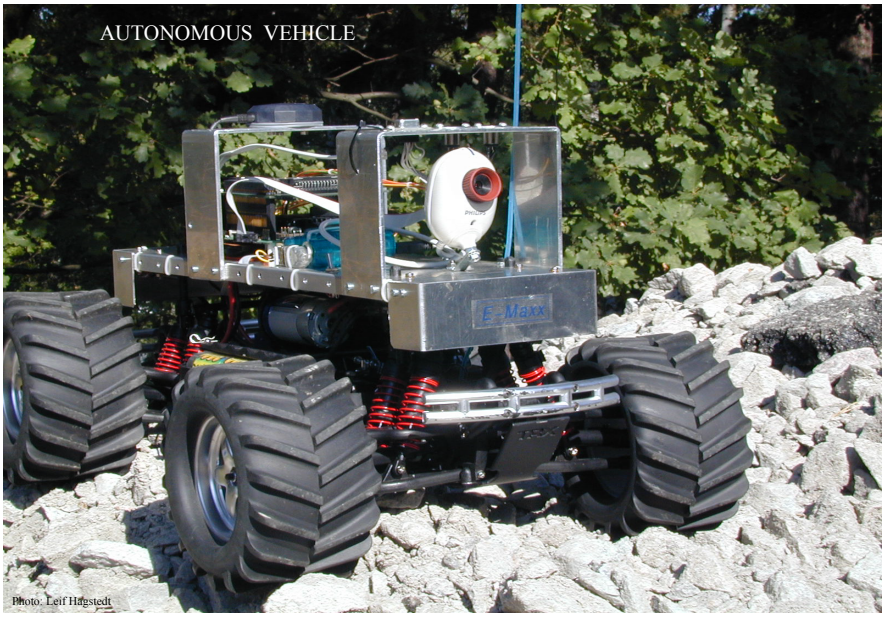
FOI-R--0770--SE
September 2002
1650-1942

Technical report

Andreas Alm

Autopilot for Autonomous Ground Vehicle, Modelling, Design and Implementation



System Technology Division
SE-172 90 STOCKHOLM
Sweden

Swedish Defence Research Agency
System Technology Division
SE-172 90 STOCKHOLM
Sweden

FOI-R--0770--SE
September 2002
1650-1942

Technical report

Andreas Alm

Autopilot for Autonomous Ground Vehicle, Modelling, Design and Implementation

Issuing organization	Report number, ISRN	Report type
Swedish Defence Research Agency System Technology Division SE-172 90 STOCKHOLM Sweden	FOI-R--0770--SE	Technical report
	Research area code: Vehicles	
	Month year: September 2002	Project no.: E6003
	Customers code: Contracted Research	
	Sub area code: Unmanned Vehicles	
Author/s (editor/s): Andreas Alm	Project manager: Peter Alvå	
	Approved by: Monica Dahlén	
	Scientifically and technically responsible: Anders Lennartsson	

Report title: Autopilot for Autonomous Ground Vehicle, Modelling, Design and Implementation

Abstract

This report describes the design and implementation of a controller that guides a ground vehicle from its current position to a predefined target. The controller is part of an Application Program Interface, API, together with lower level functions for controlling individual actuators and obtaining data from sensors, and some other, middle level controllers for various purposes. These functions are all accessible from applications based on various algorithms with the purpose of exhibiting autonomous behaviour. This particular controller observes position, velocity and course based on information obtained from a satellite navigation sensor, a GPS receiver. It uses other functions to control speed and steering on the platform, currently based on an electrically driven Radio-controlled car, about 0.5 m long.
The controllers are executed on a one chip computer, PC-104 standard, Pentium-class PC running the Debian GNU/Linux operating system.

Keywords: Autonomous Ground Vehicles, AGV, GPS, RC-car, Co-operative Vehicles

Further bibliographic information	Language
	English
ISSN: 1650-1942	Pages: 35
Distribution: By sendlist	Price: Acc. to pricelist Security classification: Unclassified

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets forskningsinstitut Avdelningen för Systemteknik SE-172 90 STOCKHOLM Sweden	FOI-R--0770--SE	Teknisk rapport
	Forskningsområde: Farkoster	
	Månad, år: September 2002	Projektnummer: E6003
	Verksamhetsgren: Uppdragsfinansierad verksamhet	
	Delområde: Obemannade farkoster	
Författare/redaktör: Andreas Alm	Projektledare: Peter Alvå	
	Godkänd av: Monica Dahlén	
	Tekniskt och/eller vetenskapligt ansvarig: Anders Lennartsson	

Rapportens titel

Autopilot för Autonoma Markfordon, Modellering, Design och Implementering

Sammanfattning

Denna rapport beskriver en modell och dess implementering av en regulator som styr ett markfordon från sin utgångspunkt till ett förutbestämt slutmål. Regulatorn är en del av ett Application Program Interface, API, med lågnivå-funktioner för individuell kontroll av fordonsrörelser och mottagning av data från sensorer, samt några andra, medelnivå-regulatorer för olika ändamål. Dessa funktioner är alla tillgängliga från applikationer baserade på olika algoritmer med målet att visa upp autonoma beteenden.
Regulatorn i detta fall observerar position, hastighet och riktning baserat på information från en satellitnavigeringssensor, en GPS mottagare. Den använder sig av andra funktioner för att kontrollera hastighet och styrning för plattformen, för tillfället en radiostyrd bil, ungefär 0.5 m lång.
Applikationerna utförs på en liten dator, PC-104 standard, Pentium-klass PC som använder sig av operativsystemet Debian GNU/Linux.

Nyckelord

Obemannade Markfarkoster, GPS, Radiostyrd bil, Samverkande Fordon

Övriga bibliografiska uppgifter	Språk
	Engelska
ISSN: 1650-1942	Antal sidor: 35
Distribution: Enligt missiv	Pris: Enligt prislista Sekretess: Öppen

Contents

Chapter 1

Introduction

The work presented here is a master's thesis in engineering, done at the Swedish Defence Research Agency (FOI), Division of Systems Technology. Supervisor at FOI was Anders Lennartsson and at the Royal Institute of Technology (KTH), Xiaoming Hu.

1.1 Project Background

The rationale behind the project under which this work has been conducted is to examine how autonomous vehicles can co-operate with each other. The vehicles do not necessarily need to be of the same kind, there can be some ground vehicles which are going to co-operate with aerial vehicles or maybe some underwater or surface vehicles.

The vehicles are equipped to communicate with each other and a basestation. Commercial Of The Shelf, COTS, products, are going to be used as much as possible.

For different missions the vehicles might need different equipment, such as digital camera(s) or other sensors. To obtain this, each vehicle's equipment is planned to be easily exchanged. The vehicles in a group may not need the same equipment, they will be able to co-operate and exchange information with each other to complete the mission. All the vehicles will most likely have a GPS-receiver, since this is the positioning system used in this project.

An immediate problem with autonomous vehicles is the ability to navigate to a predetermined target. Groups of autonomous vehicles have the additional difficulty of being able to travel in a formation well suited to the circumstances, e.g. terrain. Avoiding collision between members of the group is also important.

Some events that must be considered are:

- What happens if the base loses contact with a vehicle?
- What happens if a vehicle is taken out, for example by an enemy? And in that case, can the other vehicles still communicate with each other?

Of course there are other things that need to be addressed as well.

1.2 This Thesis

In this thesis the problem of controlling a wheel-based mobile platform is investigated, which is a well studied topic, see for example [4, 11, 1, 3]. The main task is to develop an autopilot for a single vehicle, for the particular navigation task of moving the shortest distance to the predetermined target, given the conditions imposed by its kinematical and dynamical properties. The vehicle used in this case is a radio-controlled car, which can be read about in section 2.2.

A model was first developed in Simulink, MATLAB, to see how the vehicle behaved on certain signals, and if it found its way to the predefined target. The algorithms of this model were then coded in C++, to be executed on a real vehicle. The program can also be used in computer simulations to see if and how it works. The intention was to use the program on the vehicle itself, this was not performed because of lack of time. The RC-car was supposed to be moving on an open relatively flat field with no obstacles.

1.3 Overview

The disposition of this report is as follows.

Chapter 2 Contains basic information about how the problem was formulated, how the system of co-ordinates for the world and the vehicle was defined. The vehicle model, both kinematic and dynamical, is described as well as the vehicle platform.

Chapter 3 Describes how simulation and analysis of the problem was done in Simulink, a simulation-package of MATLAB.

Chapter 4 Includes a description of the implementation. How it all worked out when tests were performed.

Chapter 5 Describes the conclusions that has been drawn from this work.

Chapter 6 Explains some problems that can and probably will be taken care of in the near future. It also contains a bit about the future development of partly the radio-controlled car, partly the system of co-operative robots.

Appendix A Focuses on how this implementation is coded in C++. How the program is built up, and the parts it has been split up in.

Appendix B Gives a short description about the satellite positioning system used in the project, the GNSS Global Navigation Satellite Systems, commonly called GPS, Global Positioning System.

Chapter 2

Modelling

The world is defined as a fixed system of co-ordinates, where the x-axis is headed north, and the y-axis is headed east. This is a very common way of explaining the co-ordinates of the world. The vehicles system of co-ordinates is fixed in the middle of its rear axle. The system is illustrated in figure 2.1.

