Building a MATLAB based Formula Student simulator

Criens, C.H.A.; ten Dam, T.; Luijten, H.J.C.; Rutjes, T.

Published: 01/01/2006

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Building a MATLAB based Formula Student simulator

C.H.A. Criens, T. ten Dam,
H.J.C. Luijten, T. Rutjes

DCT 2006.069

Master Team Project report

Supervisors:
Dr. Ir. A.J.C. Schmeitz
Dr. Ir. I.J.M. Besselink

Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Eindhoven, (June 20, 2006)

Samenvatting

Formula Student Racing Team Eindhoven (FSRTE) is een team van studenten dat deelneemt aan internationale racewedstrijden met een zelfgebouwde, eenpersoons racewagen. Een belangrijk onderdeel waar de studenten zelf zorg voor moeten dragen is de financiering van het project. Dit houdt in dat er sponsors gezocht moeten worden om alle kosten te kunnen dekken. Om mogelijke sponsors over de streep te trekken, is het noodzakelijk om serieus en ingenieus voor de dag te komen en een eigen, realistische racesimulator zou een goede indruk kunnen maken bij eventuele sponsors. Een realistische racesimulator zou ook gebruikt kunnen worden bij het trainen van de coureurs. Ze kunnen het circuit in de simulator goed verkennen voordat ze met de echte auto gaan rijden. Ook de ontwikkeling van de auto zou met behulp van een simulator sneller kunnen gaan. Aanpassingen en instellingen zouden eerst in de simulator getest kunnen worden, voordat ze in de echte auto geïmplementeerd worden.

De simulator zal met behulp van MATLAB/Simulink gemaakt worden, omdat deze software alle drie de onderdelen van de simulator ondersteunt. De drie onderdelen waaruit de simulator bestaat, zijn de besturing, het voertuigmodel en de visualisatie. Voor de besturing is interactie met een racestuur nodig en hiervoor bestaan standaard blokken in de 'Virtual Reality Toolbox'. Ook de benodigde blokken voor de visualisatie zijn terug te vinden in deze toolbox. De driedimensionale wereld en de visualisatie van de auto zijn gemaakt in de programmeertaal VRML, omdat deze taal gebruikt wordt door de visualisatieblokken in MATLAB. Het meegeleverde programma 'V-Realm Builder' kan als VRML-editor gebruikt worden. Er zijn ook nog extra dingen toegevoegd aan de visualisatie, zoals de rondetijd en veranderende oogpunten.

Het voertuigmodel bestaat uit enkele gekoppelde onderdelen, namelijk het dynamische model, de aandrijving en het bandmodel. Het dynamische model dat is geïmplementeerd, is gebaseerd op het fietsmodel. Dit is een relatief simpel model dat slechts twee wielen verondersteld, maar het wordt algemeen geaccepteerd als voertuigmodel. Als er in een later stadium voor gekozen wordt om een meer uitgebreid dynamisch model te gebruiken, dan is dit eenvoudig te vervangen. In het aandrijfgedeelte wordt de hoeksnelheid van de wielen bepaald met behulp van de krachten op het model. Voor de modellering van de motor en de transmissie zijn gegevens gebruikt die afkomstig zijn van metingen die door FSRTE zijn uitgevoerd. Het bandmodel is zo gekozen dat 'combined slip' mogelijk wordt, omdat dat voor een realistische racebeleving gewenst is. De precieze parameters voor het bandmodel zijn op het moment van dit project niet bekend, maar deze kunnen naderhand eenvoudig aangepast worden.

Om de race-ervaring nog realistischer te maken, en om de simulator leuker te maken voor het publiek en sponsors, zijn er nog enkele dingen toegevoegd, namelijk geluid en 'force feedback'. Het geluid bestaat uit het geluid van de motor en het geluid van de banden als deze slippen. De 'force feedback' die is toegevoegd, bestaat uit trillingen zodra de auto van de baan gaat en een 'self aligning moment'.

Uiteraard moet een goede racesimulator gelijk lopen met de echte tijd. Om dit te bewerkstelli-

gen in Simulink, is gebruik gemaakt van een blok uit een speciale toolbox, gemaakt door Werner Zimmerman. Dit blok zorgt ervoor dat de simulatie real-time loopt.

