such an impact on the stopping distance suggests that a separate system for emergency braking might be a good idea, as the controllers can be considered more comfortable if they cannot use full brake under normal conditions.

An almost identical simulation was carried out on the AICC in (Ioannou, 1993), in which five vehicles were considdered, but no noise was added to the measurements. The results obtained in that simulation was more based on the platoon dynamics that occur when several vehicles are considdered and state that all vehicles had stopped within ten seconds and that no collision occurred. However the stopping distance can be measured in graphs and seam to be between ten and twelve meters, which cuts the ones obtained in the simulations here in half. These differences can most likely be explained with that the simulations done here are slightly more realistic (since more effort has been put into the design of the simulated vehicle) and that the AICC used is of the exact same design (in a simulated vehicle very different from what it was designed for) and the added noise.

The cut-in simulations done give additional information about the collision avoidance capabilities of the controller. However, it may be argued the simulations are done with large reference values, which produce large errors and thereby simplifies the control situations. The distance by itself however do not offer a complete view of the control situation. It needs to be viewed as a time headway in order to give relevant information about the choise of reference value. The reference value is chosen as 30 m, which at a velocity of 20 m/s gives a time headway of 1.5 seconds. The AICC was submitted to cut-in simulations under various conditions in (Ioannou, 1993) in which the results was werther collision did or did not occur. The simulations done in this paper only results in werther or not the two compared control approaches are comparable with eachother.

7.1.1 Future work

Improve the realism of the radar sensor by, for instance, creating a better criteria for the selection of a leading vehicle, so that vehicles appearing between the simulated car and the leading vehicle are detected.

Create driving situations (in the graphical environment) that tests the capabilities of the control algorithm in “real” situations. For instance, emergency stopping and cut-in situations or even more interesting, an evasive action test. This may however be a bit challanging since the development tool seams to be broken.

Implement a sensor based on image analysis. Several interesting algorithms exists, for instance using the characteristic vertical edges of cars easily identified with sobel operators or using the chromatics of the cars.

Since the system have varying gain, due to the gearbox and brakes, it may be motivated to test adaptive control strategies or gain scheduling. It may also be motivated to test any and all control algorithms more thoroughly regarding different safety situation.

Add high frequency moiton feedback to the simulator. For instance by installing electrical vibration actuators in the driving seat.

7.2 Conclusions

In the work done the best overall control performance was found when using the control approach with feedback of distance and velocity difference (see figure 5.2 on page 34 for the structure of the controller). The suggested controller is shown to eliminate persistent errors. The closed loop system (with the linear model) is stable when using the suggested controller, with good stability margins and the comfort provided is high due to the low amplitude of overthrows. It is insensitive to high frequency measurement noise and the control signal activity is limited.

The suggested changes to the CAN-bus communication program will provide the simulator with the bidirectional realtime communication necessary to provide driver interaction.

The controller is provided with feedback signals from the software based radar plug-in program suggested. However, these signals are noisy in themselves due to discretization effects in the graphics program and the numerical derivation of this effect.

The suggested control approach has safety capabilities regarding emergency stopping and cut-in situations that are deemed acceptable since no collision occurs in the considdered situations.

Bibliography

Fabian, M. (2006) *Discrete Event Systems, Lecture Notes (ESS 200).* Göteborg: Chalmers University of Technology. (R 004/2004) ISSN 1403-266X

Ioannou, P. A. & Chien, C. C. (1993) Autonomous Intelligent Cruise Control. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.* p. 657 – 672. Vol. 42, No. 4, November 1993

Lennartson, B. (2002) *Reglerteknikens grunder.* (Fourth edition) Lund:Studentlitteratur. ISBN 91-44-02416-9

Ljung, L. and Glad, T. (2004) Modellbygge och Simulering. (Second edition) Lund: Studentlitteratur. ISBN 91-44-02443-6

Nordling, C. and Österman, J. (2004) *Physics Handbook for Science and Engineering.* (Seventh edition) Lund: Studentlitteratur. ISBN 91-44-03152-1

OSG Community (2007) OpenSceneGraph. http://www.openscenegraph.org (2007-11-20)

Robert Bosch GmbH. (1991) *CAN Specification.* (Version 2.0) Stuttgart: Bosch. www.bosch.com (2007-10-01)

Thomas, B. (2001) *Modern Reglerteknik.* (Third edition) Stockholm: Liber. ISBN 91-47-05085-3

Vector Informatik GmbH. CAN Driver Library for CANcardXL / CANcardX / CAN-AC2-PCI: User Interface Description. (Version 3.4) Stuttgart: Vector Informatik GmbH.