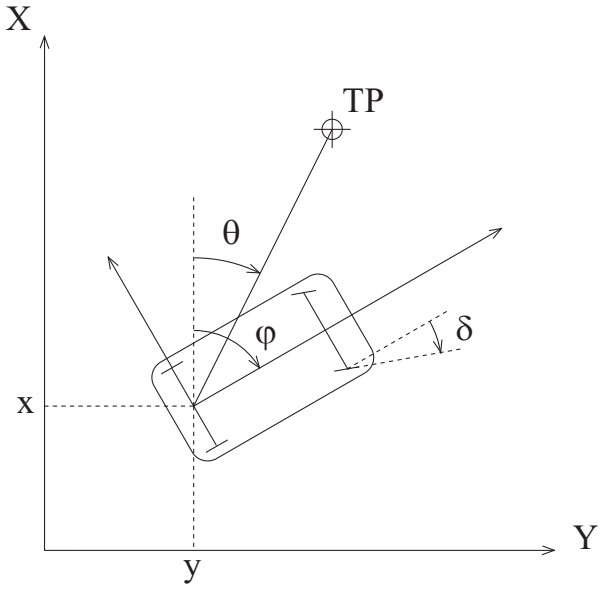


Figure 2.1: The system of co-ordinates

The figure also explains some angles defined as follows:

θ : The direction to the target, TP.

φ : The direction of the vehicle.

The difference between these angles, i.e. $\varphi - \theta$, is calculated to make decisions about the steering angle, δ. If the difference in directions are greater than the maximum steering angle, the car turn around with maximum steering angle toward the target with a small velocity. Then the steering angle decreases and when it is relatively small the car increase the speed. The speed is then decreased when the vehicle are within a certain range of the target, this is made to find the target and get as close as possible.

2.1 Vehicle Model

A kinematic model is used in this thesis, and it is described in more detail in section 2.1.1. A dynamical model can be more precise and useful. Such a model is described in section 2.1.2,

but it is not implemented in this thesis, it is left for future developers.

Figure 2.2 shows the vehicle approximated as a single track model, [6, 1]. The model is obtained by grouping the front and the rear wheels together as two single wheels, where δ is the steering angle, v the longitudinal velocity, L the distance between the front and the rear axle and R is the driving radius around *ICR, Instantaneous Center of Rotation*, [2, page 50].

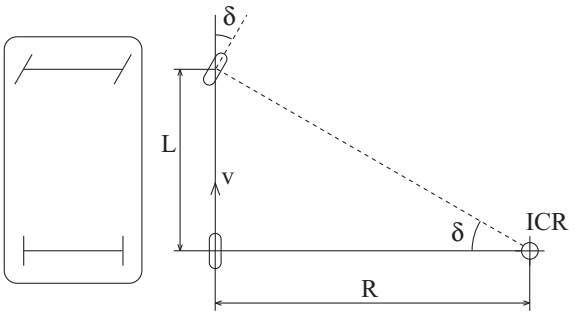


Figure 2.2: The Single Track Model

2.1.1 Kinematic Model Figure 2.1 shows that the position of the vehicle is defined by the co-ordinates (x, y) of the rear axle midpoint, and the angle φ specifying its orientation relative to the north-axis. We define ξ describing the robot placement, [2, page 48]

$$\xi \triangleq \begin{bmatrix} x \\ y \\ \varphi \end{bmatrix}. \tag{2.1}$$

Then the kinematic model that we are going to exploit is, [5, page 237]

$$\dot{\xi} = \begin{bmatrix} v\cos\varphi \\ v\sin\varphi \\ \omega \end{bmatrix}, \tag{2.2}$$

where ω is the angular velocity.

Now since we have a controlled steering angle, δ, we get the front-wheeled car model that looks like, [4, page 14]

$$\begin{aligned} \dot{x} &= v\cos\varphi \\ \dot{y} &= v\sin\varphi \\ \dot{\varphi} &= \frac{v}{L}\tan\delta, \end{aligned} \tag{2.3}$$

where L is the length of the vehicle.

Decisions about velocity and steering has to be made continuously when the vehicle is moving. To make these decisions some calculations has to be done.

First of all we need to know the direction to the target, denoted θ. And since the position of the vehicle, (x_v, y_v), and the target, (x_T, y_T), are known, we get θ as

$$\theta = \text{atan2}(\Delta y, \Delta x), \tag{2.4}$$

where $\Delta x = x_T - x_v$ and $\Delta y = y_T - y_v$. Now the difference between the orientation of the vehicle and the direction to the target gives us the necessary information about where to go. The direction in which the vehicle is moving is observed from the information collected with the GPS-receiver and is denoted φ. Therefore we get the difference Θ as

$$\Theta = \varphi - \theta. \tag{2.5}$$

If the difference is greater than π turn right, see equation 2.7, otherwise turn left, equation 2.6.

$$0 \leq \Theta \leq \pi \tag{2.6}$$
$$\pi < \Theta \leq 2\pi. \tag{2.7}$$

So, now we know which direction to go in, but we do not know the steering angle. The steering angle sent to the car is actually the difference in directions, Θ. But since the vehicle has a limitation of steering range, this has to be considered. This is however quite easily done. If the difference in directions exceeds the maximum steering angle, $\delta_{\max}$, then the maximum steering angle is sent to the car, the difference in angles is otherwise sent to the car as steering signal.

In this model we use two velocities that are decided due to the steering angle and the distance to the target D, equation 2.8. If the steering angle exceeds $\delta_{\min}$, we go slow, else we go a bit faster. The slower velocity is also used when we are relatively close to the target.

$$D = \sqrt{\Delta x^2 + \Delta y^2} \tag{2.8}$$

To avoid rolling we need to control the velocity and acceleration. Denote the critical acceleration as κ, and the perpendicular acceleration as a. Now κ needs to be greater than a, $(\kappa > a)$, and therefore, since

$$a = \frac{v^2}{R} \tag{2.9}$$
$$\tan\delta = \frac{L}{R}, \tag{2.10}$$

the condition for v is

$$v < \sqrt{\frac{L\kappa}{\tan\delta}}. \tag{2.11}$$

If we now put the maximum steering angle in to equation 2.11 ($\delta = \delta_{\max}$), we get the maximum speed to avoid rolling. Since we need a safety margin this velocity is multiplied by a factor less than one, and this new velocity is used as the lower velocity in this thesis.

If the steering angle are within certain limits, say $\pm\delta_{\text{narrow}}$, we would like to increase the speed. The speed used in this case is computed by putting δ_{narrow} into equation 2.11. And of course this velocity also need to be multiplied by the factor mentioned above.

2.1.2 Dynamical Model As pointed out above, the dynamical model is much more accurate than the kinematic. To analyze the control algorithm, we use the so called single track dynamical model, [5, 1]. There are some slight changes from figure 2.2, these are illustrated in figure 2.3. This model is based both on the balanced forces and the torque conditions.

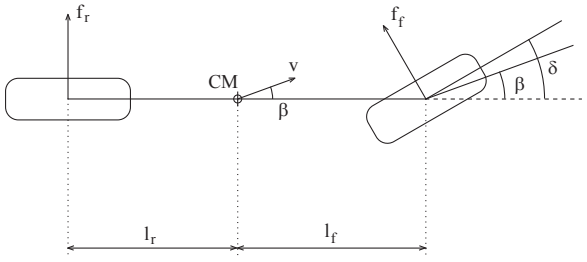


Figure 2.3: The Single Track Dynamical Model

As seen in figure 2.3, the side forces acting on the wheels are f_f and f_r. Furthermore, if we let f_x and f_y be the forces acting on the center of mass, CM, and m_z be the torque, the force-torque balance for three degrees of freedom in the horizontal plane gives us that, [6]

$$\begin{pmatrix} -mv(\dot{\beta}+r)\sin\beta + m\dot{v}\cos\beta \\ mv(\dot{\beta}+r)\cos\beta + m\dot{v}\sin\beta \\ J\dot{r} \end{pmatrix} = \begin{pmatrix} f_x \\ f_y \\ m_z \end{pmatrix}, \tag{2.12}$$

where r is the yaw rate, v the longitudinal velocity and β the side slip angle between the vehicle center line and the velocity vector $\vec{v}$ at the center of gravity. The mass is m and J the moment of inertia around the vertical axis.

The tire characteristics can be approximated by, [4, page 100]

$$\begin{aligned} f_f &= c_f^*\mu\left(\delta_f - \beta - \frac{l_f r}{v}\right) \\ f_r &= c_r^*\mu\left(-\beta + \frac{l_r r}{v}\right), \end{aligned} \tag{2.13}$$

where c_f^* and c_r^* are the front and rear cornering stiffness of the car, and μ is the adhesion constant, depending on the surface of the road. Typical values of μ are, [1]

$\mu = 1$	dry road
$\mu = 0.5$	wet road
$\mu = 0.15$	ice.

In the equations above we use l_f and l_r, these are the distances between the center of mass and the front and rear wheels respectively, shown in figure 2.3. Let $c_f^*\mu = c_f$ and $c_r^*\mu = c_r$ for easier calculations.