Na de implementatie van het geheel, bleek dat de totale simulator met visualisatie veel van een computer eist. Daarom is de mogelijkheid ingebouwd om de visualisatie en het geluid op een andere computer te draaien. Hiervoor is gebruik gemaakt van het communicatie protocol UDP. Dit protocol maakt het mogelijk om te communiceren tussen computers over een netwerk en kan dus gebruikt worden op de werklast te verdelen over verschillende computers.

De verkregen simulator kan nu dus gebruikt worden voor promotie en voor de ontwikkeling van de auto van komend jaar. De ontwikkeling van deze auto leverde echter nog een probleem op, namelijk de vraag er volgend jaar een lichte auto met een lichte motor of een zwaardere auto met een zware motor gebouwd moet worden. Om hier iets over te kunnen zeggen is een rondetijdsimulatie uitgevoerd.

Deze simulatie is gebaseerd op een simpel voertuigmodel dat bestaat uit een puntmassa en een band met beperkte grip, afhankelijk van de massa. Het circuit is gemodelleerd door op elk punt een kromming te definiëren. Met behulp van deze kromming, het vermogen van de motor en de maximale grip, kan de maximale versnelling op elk punt van de baan uitgerekend worden. De simulatie is ook in tegengestelde richting uitgevoerd, om ervoor te zorgen dat de maximale remkracht niet wordt overschreden. Het minimum van beide simulaties geeft de maximale snelheid op elk punt van het circuit en het resultaat is de laagst haalbare rondetijd. Deze rondetijd zal in de praktijk niet gehaald worden, omdat een echte coureur nooit continu op de rand van de wrijvingscirkel zal rijden.

De uitkomst van deze simulatie geeft aan dat er maar een heel klein verschil in rondetijd zal zijn tussen een zware en een lichte auto. Een gevoeligheidsanalyse toont echter aan dat het gewicht een zeer belangrijke factor speelt in de prestaties van de auto, dus dit moet een belangrijke rol spelen bij het ontwerp van de nieuwe auto.

Summary

Formula Student Racing Team Eindhoven (FSRTE) is a team of students that participates in international racing contests with a self made, single seated racecar. An important aspect for the students to take care of is the finances of the project. They have to look for sponsors themselves to raise enough money for the project to be a success. In order to persuade possible sponsors, a serious and ingenious attitude is necessary and a self-made, realistic race simulator might be the final key to get a sponsor on board. A realistic simulator could also be used to train the driver, because he is able to get to know the circuit before he enters the real car. Also the development of the car can be improved with the help of a simulator. Adjustments and tuning of the car can first be tested in the simulator, before they are applied to the real car.

This simulator will be built using MATLAB/Simulink, because this software package supports all three parts of the simulator, namely the user input, the vehicle model and the visualization. The user input is done by connecting a steering wheel to a computer and this can be done using the standard blocks in the 'Virtual Reality Toolbox' in Simulink. This toolbox contains also the blocks, necessary for the visualization. A three-dimensional world and the visualization of the car are made with the programming language VRML, because the blocks in Simulink work with this language. An editor for this language is delivered with MATLAB and is called 'V-Realm Builder'. For a more appealing simulator, some extra features as laptime and changing viewpoints are included in the visualization.

The vehicle model consists of some coupled parts, namely the dynamical model, driveline and the tyre model. As a dynamical model, the bicycle model is used. This is a relatively simple model that includes only two wheels, but it is widely accepted as a vehicle model. When a more sophisticated dynamical model is desired, this can easily be adjusted. In the part of the driveline, the angular velocity of the tyres is calculated according to the forces acting on the model. To model the engine and the transmission, measured data from FSRTE is used. The choice for the tyre model depended on the ability to have combined slip, because this feature is desired for a realistic simulator. At the moment of this project, the parameters for the tyre model are not exactly known, but when they are, they can easily be inserted.

To make the driving experience even more realistic, and to make the simulator more appealing to the audience and sponsors, sound and force feedback are added. Sound consists of the noise of the engine and the slipping tyres. Force feedback includes off track vibrations and a self aligning moment.

It is obvious that a simulator needs to run in real time in order to be realistic. To achieve this goal, a block from a special toolbox made by Werner Zimmerman is used. This block makes every simulation run in real time.

After implementation of the whole model, it turned out that the simulator and all its contents require a very fast computer. Therefore the possibility to the visualization and the sound on different computers is included. In order to do this, the communication protocol UDP is used. This

protocol makes fast communication between computers possible when they are connected to a network.