Table of figures

Appendix A: Source code for proposed CAN application changes

The major changes to the CAN communication program suggested in this paper, can be found here. Most of the source code has few and outspread changes and was determined to be far to volumous to present here, therefore everything but the threads dealing with the CAN-bus communication is omitted.

```
void CANCom_thread(void *dummy){

      //Local variables:
      int i               = 0;        //Index varable.
      Vstatus vErr;                   //Variable for status management in CAN
                                      communication.
      Vevent *pEvent;                 //Event containing the current message.
      DWORD CCount        = 0;        //Variable for time-keeping.

      threadRuning              = 1;
      while (threadRuning) {

            WaitForSingleObject(gEventHandle,10);

            for (i = 1; i <= numEvents; i++) { //Update CAN-list loop.
                  vErr = ncdReceive1(gPortHandle,&pEvent);
                  //Deal with possible errors.
                  if (vErr&&vErr!=VERR_QUEUE_IS_EMPTY) goto ncdError;
                  if (vErr==VERR_QUEUE_IS_EMPTY)      break;
                  if (LogEvents)
                        printf("%s\n",ncdGetEventString(pEvent));

                  CCount = GetTickCount();            //Get time.
                  EnterCriticalSection(&cs);          //Use critical section
                  FaS->canMsgframeList->CheckCan(*pEvent);

                  LeaveCriticalSection(&cs);
            }     //End for-loop.

            outData[0] = (char) FaS->CANUpdateReadSignals(KeyAutoStart);

            FaS->CANUpdateCC(CCArray);
            outData[1] = CCArray[0];            //CC_On
            outData[2] = CCArray[1];            //CC_Inc.
            outData[3] = CCArray[2];            //CC_dec.

      }// End while

      DeleteCriticalSection(&cs);
      _endthread();

ncdError:
      DeleteCriticalSection(&cs);
      printf("ERROR: %s\n",ncdGetErrorString(vErr));
      _endthread();
}                                                          // End thread.
```

```
//////////////////////////////////////////////////////////////////////
// CANWrite_thread()
//--------------------------------------------------------------------
// Thread for writing to the CAN-bus.
void CANWrite_thread(void *dummy){

      //Local variables:
      double *pFloat    = (double *) &inData[4];      //Pointer for use with
                            communicating the global variable VehicleSpeed.
      int *pInt         = (int *) &inData[0];         //-"- EngineRPM.
      DWORD CCount      = 0;                //Varable used for time-keeping.

      threadRuning = 1;
      while (threadRuning) {
            EngineRPM = (float) *pInt;
            VehicleSpeed = (float) *pFloat;

            CCount = GetTickCount();       //Get time.
            EnterCriticalSection(&cs);     //Enter critical section.
            FaS->CANUpdateWriteSignals(VehicleSpeed, EngineRPM);
            FaS->canMsgframeList->UpdateWrites(CCount, gPortHandle,
                  gChannelMaskLS, gChannelMaskHS);
            LeaveCriticalSection(&cs);
            Sleep(1);   //Release processor from thread. Unit is in
                              millisec.
      }
      DeleteCriticalSection(&cs);
      _endthread();
}                                                     // End thread.
```