A projection of the wheel forces onto the center of gravity gives the forces in x- and y-direction as

$$\begin{aligned} f_x &= -f_f \sin\delta_f \\ f_y &= f_f\cos\delta_f + f_r, \end{aligned} \tag{2.14}$$

and the torque can be computed as

$$m_z = f_f l_f \cos\delta_f - f_r l_r. \tag{2.15}$$

The motion of the center of mass (x, y) of the vehicle is defined as

$$\begin{aligned} \dot{x} &= v\cos(\varphi+\beta) \\ \dot{y} &= v\sin(\varphi+\beta). \end{aligned} \tag{2.16}$$

Under the assumptions that the velocity is constant and that the side slip angle is small, we get a model that can be written as, [10]

$$\dot{x} = v\cos(\varphi+\beta) \tag{2.17}$$

$$\dot{y} = v\sin(\varphi+\beta) \tag{2.18}$$

$$\dot{\beta} + r = \frac{f_f + f_r}{mv} = a_{11}\beta + a_{12}r + b_{11}\delta_f \tag{2.19}$$

$$\dot{\varphi} = r \tag{2.20}$$

$$\dot{r} = \frac{f_f l_f - f_r l_r}{J} = a_{21}\beta + a_{22}r + b_{21}\delta_f, \tag{2.21}$$

where

$$\begin{aligned} a_{11} &= \frac{-(c_r + c_f)}{mv} & a_{12} &= \frac{(c_r l_r - c_f l_f)}{mv^2} \\ a_{21} &= \frac{(c_r l_r - c_f l_f)}{J} & a_{22} &= \frac{-(c_r l_r^2 + c_f l_f^2)}{Jv} \\ b_{11} &= \frac{c_f}{mv} & b_{21} &= \frac{c_f l_f}{J}. \end{aligned} \tag{2.22}$$

This is a simplified model of the vehicle, since we have the assumptions about the constant velocity and the small side slip angle. But this model is satisfying enough to the radio-controlled car. To operate at moderately high speeds, the dynamical model is better used than the kinematic. This model can be improved to include various velocities. This improved model is then better to use if the terrain is an open field with obstacles. The model is more accurate in the maneuvers.

2.1.3 Stability Analysis An appropriate dynamical model, 2.1.2, has been set up and we can now proceed by showing that the algorithm is stable. If we write the control algorithm as $\delta_f = -k(\varphi - \theta)$, where k should be chosen to reflect the constraint of the maximum steering angle (since $\varphi - \theta \in [-\pi, \pi]$), we can calculate the errors in orientation Θ and yaw rate e by letting, [4, Paper C]

$$\Theta = \varphi - \theta \tag{2.23}$$
$$e = -\frac{a_{22}}{b_{21}} r - \delta_f, \tag{2.24}$$

we can now use equation 2.20 to obtain

$$\dot{\Theta} = \dot{\varphi} - \dot{\theta} = k\frac{b_{21}}{a_{22}}\Theta - \frac{b_{21}}{a_{22}}e - \dot{\theta},$$

since $r = (k\Delta\varphi - e)b_{21}/a_{22}$. Equation 2.21 can be used to compute

$$\dot{e} = -\frac{a_{21}a_{22}}{b_{21}}\beta + \left(a_{22} - k\frac{b_{21}}{a_{22}}\right)e + k^2\frac{b_{21}}{a_{22}}\Theta - k\dot{\theta}.$$

Finally we calculate $\dot{\beta}$ using equation 2.19,

$$\dot{\beta} = a_{11}\beta - (a_{12} - 1)\frac{b_{21}}{a_{22}}e + \left((a_{12} - 1)\frac{b_{21}}{a_{22}} - b_{11}\right)k\Theta.$$

From this we get the error model

$$\dot{\chi} = A\chi + b\dot{\theta}, \tag{2.25}$$

where $\chi^T = (\beta, \Theta, e)$, and

$$A = \begin{pmatrix} a_{11} & k(a_{12} - 1)b_{21}/a_{22} - kb_{11} & -(a_{12} - 1)b_{21}a_{22} \\ 0 & kb_{21}/a_{22} & -b_{21}a_{22} \\ -a_{21}a_{22}/b_{21} & k^2b_{21}/a_{22} & a_{22} - kb_{21}a_{22} \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ -1 \\ -k \end{pmatrix}.$$

To show that matrix A has eigenvalues with negative real part, MATLAB was used. Different values of v and k was tried and as long as v and k is greater than zero, the eigenvalues has a negative real part.

2.2 Platform

The platform that was supposed to be used in the project was as mentioned before a radio-controlled car. It is a relatively cheap platform to try out som basic algorithms with, and it is quite easy to use and it still fulfills its tasks.

The vehicle basically consists of:

- Four wheels, two steerable at the front axle and two fixed wheels at the rear axle.
- One engine to drive the vehicle forward and backward, controlled by a servo.
- One servo to control the steering of the two frontwheels.
- A batterypack, consisting of two 7.2[V], 3000[mAh] NiMH accumulators.

Beside this the vehicle consists of the following equipment that are being used in this thesis:

- One small computer, Pentium-class PC-104 size, used for control and information processing.
- One GPS-receiver to collect the position and velocity of the vehicle.
- Wireless Local Area Network, WLAN 802.11b for short and medium range communication with the base and other vehicles.
- A Bluetooth device for short range communication.
- One digital camera.
- Two USB-ports for extra sensors and other equipment.
- Accumulators for the computer and sensor equipment.

Chapter 3

Simulation

To determine how the vehicle behaves when control signals are transmitted to it, a simulation in Simulink, MATLAB, has been developed. The model is built up on block diagrams where the steering angle and the velocity are controlled.

3.1 Basics About the Simulation

Figure 3.1 is an overview of the whole system developed in Simulink. The subsystems can be viewed and will be explained in the following sections.

Initiation of some values are needed in the simulation, these values are the position and direction of the vehicle and the position of the target. These are set to some reasonable values, see MATLAB-code 3.1.

MATLAB-code 3.1 Vehicle Information

```
% Declaration of constants
LowVelocity = 2;                  %[m/s]
HighVelocity = 5;                 %[m/s]
TargetPosition = [100 200];       %[m]
TargetPosition2 = [250 150];      %[m]
MaximumSteeringAngle = pi/6;      %[rad]
SmallMaximumSteeringAngle = pi/8; %[rad]
DistanceBetweenAxles = 0.4;       %[m]
RangeLimit = 5;                   %[m]
DomainDegree = 2;

% Define initial conditions
InitialVehiclePosition = [0 0];   %[m]
InitialVehicleDirection = 0;      %[rad]
InitialSteeringAngle = 0;         %[rad]
```

3.1.1 Vehicle –Target Separation The subsystem *Vehicle–Target Separation* is illustrated in figure 3.2. The block simply transforms from cartesian to polar co-ordinates, that gives the direction and the distance from the vehicle to the target.

The vehicles position and the targets position can be used to compute the direction θ to the target

$$\theta = \operatorname{atan2}(u[2], u[1]), \tag{3.1}$$

where $u[1] = \Delta x$ and $u[2] = \Delta y$, see also equation (2.4). All this is measured in the above mentioned fixed co-ordinate system, figure 2.1. As the vehicle moves, new directions are being computed. These computations are being made to decide about the steering angle. This block also computes

$$D = \operatorname{hypot}(u[1], u[2]), \tag{3.2}$$

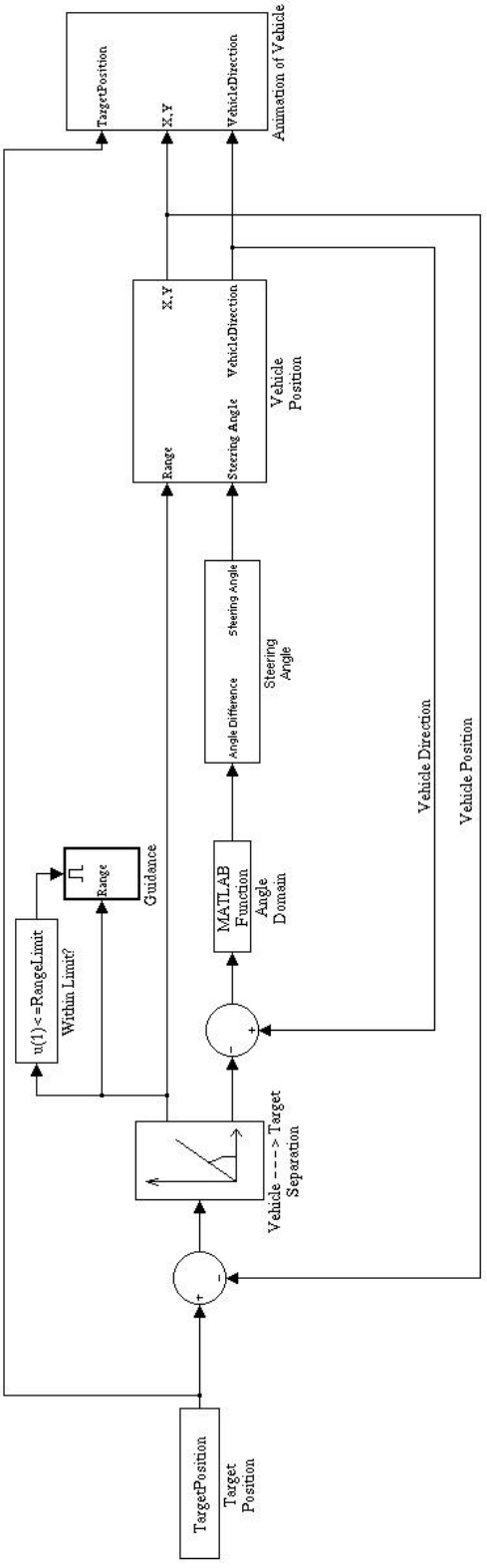


Figure 3.1: Vehicle Control

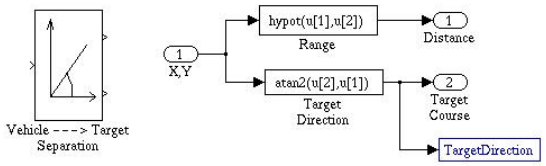


Figure 3.2: Vehicle–Target Separation

which simply is the Pythagoras' theorem to calculate the distance D to the target, this is also shown in equation (2.8). The distance is used to decide the velocity of the car and when to enable the subsystem that decides when to stop the simulation, this subsystem will be further explained in section 3.1.5.