The obtained simulator can now be used for promotion and the development of next year's car. There is however an unanswered question regarding this car, namely the question whether a light car with a small engine is better than a heavier car with a big engine. To be able to tell something about the right decision, a lap time simulation is done.

This simulation is based on a simple vehicle model including a point mass and a tyre with limited grip, depending on the mass only. The circuit is transformed into a curvature at all points. With this curvature, together with the maximum engine power and the maximum grip, the maximum acceleration of the car can be calculated. This simulation is also done in reverse, to make sure the braking force does not exceed its maximum. The minimum of these simulations gives the maximum speed at any point on the track and the result is the lowest possible lap time. This lap time will never be reached at the real circuit, because a real driver can not drive on the edge of the friction ellipse.

The outcome of this simulation shows that there is only a minor difference between the lap times of the light car and the heavy car. However a sensitivity analysis shows that weight is a very important factor for the performance of a car, so this should be an important factor in the new car's design.

Contents

List of Figures

Notation

Symbol	Quantity	Unit
a	Acceleration	$[ms^{-2}]$
A	Area	$[m^2]$
C_d	Drag coëfficiënt	$[-]$
F	Force	$[N]$
g	Gravitational acceleration	$[ms^{-2}]$
i	Ratio	$[-]$
I	Mass moment of inertia	$[kgm^2]$
m	Mass	$[kg]$
M	Moment	$[Nm]$
P	Power	$[W]$
r	Angular velocity	$[rads^{-1}]$
R	Radius of curvature	$[m]$
s	Distance	$[m]$
t	Time	$[s]$
u	Longitudinal speed	$[ms^{-1}]$
v	Lateral speed	$[ms^{-1}]$
V	Combined lateral and longitudinal speed	$[ms^{-1}]$
α	Side slip angle	$[rad]$
δ	Steer angle	$[rad]$
κ	Longitudinal slip	$[-]$
ρ	Density	$[kgm^{-3}]$
σ	Combined slip	$[-]$
μ	Friction coëfficiënt	$[-]$
Ω	Rotational Speed of the wheel	$[rads^{-1}]$

Chapter 1

General Introduction

Formula Student Racing Team Eindhoven (FSRTE) is a team of students that participates in a project called Formula Student. This is a project for engineering students to design and build a small single seated racing car. The project is part of their academic studies, and culminates in a competition where teams from all over the world come to compete against each other. In the designing process a lot of questions and problems arise. Therefore some of these questions and problems are handed over to study in related project groups. In this report two of these problems are worked out.

1.1 Motivation

Since developing a race car is very expensive, promotion and sponsoring is very important. To persuade potential sponsors, an appealing promotion stand is desired. Part of this stand could be a real-time simulator, in which the funder could take place and get enthusiastic about the project. Moreover, a simulator is needed to get insight in the car behavior and let drivers get familiar with the track.

Every year a new car is designed. For the design of next years Formula Student car of Eindhoven University two main designs are being considered. The first would be an improved version of the FSRTE02, i.e. an aluminium sandwich chassis weighing about 225 kg having a 60 kW Suzuki GSX-R600 engine. The second alternative design would be a 125 kg carbon fiber chassis similar to the DUT04 using a 41 kW Aprilia SXV5.5 engine. In a very early design stage, a choice has to be made between these two options.

1.2 Aims and scope

The goal for the real-time simulator is to be able to drive the formula student race car in a three dimensional environment in such a way that the dynamics meet reality as good as possible. This should be done using MATLAB Simulink for the model and the visualization. For the input a Logitech MOMO steering wheel will be used. The end product must be appealing and user friendly, because it is the intention to use it in formula student promotion stands.

The goal for the lap time simulation is to be able to decide on the next years car design. Moreover, the aim is to get a better grip on important design parameters. This will be done by a lap time simulator that calculates lap times with use of a simple car model and track information.

On the basis of the lap times and changes in the model parameters, decisions can be made for next years design.

1.3 Contents of this report

In part I of this report the driving simulator is presented. At first a description of the dynamical model, which is divided in vehicle dynamics, tire dynamics, drive train and brakes, is presented. This is followed by a chapter about the input in MATLAB Simulink. Then the visualization and the realization of the sound are presented. After this the force feedback model is explained and a description is given of how the simulator runs in real-time. Before closing with the conclusions and recommendations a chapter is used to outline how the workload is divided and how all parts are coupled.