Appendix B: Source code for the suggested “radar” sensor

Here source code for the so called radar sensor, developed in the context of this paper, is presented. The plug-in stucture was inspired by the existing plugins, but represents a very different use of plug-ins than the previously existing ones.

```
#include <stdio.h>
#include <communication/TransmitterPool.h>
#include <Environment/objectnodes/car.h>
#include <Environment/objects.h>
#include <Environment/SimCar.h>
#include <osg/group>
#include <Environment/world.h>
#include <math.h>

extern "C"
{

__declspec(dllexport)
bool update(float carPos[3], Communication::TransmitterPool&
transmitterPool, float output[4])
{
   memset(output, 0, sizeof(float)*4);
   //Constants
   const double l = 305;        //305ft is how far this "radar" can see.
   const int P = 200;
   const int Q = 8000;
   const double maxAng = 90;
   const double PI = 3.141592685358979;
   const osg::Group &fwdVehicles = Environment::World::get()->getObjects()-
>getFwdDirVehicles();  //First part of all cars.
   const osg::Group &bwdVehicles = Environment::World::get()->getObjects()-
>getBwdDirVehicles();  //The other part of all cars.

   //Global variables
   double        theta;        //Angle to leader.
   double        optTemp;      //Optimization variable.
   double        opt = 1e20;   //Maximum optimization value
   double        tempL;        //Temporary length.
   unsigned int  i = 0;        //For-loop iteration variable.
   double        direction = 0; //Outgoing direction.
   double        lOut = l;     //Outgoing distance to leader.
   float         phyCarVel = 0; //Outgoing Velocity of leader.
   osg::Node *   node;         //Temporary variable for extracting vehicles
                               from database.
   osg::Vec3     dirVec;       //Vector from your car to leader.
   osg::Vec3     phyCarPos;    //Position of vehicles in 3D.
   osg::Vec3     phyCarDir;    //Rotation of the car's local to the global
coordinate system.
   osg::Vec3     carPosVec = ((Environment::ObjectNodes::PhyCar *)
Environment::SimCar::get())->getPosition();    //Your position vector.
   osg::Vec3     eX;           //Unit vector along x-axis.
   osg::Vec3     eZ;           //Unit vector along z-axis.
   osg::Vec3     planeNormal;  //Normal to plane.
   osg::Quat     carRot = ((Environment::ObjectNodes::PhyCar *)
Environment::SimCar::get())->getRotation(); //Rotation from simulated car's
                               local to global coordinate system.
   osg::Vec3     carDir;       //Direction of simulated car in global
                               coordinate system..
   osg::Group *     vehicles = new osg::Group;    //To handle all cars.
   FILE *        tDist = fopen("DistCase.txt","a");

   Environment::ObjectNodes::PhyCar *phyCar;//Pointer to vehicle.

   //Initialize variables
   eX.set(1,0,0);
   eZ.set(0,0,1);
   carDir.set(1,0,0);
   carDir = carRot*(carDir);
   //Unite vehicles in a single database.
   for(i = 0; i < bwdVehicles.getNumChildren(); i++) {
   //Put first part of all cars in Group vehicles.
      node = const_cast<osg::Node *>(bwdVehicles.getChild(i));
      if (!vehicles->insertChild(i, node)) {
         fprintf(tDist,"Error 1: error while inserting bwdVehicles.");
      }
   }
   for(unsigned int ii = 0; ii < fwdVehicles.getNumChildren(); ii++) {
      //Put the other part of all cars in Group vehicles.
      node = (osg::Node *)(fwdVehicles.getChild(ii));
      if (!vehicles->insertChild(i, node)) {
         fprintf(tDist,"Error 2: error while inserting fwdVehicles.");
      }
      i=i+1;
   }

   //Search Group of all vehicles for appropriate leader car.
   for (i = 0; i < vehicles->getNumChildren(); i++)  {
      node = (osg::Node *) (vehicles->getChild(i));
      phyCar = (Environment::ObjectNodes::PhyCar *) (node);
      //Car to be examined.
      phyCarDir.set(1,0,0);

      if (phyCar != Environment::SimCar::get()){
         phyCarDir = (phyCar->getRotation()) *(phyCarDir);
         if (carDir.operator *(phyCarDir) <= 0) { //Check if going in
                                                   opposite direction.
            continue;
         }else {
            phyCarPos = phyCar->getPosition();
            dirVec = phyCarPos.operator -(carPosVec);//Calculate a
                              directional vector from simulated car to
                              leading vehicle.
            planeNormal = carDir.operator ^(eZ);  //Calculate normal to
                                                    plane.
            theta = PI/2 - acos(dirVec.operator *(planeNormal) /
(dirVec.length() * planeNormal.length()));  //Calculate angle to plane.
            tempL = dirVec.length();  //Distance to simulated vehicle.
            optTemp = P * pow(tempL-160,2) + Q * pow((abs(theta)/PI)*180,2);
                             //Quadratic criteria with the penalties P and Q.
            if (optTemp <= opt && (dirVec.operator *(carDir)) > 0) {
                              //Minimization of the criteria and check that
                              leader is in front of you.
               if (tempL <= l && tempL > 10) {     //Check for acceptable
                                                    ranges.
                  lOut = tempL;
                  direction = (abs(theta)/PI)*180;
                  phyCarVel = phyCar->getSpeedMeterPerSec();
                  opt = optTemp;
               }
            }
         }
      }
   }
   output[0] = lOut;
   output[1] = direction;
   output[2] = phyCarVel;

   if (lOut >= l) {//No cars fulfill requirements.
      output[3] = 0;//flag, 0 = no leader found.
   }else {
      output[3] = 1;//flag, 1 = leader found.
   }

   fclose(tDist);
   vehicles->unref();
   return true;
}
__declspec(dllexport)
const char* name()
{
   return "Radar Sensor";
}
__declspec(dllexport)
const char* description()
{
   return "Distance, direction and velocity of the closest vehicle";
}

}
```

Appendix C: Simulated Vehicle

The Chalmers Vehicle Simulator runs a mathematical model of a Volvo XC90. The simulator has a setting that changes the engine model from a physicaly modeled simulation (basicly a Simulink model of an engine) to a 2 dimensional mapping, which outputs torque based on the engine speed (revolutions per minute) and throttle level. The mapping is presented in figure C.1.