3.1.2 Turning Left or Right? A decision is about to be made: Are we going to turn left or right to reach the target? This decision can be made by computing the difference in directions, equation (2.5). The difference are supposed to vary between 0 and 2π. To be sure about this, the difference in directions Θ are sent to a function in MATLAB, see MATLAB-code 3.2. The function adds or subtracts 2π until the domain is reached and outputs the steering angle δ.

MATLAB-code 3.2 Angle Domain

```
function delta=AngleDomain(Theta)
delta=Theta;
while( (delta<0) | (delta>2*pi) ),
  if delta>2*pi,
    delta=delta-2*pi;
  elseif delta<0,
    delta=delta+2*pi;
  end
end
```

This angle is then sent to the block called *Steering Angle* figure 3.3, that computes the steering angle to send.

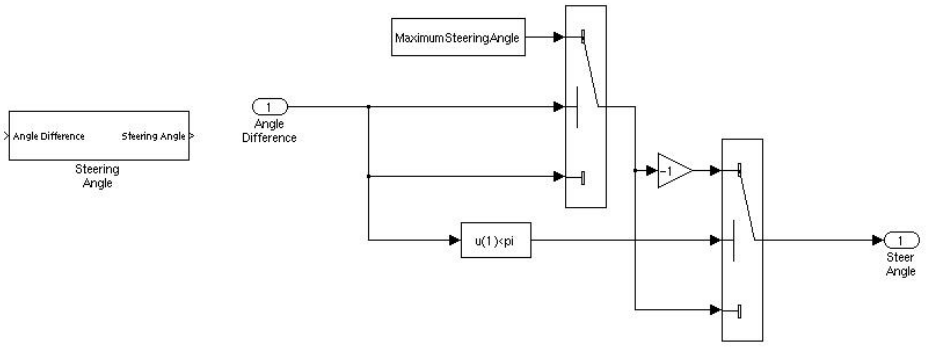


Figure 3.3: Steering Angle

Since the vehicle has a limitation of possible steering angles, there must be some kind of control that the steering angle sent to the vehicle is within these limits. This is done by a

switch as shown in figure 3.3. When the difference in directions exceeds the maximum steering angle, $\delta_{\max}$, the maximum steering angle are sent to the vehicle, otherwise the difference in directions are sent as steering angle.

Then the decision is about to be made about whether to turn left or right. If the difference in directions are less than π, then the signal is multiplied by -1, and therefore a left turn is made, otherwise we will go right, this is what the second switch in the figure does.

3.1.3 The Vehicles Position and Direction The subsystem *Vehicle Position*, figure 3.4 computes the new position of the vehicle and also the new direction.

The vehicles position is denoted (x, y), where x is the south-northposition and y is the west-eastposition, as showed in figure 2.1. The vehicles orientation in this fixed co-ordinate system is denoted φ.

By integration of equation 2.3 the vehicles position and direction is calculated. This is in Simulink easily done by the block *Integrator* that is a predefined block in Simulink. The position of the vehicle is printed with the block called *XY Graph*, then we can see the route of the vehicle during the simulation.

To compute the direction of the vehicle, the distance between the axles and the velocity are needed. The distance between the axles are measured and set as an initial value. There are two different velocities used in this thesis, one for large steering angles and one for smaller steering angles. The lower velocity is used when the target is being located and when we are close to the target. The high velocity is used otherwise. These velocities were in the simulation set to 2 and 5 respectively.

3.1.4 Animation of the Vehicle The system is animated in a final block, figure 3.5 *Animation of Vehicle*, that is predefined in Simulink. This animation concists of a missile, as the vehicle, and a target that has been put out.

This block requires three inputs, the position of the target, the position of the vehicle and the orientation of the vehicle.

Then for each calculation the model is updated and redrawn. By this we are able to see how the vehicle behaves in the system.

3.1.5 Guidance When are we going to stop the simulation? This is done by the block *Guidance*, see figure 3.6. First the block is enabled when we are within a certain predefined range of the target. Then if the distance to the target starts to grow, the simulation stops. When this is done the distance by which we missed the target is calculated.

3.2 Two Examples of Simulation

Two different kind of simulations has been developed. The first one describes how the vehicle is taking the closest way to a point A, and the second one the closest way to point B via point A. There are some minor changes between the simulations but these will be explained in the sections below.

3.2.1 Closest Way To Point A To take the vehicle the closest way to point A, the target, is the main task of this thesis. The simulation is based on the system described above. To find the closest way we first need to know the initial orientation of the vehicle. This and the known current vehicleposition and the targets placement is used as initial conditions in the system above.

In figure 3.7(a) we see the initial conditions in the simulation, the vehicles position and direction and the targets position. Further, figure 3.7(b) shows how the vehicle steers toward the target with maximum steering angle, and in figure 3.7(c) we go straight toward the target. In figure 3.7(d) we have reached the target.