In part II the mathematical calculations of the two designs and the outcome will be explained. First some information will be given about the model that is developed. After that the algorithm used to solve the model will be explained. Next, some results are presented. In the last chapter of this part the accuracy of the model and the sensitivity of the outcome to changes in parameter values will be examined. Finally, conclusions and recommendations are drawn.

Part I

Realtime Simulator

Chapter 2

Dynamic model

2.1 Vehicle Dynamics

Since our model has to run in real-time, the extent of the model is rather limited. Therefore an extended version of the simple bicycle model [5] is used. This model can give, to some extent, realistic vehicle behavior.

2.1.1 Modeling

The bicycle model is a model of a car that uses only two tyres, one front and one rear tyre. Roll of the vehicle can not be implemented in this model. In figure 2.1 a simplified drawing of the used model can be found.

Because the coordinate system is fixed on the car, the general equations in the horizontal-plane are:

$$\begin{aligned} m\left(\dot{u} - vr\right) &= \sum F_{La} \\ m\left(\dot{v} + ur\right) &= \sum F_{Lo} \\ I\dot{r} &= \sum M \end{aligned} \tag{2.1}$$

In this equation m is the mass of the vehicle, u the longitudinal speed, v the lateral speed, r the rotation of the vehicle around the center of gravity, $\sum F_{La}$ the sum of the lateral forces,

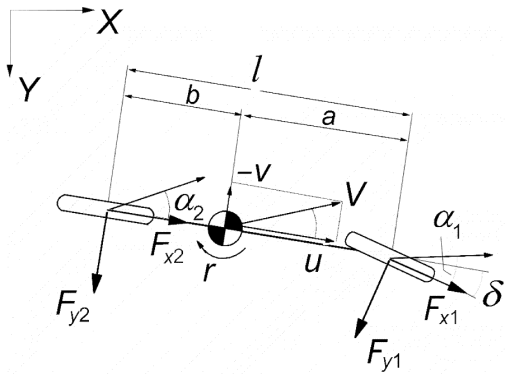

Figure 2.1: The bicycle model. (source: [2])

$\sum F_{Lo}$ the sum of the longitudinal forces, I the mass moment of inertia and $\sum M$ the sum of the moments around the center of gravity. Next $\sum F_{La}$ and $\sum F_{Lo}$ have to be specified. $\sum F_{La}$ is composed of the lateral force of the front and rear tyre, $\sum F_{Lo}$ is composed of the longitudinal parts, calculated as follows:

$$\begin{aligned} \textstyle\sum F_{La} &= F_{x1}\cos\delta - F_{y1}\sin\delta + F_{x2} - F_{airdrag} \\ \textstyle\sum F_{Lo} &= F_{x1}\sin\delta + F_{y1}\cos\delta + F_{y2} \\ \textstyle\sum M &= a\left(F_{x1}\sin\delta + F_{y1}\cos\delta\right) - bF_{y2} \end{aligned} \tag{2.2}$$

In this equation F_{x1} and F_{x2} are the longitudinal forces on the front and rear tyre, F_{y1} and F_{y2} the lateral forces on the front and rear tyre, $F_{airdrag}$ the air drag, δ the steer angle and a and b the distance between the center of gravity and the front and rear tyre. The forces F_{x1}, F_{x2}, F_{y1} and F_{y2} follow from the tyre model as explained in section 2.2. $F_{airdrag}$ is speed dependent, it can be calculated using:

$$F_{airdrag} = \frac{1}{2}\rho A u^2 \tag{2.3}$$

In this equation ρ is the density and A the frontal area.

2.1.2 Implementation

The implementation of this model in Simulink is relatively straightforward. As inputs the forces F_{x1}, F_{x2}, F_{y1} and F_{y2} and the steering angle δ are used. From this the forces F_{La} and F_{Lo} can be calculated. Using three integrators the dynamic equations from 2.1 can be solved. The solution gives the speed of the vehicle in all directions. From these speeds the position can be calculated and visualized.

To be able to solve the model, the tyre forces have to be known. How these forces are obtained is described in the next section.

2.2 Tyre Mechanics

The tyres determine the amount of grip. The tyre model should enable the simulator to give under- and oversteer, wheel spin or lock and limit the corner speed. Also, there should be a connection between the grip in longitudinal and lateral direction.