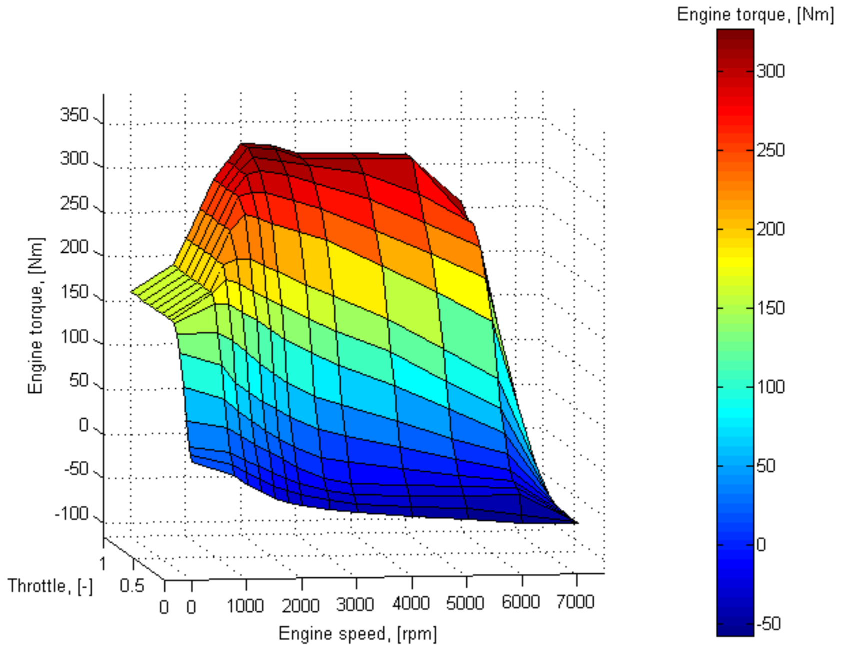
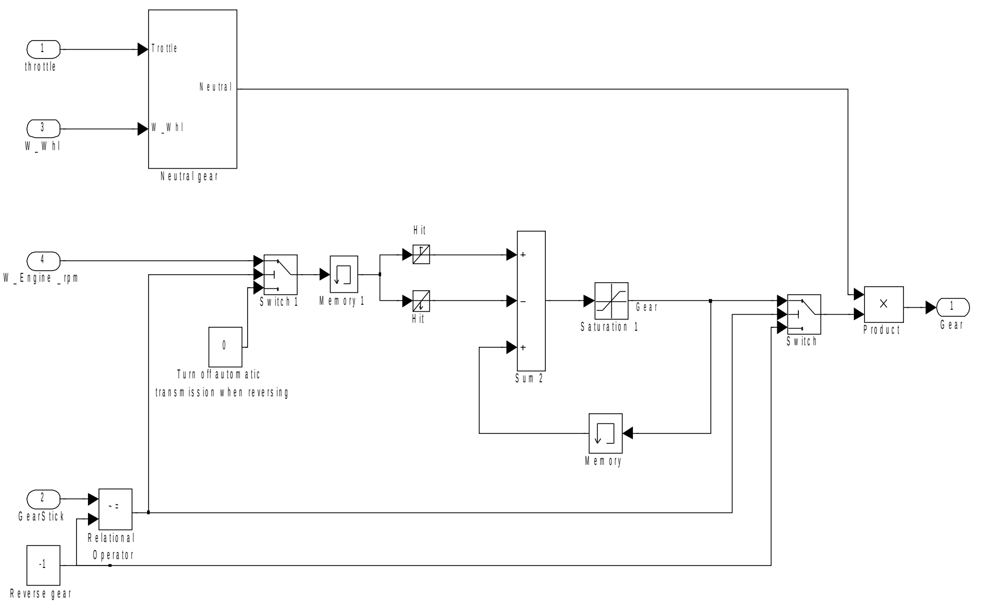


figure C.1: Mapping of an engine´s torque output. Torque based on the engine speed and throttle level. The mapping is a possible choise in the CVS that requires less computation time than the modeled engine.

Another important component that affects preformance is the gearbox. The gearbox implemented in the simulator is of automatic type and the shifting mechanism is modelled in Simulink and as appears in figure C.2 the behavior of the mechanism introduces a complicated nonlinearity.



Last modified : 2006_03_09
By: Henrik Nilsson & Guillermo Benito
Inputs: Engine speed [rpm].
Outputs: gear [1-6].
Descripction: When the engine speed is above /below ShiftUpRpm /ShiftDownRpm the gear increases /decreases. Memory 1 is preventing an algebraic loops .
Tunable Parameters : ShiftUpRpm , ShiftDownRpm

figure C.2: The Simulink model of the automatic gear shifting mechanism. At a specific rpm rate (5000) the mechanism shifts up and at another (2300) it shifts down, unless the gearbox is in neutral.