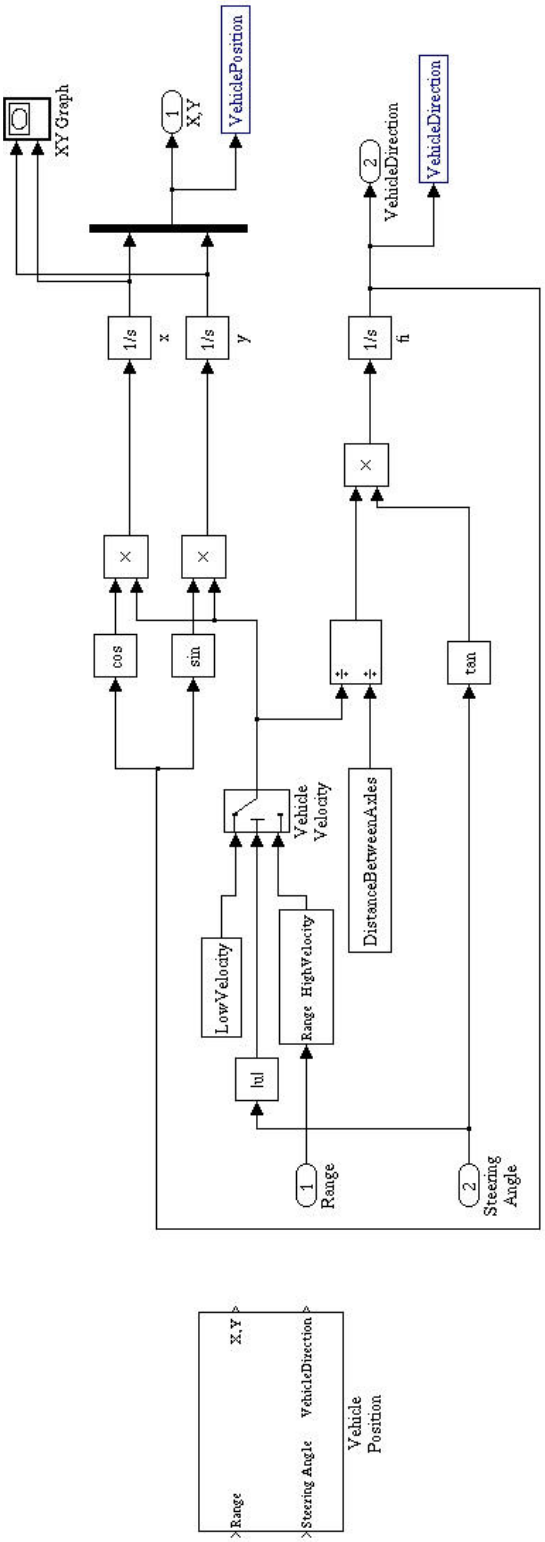


Figure 3.4: Vehicle Position

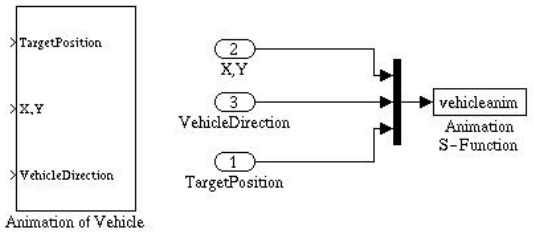


Figure 3.5: Animation of Vehicle

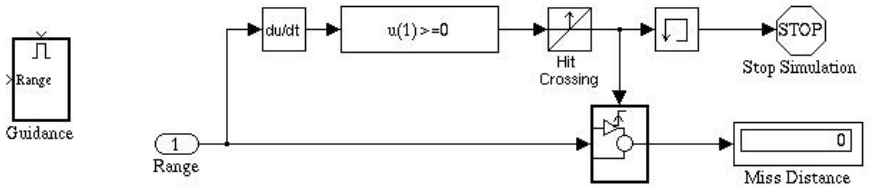


Figure 3.6: Guidance

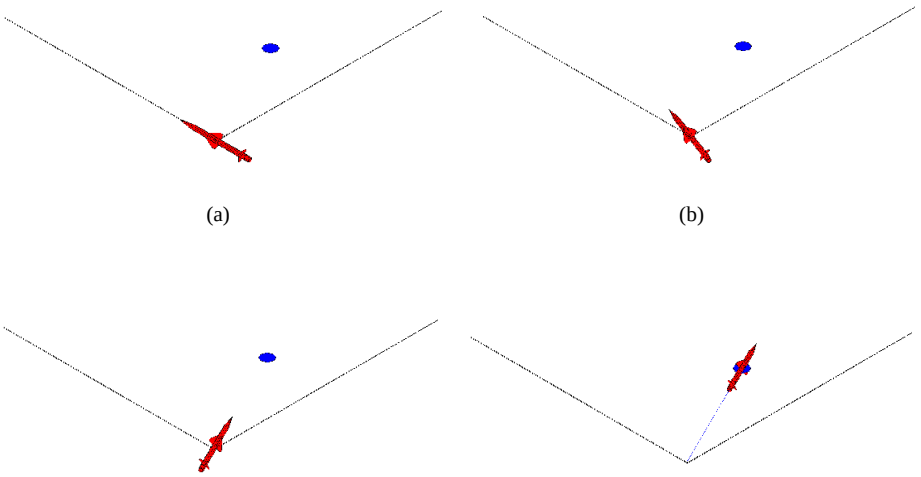


Figure 3.7: Simulation sequences

3.2.2 Closest Way To Point B via Point A In this second simulation, the vehicle is supposed to go from its initial position to point B. But before it goes to point B, it has to go via point A. This is quite easily fixed since the basics is already done as above (3.2.1). The difference is that when the first point is reached, a new targetposition has to be defined. The whole system can be viewed in figure 3.8.

There are some slight changes that has to be done with the block *Guidance* from section 3.1.5 above. All changes are about the targetposition and when to stop the simulation. We go toward the first target as in the previous section (3.2.1). When we are within limit of

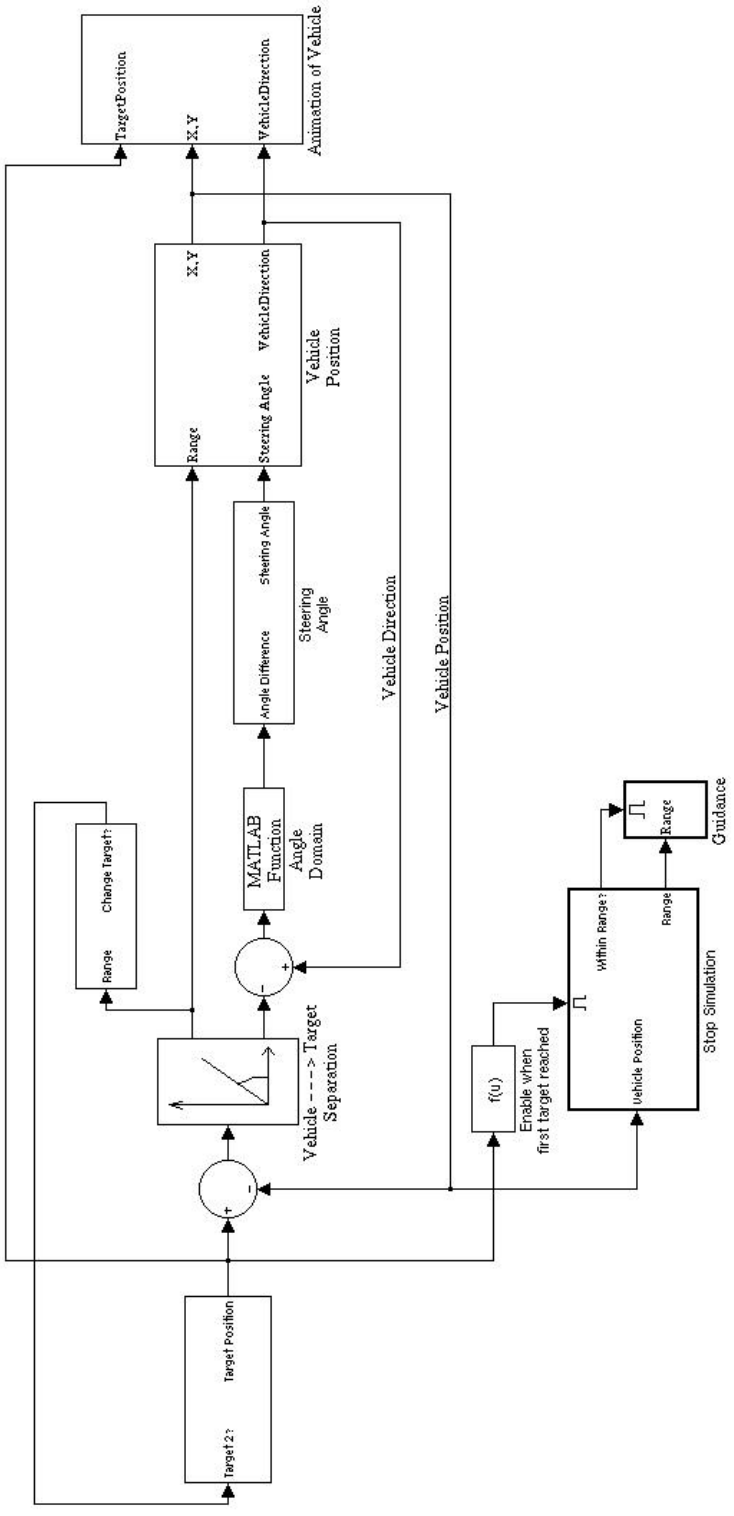


Figure 3.8: Vehicle Control for B via A-mission

the first target we send another target position to the system. So when we reach the first target we look for the second one and turn towards that one. The simulation then stops for the same reason as in 3.2.1 when we reach the second target

The simulation can be viewed in figure 3.9. As in the previous case the first figure shows the vehicles position and orientation, it also shows the first target point to be reached. Figure 3.9(b) pictures when the vehicle is moving toward the target. Further, in figure 3.9(c) we can see how the first target is reached. Then a second target position is sent to the system. This is in Simulink solved by putting a switch into the system.

After reaching the first target a subsystem is enabled. The reason for this system is to know when to stop the simulation. The simulation stops for the same reason as in the first simulation, that is when the distance to the target starts to grow.

The vehicle continues its journey toward the second target. Finally, in figure 3.9(d) we can see how the second and final target is reached.

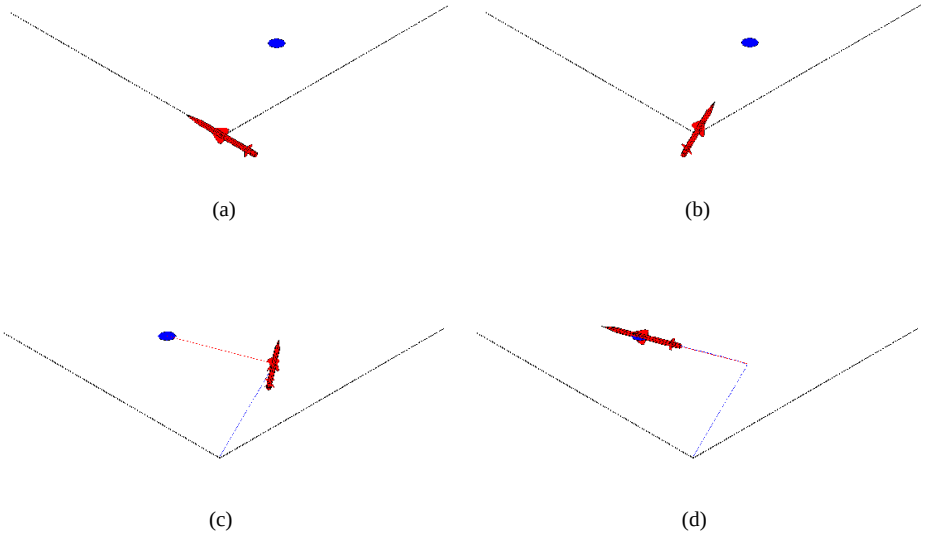


Figure 3.9: Simulation sequences

Chapter 4

Implementation

The program that has been developed in C++ has been implemented both for use in simulations and real runs on the vehicle. Since the model does not contain any dynamics of the vehicle, the tests on the RC-car are going to be performed in an open flat field and in the simulation they were performed in open terrain.

In appendix A there is a short description about how the program was developed in C++.

4.1 Implementation on the Computer

The implementation on the computer was done as an extension to software developed by Emil Salling at FOI. This program was used to see if the vehicle behaved as it should.

When the implementation was done on the computer, three different simulations were performed. The first one found the closest way to point A with no consideration to the final orientation of the vehicle. The second one however found the closest way to point A and placed itself in a predetermined direction at that position. The third simulation chased another vehicle or moving target around until they ended up on the same place.

4.1.1 Closest Way to Point A With No Consideration to Final Orientation The first mission is to take the vehicle from its current position to a predetermined fixed position with no consideration to the final orientation of the vehicle. This is done due to the model presented in appendix A.1. The vehicle got to the decided position where it stopped.

4.1.2 Closest Way to Point A With Consideration to Final Orientation The previous model does not take into consideration the final orientation of the vehicle. Suppose we would like to go to a point and from that place look in a predetermined direction and the camera is rigidly attached to the vehicle. We then need to make some minor changes to the model. This part has also been coded in C++, and this can be viewed in appendix A.2.