2.2.1 Modeling

The model used in the simulator consists of two parts. The first part calculates the slip conditions of the tyre. The second part calculates the tyre forces. For the calculation of the slip conditions the rotational speed of the tyre and the speed of the car in various directions are used. From this the wheel slip angle α and the longitudinal slip κ are calculated. For the computation of α, it is assumed that α is small. This is, as long as you are not spinning, a reasonable estimate. In [5] the following formulas for the computation of the slip angles can be found.

$$\begin{aligned} \alpha_1 &= \delta - \tfrac{v+ar}{u} \\ \alpha_2 &= -\tfrac{v-br}{u} \\ \kappa_i &= \tfrac{u-r_{tyre}\Omega_i}{u} \end{aligned} \tag{2.4}$$

In these equations Ω_i is the angular velocity of the tyre and r_{tyre} the tyre radius. The calculated slip quantities are combined using formulas from the section Combined Slip Conditions in [5]. A theory is presented there for modeling non-isotropic tyres using only α and κ. This is done using the following formulas.

$$\begin{aligned} \sigma_x &= \tfrac{\kappa}{1+\kappa} \\ \sigma_y &= \tfrac{\tan\alpha}{1+\kappa} \\ \sigma &= \sqrt{\sigma_x^2 + \sigma_y^2} \end{aligned} \tag{2.5}$$

The resulting combined slip, σ, is used to calculate the tyre forces. This is done using two general magic formulas for each tyre. One for the longitudinal force, $F_x(\sigma)$, and one for the lateral force, $F_y(\sigma)$, as given in the next equation:

$$F(\sigma) = D\sin\left[C\arctan\left\{B\sigma - E\left(B\sigma - \arctan\left(B\sigma\right)\right)\right\}\right] \tag{2.6}$$

In this equation B, C, D and E are tyre dependent parameters. With these magic formulas, together with combined slip, the resulting longitudinal an lateral forces on each tyre can be calculated as follows:

$$\begin{aligned} F_x &= \tfrac{\sigma_x}{\sigma} F_{x0}(\sigma) \\ F_y &= \tfrac{\sigma_y}{\sigma} F_{y0}(\sigma) \end{aligned} \tag{2.7}$$

The parameters for our simulator are only estimated to give reasonable results. To get a simulator that gives more accurate output it is recommended to investigate this further.

2.2.2 Model validation

Validation of the tyre mechanics is done by doing simulations with the tyre model only, in order to obtain the relation between the tyre forces and the slip conditions. The results of the simulations are shown in figure 2.2 and figure 2.3. These figures show a realistic relation between force and slip condition and therefore the implemented tyre mechanics are considered reliable.

2.3 Drive train

A car's drive train or power train consists of all the components that generate power and deliver it to the road surface. This includes the engine, transmission, drive shafts, differentials, and the final drive.

2.3.1 Modeling

To realistically model the engine in the simulator, data from FSRTE measurements is used. In figure 2.4 the engine velocity is plotted against the torque. The engine velocity varies from 3000 to 15000 rev/min, so stationary the engine has a velocity of 3000 rev/min.

The transmission is also based on values measured by FSRTE, see table 2.1. The primary reduction is the first reduction from the crankshaft to the gear box. The gear box consists of 6 gears. The final reduction is from the last gear wheel to the gear wheel of the differential. For this final reduction two sets of gear wheels are available. One will be especially used for the acceleration test, the other one for the endurance test.

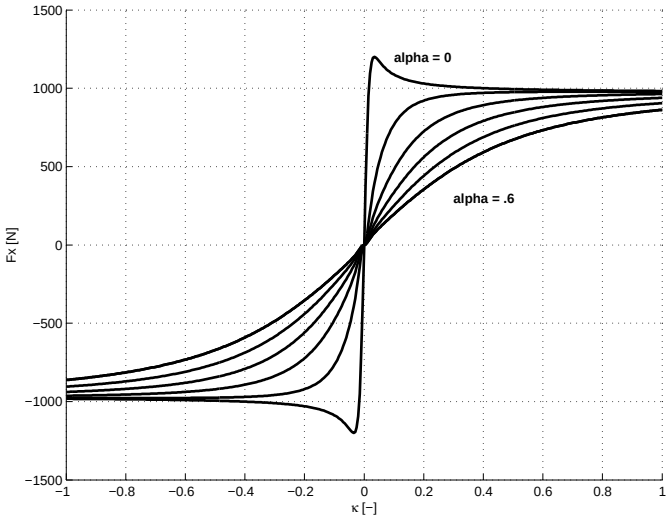

Figure 2.2: Tyre behavior: $F_x - \kappa$.