Let R denote the minimum turning radius for the vehicle. Then make two circles with radius R around the final position of the vehicle as in figure 4.1. To find the center of the circles, we need to express the final orientation as a vector. The perpendicular vector to this points toward the center of the circle. Now the vector from the final position to the center of the circle can be obtained by multiply the perpendicular vector by the radius of the circle. This vector is added or subtracted to the target vector, to receive the two circles. To decide which one of these to aim for, the distances from the vehicle to the circles are obtained, the circle with the shortest distance is the one to aim for.

When the decision about the circle has been made, we should find the tangent (x_p, y_p) to this circle from the vehicles current position (x_v, y_v) (see equation 4.1–4.2 and figure 4.2). The two equations (4.1–4.2) are solved by Mathematica and the result is given by equation 4.3–4.4. Now the car will go toward that point instead of the target position (x_T, y_T).

$$\overline{v}_{pc} \cdot \overline{v}_{pc} = R^2 \tag{4.1}$$

$$\overline{v}_{vp} \cdot \overline{v}_{pc} = \overline{0}, \tag{4.2}$$

where $\overline{v}_{pc}$ is the vector between the tangent point and the center of the circle and $\overline{v}_{vp}$ the vector from the vehicle to the tangent point.

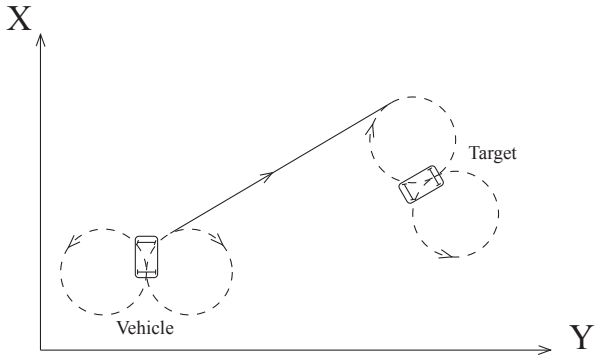


Figure 4.1: Control of Final Orientation

$$\Delta x_{pc} = \frac{R^2\Delta x_{vc}^2 \pm \Delta y_{vc}\sqrt{R^2\Delta x_{vc}^2(-R^2+\Delta x_{vc}^2+\Delta y_{vc}^2)}}{\Delta x_{vc}(\Delta x_{vc}^2+\Delta y_{vc}^2)} \tag{4.3}$$

$$\Delta y_{pc} = \frac{R^2\Delta y_{vc} \mp \sqrt{R^2\Delta x_{vc}^2(-R^2+\Delta x_{vc}^2+\Delta y_{vc}^2)}}{\Delta x_{vc}^2+\Delta y_{vc}^2}, \tag{4.4}$$

where R is the radius of the circle with centre at (x_c, y_c), $\Delta x_{vc} = x_v - x_c$, $\Delta y_{vc} = y_v - y_c$, $\Delta x_{pc} = x_p - x_c$ and $\Delta y_{pc} = x_p - x_c$.

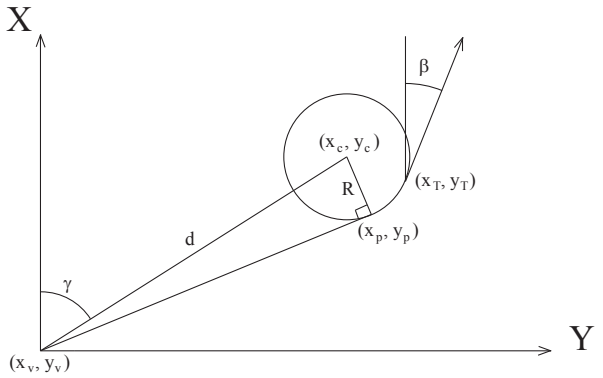


Figure 4.2: Explanation of Variables

As can be seen in equation 4.3 and 4.4 there is a $\pm$- or $\mp$-sign, this is because there is two tangent points to each circle, we need to find out which one of these points to aim for. We should aim for the tangent point that coincide with the direction of the circle. Let $\overline{v}_{Tc}$ be the vector from the final target position to the center of the circle and $\overline{v}_T$ denote the final orientation of the vehicle. In equation 4.5 it can be seen that the vector products are compared. If they have the same sign the equation will return a positive value which implies that the vector products coincide and the correct tangent point was choosen, if they have different signs the equation will return a negative value and the other tangent point is the correct one to aim for.

$$\frac{\overline{v}_{pc} \times \overline{v}_{pv}}{\overline{v}_{Tc} \times \overline{v}_T} \tag{4.5}$$

When this tangent point is reached we send the true final position to the system, the vehicle

now goes toward this point with more or less the maximum steering angle. Then the car end up in the correct position with the predetermined final orientation of the vehicle.

The path computed above are made of circular arcs tangentially connected by line segments, [8], but this is only locally optimal. The curvature profile of this path is not continuous. To follow the path precisely, the vehicle must stop and reorient at each curvature discontinuity.

For this project the model are good enough, but it can be improved by doing the curve continuous, [9, 7]. Optimal paths are proved to be made of line segments, circular arcs and pieces of clothoid [1].

4.1.3 Chasing a Moving Target The algorithms does not only work with fixed targets, without modification it can also be applied to the case of following or chasing a moving target.

A moving target (in this case an armoured car) was controlled via the keyboard, and the other vehicle was controlled by the developed model. Then our vehicle chased the armoured car until they ended up on the same place, then they stopped at that position. This case was solved by the fact that the targets position is continuously updated, and by that the direction to the target is updated. There is however a limitation to this problem that the chasing vehicle has to have a greater velocity than the chased one. Otherwise the chasing vehicle will not be able to catch up, but will still follow the target.

4.2 Implementation on the Radio-controlled Car

The intention was to implement the program on a radio-controlled car, but there was no time for that. The program is supposed to work on the RC-car without any major changes since simulations has been performed. The schematic view of the car can be seen in figure 4.3. The computer (PC104 in the figure) is going to be programmed with the code presented in appendix A. The car will then be able to perform different tasks such as those presented in the previous section (4.1) that was simulated.

[1] A clothoid is a curve whose curvature is a linear function of the arc length.

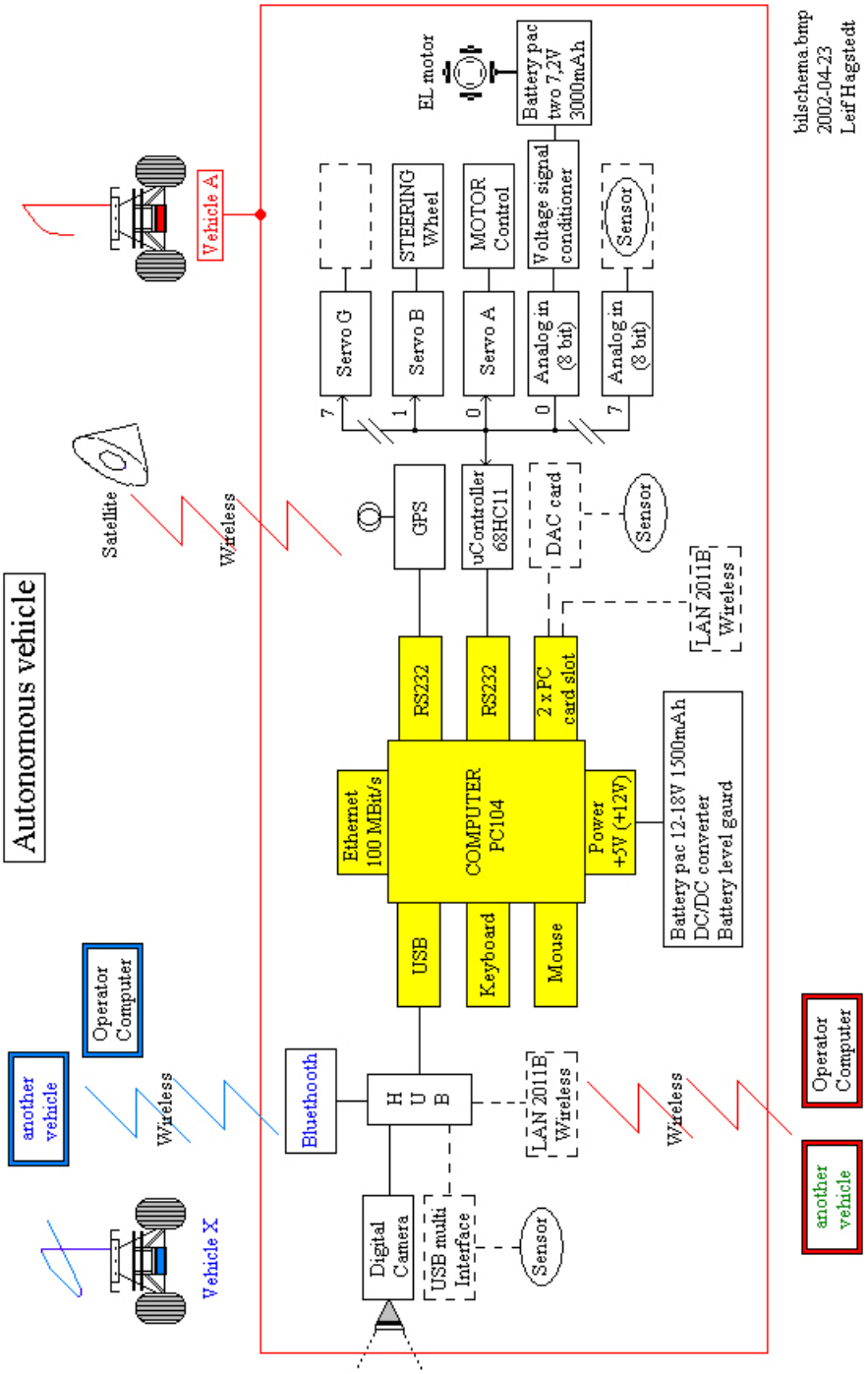


Figure 4.3: Schematic view of the car

Chapter 5

Conclusions

During this project an autopilot for unmanned ground vehicles was developed. The model was first simulated in Simulink, MATLAB and then coded in C++. A radio-controlled car were supposed to be used as platform to try out the model, but there was no time for that. The implementation was performed on the computer to see how it worked out. The RC-car that was going to be used is equipped with a computer to handle the program and information.

GPS was used as a navigation sensor from which position, velocity and course can be observed after some processing of the received data to control the vehicle. The vehicle was also equipped with some USB-ports to add other sensors to the system. A digital camera was mounted on the vehicle to reconnoitre the terrain and in future development this will be used to avoid obstacles.

The model was implemented on the computer and a simple algorithm was tried out. This algorithm was done to see how the vehicle behaved on certain signals, and the mission for the vehicle was to transport from its current position to a predetermined target. This implementation was split in two parts, one were the vehicle went to the final position with no consideration to the final orientation and one were the final orientation was taken into consideration. This latter implementation is done to make it easier to reconnoitre the terrain with the digital camera, and “look” in different directions.

A stability analysis was made, see section 2.1.3. This analysis was based on the dynamical model (2.1.2), and it showed that the model is stable for all velocities greater than zero as long as there is a limited maximum steering angle.

Chapter 6

Continued Work

This work includes the basics of an autopilot for autonomous ground vehicles. There are a lot of work that can be done to improve the model. First of all the model should be implemented on the RC-car. Second the dynamical model that was presented in section 2.1.2 should be implemented on the platform. Only the steering is controlled in this work, but it should probably not be a major problem to implement a model with speed control as well. The speed of the vehicle can be calculated with respect to the critical acceleration mentioned in section 2.1.1. To improve the model even more the length, steering signal, cornering stiffness, moment of inertia etc. for the platform can be more accurate.

At this time the model is built to take the closest way to a predetermined target point. This was done on a flat field with no obstacles. The model are also constructed to take the closest way to the target with a predetermined final orientation of the vehicle. Next step is to implement the problem presented in section 3.2.2, to take the closest way to point B via point A. Even in this case the vehicle should be able to have a controlled final orientation. Furthermore, the model will be improved to follow a path and finally be intelligent enough to take the best way to the target without hitting any obstacles in open terrain.

A model for computer vision will be developed for the platform. This will be used to obtain obstacles that need to be avoided during transportation.

The model will then be implemented, not just on ground vehicles as in this case, but also on aerial vehicles as well as underwater or surface vehicles. Different kind of vehicles will then perform commissions together as a group.

GPS was used in this work to guide the vehicle, this system can also be improved. The GPS delivers the position, orientation and speed of the vehicle, but it contains some errors that need to be minimized. This can partly be done by using DGPS (see appendix B.1). The accuracy of the GPS is continuously worked on. In a few years time the european system called GALILEO, see appendix B.3, can be used to increase the accuracy.

Appendix A

Coding in C++

The autopilot has been coded in C++. There were two algorithms that were implemented. The first one can guide the vehicle the closest way to a predetermined target with no consideration to final orientation of the vehicle. The second one can relocate the vehicle while considering the additional constraint of a certain predetermined orientation as the vehicle stops near the target. These two algorithms are further explained in the following sections.

A.1 Coding of Closest Way to Target

The algorithm in this case was split in two parts, the *VehicleSteeringController* and *VehicleThrustController*. These two parts decides what signals are going to be sent to the vehicle.

A.1.1 Calculation of Steering Signal The calculation about the steering angle follows from section 2.1.1, that describes what decisions has to be made due to the position and orientation of the vehicle and the target. These calculations has been translated to C++-code, which can be viewed in C++-code A.1.

C++-code A.1 calcSteerAngle

```
float
VehicleSteeringController::calcSteerAngle() {
  angleDiff = mCar->getAttitBV().getAzimuth( (mTarget->
              getPositionE()-mCar->getPositionE()) );

  if( abs(angleDiff)>MAXIMUMSTEERINGANGLE ) {
    steerAngle = sign(angleDiff)*MAXIMUMSTEERINGANGLE;
  }
  else {
    steerAngle = angleDiff;
  }

  return steerAngle/MAXIMUMSTEERINGANGLE;
}
```

As can be seen in C++-code A.1, the difference in angles are being calculated. Then the decision about turning right or left is being made. Before the steering signal is sent, it is divided by the maximum steering angle. The steering signal must be within the range $[-1, 1]$, that is why it is divided by the maximum steering signal.

A.1.2 Calculation of Velocity Signal In this algorithm we use two velocities, a lower one and a higher one, and of course zero when we stop. To make a decision about the velocity we need to know about the distance to the target as mentioned in section 2.1.1. We also need to take into account about the steering angle. The distance is being calculated as shown in C++-code A.2.

C++-code A.2 calcVehicleTargetSeparation

```
void
VehicleThrustController::calcVehicleTargetSeparation() {
  currentDistanceToTarget = Magnitude(
                                  (mTarget->getPositionE()-
                                   mCar->getPositionE()) );
}
```

The *VehicleThrustController* now calculates the velocity signal that are going to be sent to the vehicle, see C++-code A.3.

C++-code A.3 calcVelocity

```
float
VehicleThrustController::calcVelocity() {
  if( abs( steerAngle )>SMALLMAXIMUMSTEERINGANGLE ) {
    velocity = LOW_VELOCITY;
  }
  else {
    if(  currentDistanceToTarget>RANGE_LIMIT*ADOMAIN ) {
      velocity = HIGH_VELOCITY;
    }
    else {
      velocity = LOW_VELOCITY;
    }
  }
  if( currentDistanceToTarget<RADIUS/2.0f ) {
    velocity = 0.0f;
  }
  return velocity;
}
```

First of all it finds out if the steering angle is greater than $\pm\delta_{\text{narrow}}$, see section 2.1.1. If this is true, then the lower velocity is sent to the car. Then it takes into account about the distance to the target. If the distance is greater than a predefined range of the target, then the higher velocity is sent, else the lower one if we are within this range of the target.

The velocity signal must as the steering signal be within the range $[-1, 1]$, so the lower and the higher velocities are set to 1.0 respective 0.5.

A.2 Coding of Closest Way to Target with Final Orientation Control

This algorithm consists of two parts, *VehicleThrustController* and *FinalOrientationController*. The first part is the same as above, appendix A.1.2 and the latter part is further explained in the following section.

A.2.1 Calculation of Steering Signal with Final Orientation Control Section 4.1.2 explains the theory about how to go to the final position with a predefined final orientation. This theory has been translated to C++-code as follows.

First of all we need to find the center of the circle mentioned in section 4.1.2, this is done by the code presented in C++-code A.4. As mentioned there are two circles, one at each side as can be seen in figure 4.1. There is now a decision that has to be made, which one of these circles to aim for. This decision is based on the distance from the current vehicle position and the center of the circles, the car should aim for the circle with the shortest distance.

The tangent to this circle can then be obtained by using C++-code A.5. Since there are two

C++-code A.4 centerOfCircle

```
Vector3f
FinalOrientationController::centerOfCircle() {
  Vector3f targetYaxisInE = RADIUS*mTarget->
          getAttitBV().getEfromB( Vector3f::yAxis );

  float distanceToCircle1 = Magnitude( (mTarget->getPositionE()
          +targetYaxisInE)-mCar->getPositionE() );
  float distanceToCircle2 = Magnitude( (mTarget->getPositionE()
          -targetYaxisInE)-mCar->getPositionE() );

  if( distanceToCircle1 < distanceToCircle2 ) {
    return mTarget->getPositionE()+targetYaxisInE;
  }
  else {
    return mTarget->getPositionE()-targetYaxisInE;
  }
}
```

tangent points on each circle we need to make a decision about which one of these points we would like to go towards. This decision is made due to the theory presented in section 4.1.2, equation 4.5. The car are now supposed to go toward the tangent point. To calculate the steering angle, this point is sent as an argument to C++-code A.6, which returns the angle to send to the car. When the car has reached the tangent point it is supposed to go toward the final position, this is done by sending the final position instead of the target point to the part that calculates the steering angle (C++-code A.6).

C++-code A.5 tangentToCircle

```
Vector3f
FinalOrientationController::tangentToCircle() {
  Vector3f circleCenter = centerOfCircle();
  Vector3f carToCircleVec = circleCenter-mCar->getPositionE();
  Vector3f targetXaxisInE = mTarget->getAttitBV().getEfromB(
                                  Vector3f::xAxis );

  float VPx = mCar->getPositionE().x();
  float VPy = mCar->getPositionE().y();
  float CCx = circleCenter.x();
  float CCy = circleCenter.y();
  float vX = VPx-CCx;
  float vY = VPy-CCy;

  float slask = (-pow(RADIUS,2)+pow(vX,2)+pow(vY,2));
  if( slask<0.0f ) {
    slask = 0.0f;
  }

  float XPart = vY*sqrt(pow(RADIUS,2)*pow(vX,2)*slask);
  float YPart = sqrt(pow(RADIUS,2)*pow(vX,2)*slask);

  float tX1 = (pow(RADIUS,2)*pow(vX,2)-XPart)/
                    (vX*(pow(vX,2)+pow(vY,2)));
  float tY1 = (pow(RADIUS,2)*vY+YPart)/(pow(vX,2)+pow(vY,2));

  float tX2 = (pow(RADIUS,2)*pow(vX,2)+XPart)/
                    (vX*(pow(vX,2)+pow(vY,2)));
  float tY2 = (pow(RADIUS,2)*vY-YPart)/(pow(vX,2)+pow(vY,2));

  mTargetPosition.x() = circleCenter.x()+tX1;
  mTargetPosition.y() = circleCenter.y()+tY1;
  mTargetPosition.z() = mTarget->getPositionE().z();

  float tangent = ( (mTargetPosition.x()-circleCenter.x())
           *(mCar->getPositionE().y()-mTargetPosition.y())
           -(mTargetPosition.y()-circleCenter.y())
           *(mCar->getPositionE().x()-mTargetPosition.x()) );
  float target = ( (mTarget->getPositionE().x()-circleCenter.x())
           *targetXaxisInE.y()
           -(mTarget->getPositionE().y()-circleCenter.y())
           *targetXaxisInE.x() );

  if( tangent/target > 0.0f ) {
    return mTargetPosition;
  }
  else {
    mTargetPosition.x() = circleCenter.x()+tX2;
    mTargetPosition.y() = circleCenter.y()+tY2;
    mTargetPosition.z() = mTarget->getPositionE().z();
    return mTargetPosition;
  }
}
```

C++-code A.6 calcSteerAngle

```
float
FinalOrientationController::calcSteerAngle( const Vector3f&
                                                targetPosition ) {
  angleDiff = mCar->getAttitBV().getAzimuth( (targetPosition-
                                    mCar->getPositionE()) );


  if( abs(angleDiff)>MAXIMUMSTEERINGANGLE ) {
    steerAngle = sign(angleDiff)*MAXIMUMSTEERINGANGLE;
  }
  else {
    steerAngle = angleDiff;
  }

  return steerAngle/MAXIMUMSTEERINGANGLE;
}
```

Appendix B

Basics About the Navigation System

In this work we use the Global Navigation Satellite System, GNSS, to obtain position and velocity of the vehicles. GNSS is a generic term for global navigation systems based on satellites. Global Positioning System, GPS is a part of the GNSS and GPS is in this work used to find out about the vehicles position, direction and speed.

B.1 Global Positioning System

GPS is funded by and controlled by the U.S. Department of Defense, DoD. The system are being used by thousands of civilians, but it was designed for and is operated by the U.S. military.

GPS consists of 24 satellites in 6 planes that orbit the Earth in 12 hours and on an altitude of approximately 20 200 kilometers, see figure B.1[1].

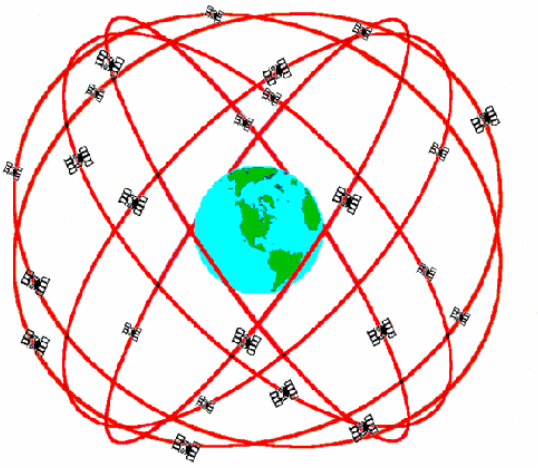


Figure B.1: The satellites that orbit the earth

Signals from four satellites are required to compute a receivers position in space and time. The height is not needed in this research why only three satellites are required, of course the time is needed as well, therefore the third satellite. The aim is that the satellites should be as far away from each other as possible due to minimization of the error.

Some stations with very accurate positions on the Earth are placed on the ground to receive data from the satellites. The satellites and the stations are equipped with atomic clocks that are extremely exact, the error growth rate is about one second in 300 000 years. Each satellite has four clocks, one that is used and three spare.

Since the speed of light is known, and the time it takes for the signal to be sent can be measured, the distance to the satellite can be calculated by multiplying the time with the

[1] http://www.colorado.Edu/geography/gcraft/notes/gps/gps_f.html

speed of light. For example, if the satellite is straight above the receiver, it takes approximately 0.067 seconds for the signal to reach the receiver. Now, since the speed of light is approximately 300 000 kilometers per second this results in an altitude of 20 100 kilometers for the satellite. Of course these calculations are done with greater numerical accuracy, to obtain more precise information.

Positions can be gathered from a map and then stored in a computer. These positions can then be put together to maintain a route to follow. When this route is followed, new positions are obtained from the GPS, and if they diverge from each other, corrections are being made.

There are two kinds of services available, the Standard Positioning Service, SPS, used by the civilians, and the Precise Positioning Service, PPS, used by the U.S. military. The difference between these systems is the precision and disturbances from other parts. The radiosignals that are sent slow down in the atmosphere, this delay effects the accuracy since it depends on the thickness of the atmosphere, which varies over time and with the weather. Another thing that affects the result is the receiver's electronics. A lot of other things affects the accuracy but these are the main ones.

Civilians can use a method to increase the accuracy, this is called Differential GPS, DGPS, figure B.2[2].

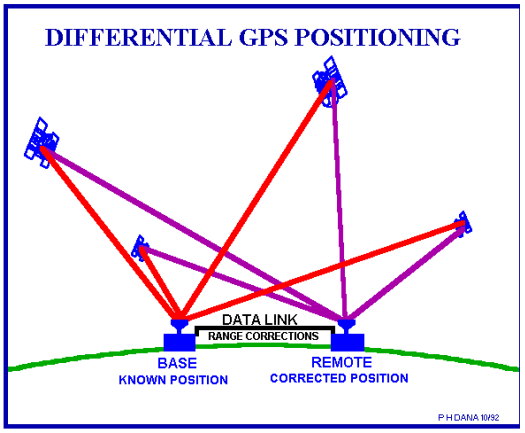


Figure B.2: Differential Global Positioning System

This system works in a way where you have a fixed reference station with known position. This station is equipped with a GPS-receiver and gets the position from this, but since it already knows its position, the error can be calculated. This error is sent by another communication to GPS-receivers nearby, where it can be as a correction to the computed position.

B.2 The RT90 System

The GPS-receiver used in this project has a precision of less than 10 meters. To increase this precision some fixed receivers with known positions can be used, the DGPS above, then the accuracy can be as good as about one meter, or even better. For the car, the position is converted to RT90, or SWEREF99, figure B.3[3].

[2] http://www.colorado.Edu/geography/gcraft/notes/gps/gps_f.html
[3] http://www.lm.se/geodesi/

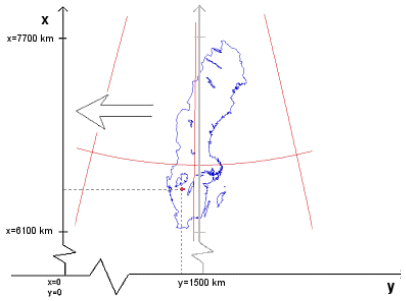


Figure B.3: The RT90-system used in this thesis

B.3 GALILEO

GPS is as mentioned above a system developed for the U.S. military. It has been decided that Europe is going to develop a similar system called GALILEO, an initiative launched by the European Union and the European Space Agency. This satellite radio navigation system is supposed to have an accuracy of about a metre.

GALILEO is based on a constellation of 30 satellites and ground stations providing information about the position.

Advantages over GPS:

- GALILEO has been designed and developed as a non-military application.
- It provides a similar - and possibly higher - degree of precision, due to the constellation of satellites and the ground based control and management systems planned.
- GALILEO is more reliable as it informs the user of any errors.
- It guarantees continuity of service. GPS signals can become unavailable, sometimes without prior warning.

GALILEO and GPS complement each other by the fact that they are independent. The users will be able to receive both GALILEO and GPS signals with the same receiver.

Bibliography

[1] J. Ackermann. *Robust Control*. Springer-Verlag, London, 1993.

[2] G. Campion, G. Bastin, and B. D'Andréa-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. *IEEE Trans. Robot. Automat.*, 12, February 1996.

[3] C. Canudas de Wit, B. Siciliano, and G. Bastin. *Theory of Robot Control*. Springer-Verlag, London, 1996.

[4] M. Egerstedt. *Motion Planning and Control of Mobile Robots*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2000.

[5] M. Egerstedt, X. Hu, H. Rehbinder, and A. Stotsky. Path planning and robust tracking for a car-like robot. In *Proceedings of the 5th Symposium on Intelligent Robotic Systems*, pages 237–243, Stockholm, Sweden, July 1997.

[6] M. Egerstedt, X. Hu, and A. Stotsky. Control of a car-like robot using a virtual vehicle approach. In *IEEE Conference on Robotics and Automation*, volume 4, May 1998.

[7] Th. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Seoul, May 2001.

[8] P. Jacobs and J. Canny. Robust motion planning for mobile robots. In *IEEE Conference on Robotics and Automation*, pages 2–7, 1990.

[9] A. Scheuer and Ch. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 25–31, October 1998.

[10] A. Stotsky, X. Hu, and M. Egerstedt. Sliding mode control of a car-like mobile robot using single-track dynamic model. In *Proceedings of the IFAC'99: 14th World Congress*, Beijing, China, July 1999.

[11] N-G. Vågstedt. *On Cornering Characteristics of Ground Vehicle Axles*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1995.