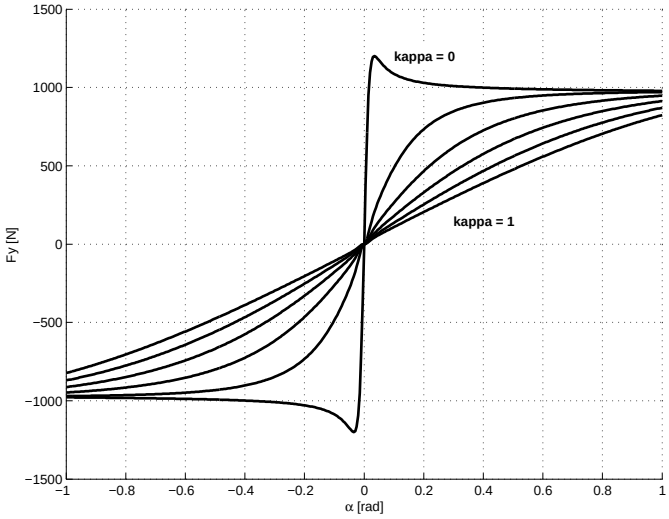

Figure 2.3: Tyre behavior: $F_y - \alpha$.

Table 2.1: Gear ratios

Gear	ratio
Primary reduction	(79/41)
1st	(39/14)
2nd	(32/16)
3th	(32/20)
4th	(30/20)
5th	(29/24)
6th	(25/23)
Final reduction	(50/14) or (43/16)

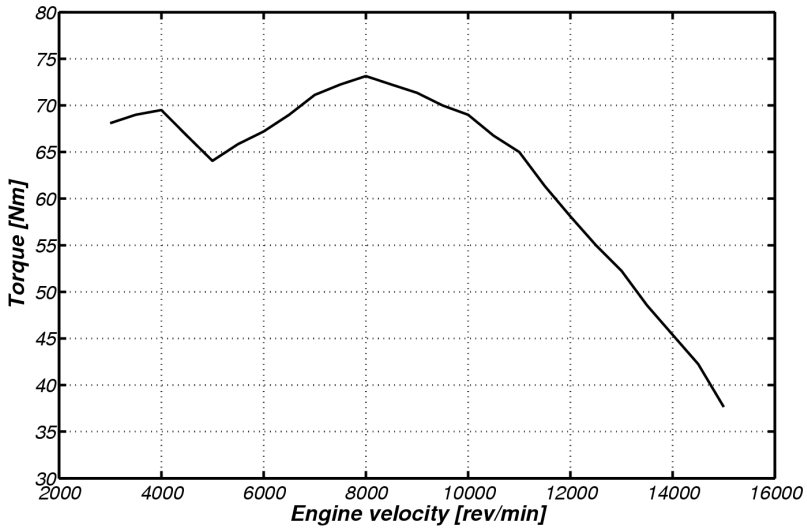

Figure 2.4: Engine characteristics

2.3.2 Implementation

The drive train has as inputs: gear, brake, throttle and the forces calculated in the tire model. How these forces are used to calculate the brake forces is described in section 2.4. To avoid problems when the car is almost at rest, the speed is set to zero when it reaches a certain minimum. For now this minimum is set to 0.05 rad s^{-1}. This implementation can be found in figure 2.5

2.4 Brakes

In order to model the driveline correctly, the moment applied to the wheels needs to be adjusted. These are namely slowed down by braking, gravity and drag. These three moments on the back wheels are calculated using the next formulas:

$$\begin{aligned} M_b &= (1-i) \cdot F_b \cdot r_{tyre} \\ M_g &= 0.01 \cdot \tfrac{m}{2} \cdot r_{tyre} \\ M_x &= F_x \cdot r_{tyre} \end{aligned} \tag{2.8}$$

In this equation M_b is the braking moment, M_g the moment due to gravity, M_x the moment due to longitudinal forces on the tyre, i the brake ratio and F_b the braking force. This force is chosen as 3.5 times the gravity of the vehicle. The factor 3.5 is chosen, since the tyres then lock when braking. The brake ratio is chosen to be 0.6. This means that 60 percent of the braking is on the front tyres. These moments are then combined to calculate the resulting moment and the angular velocity of the tyres.

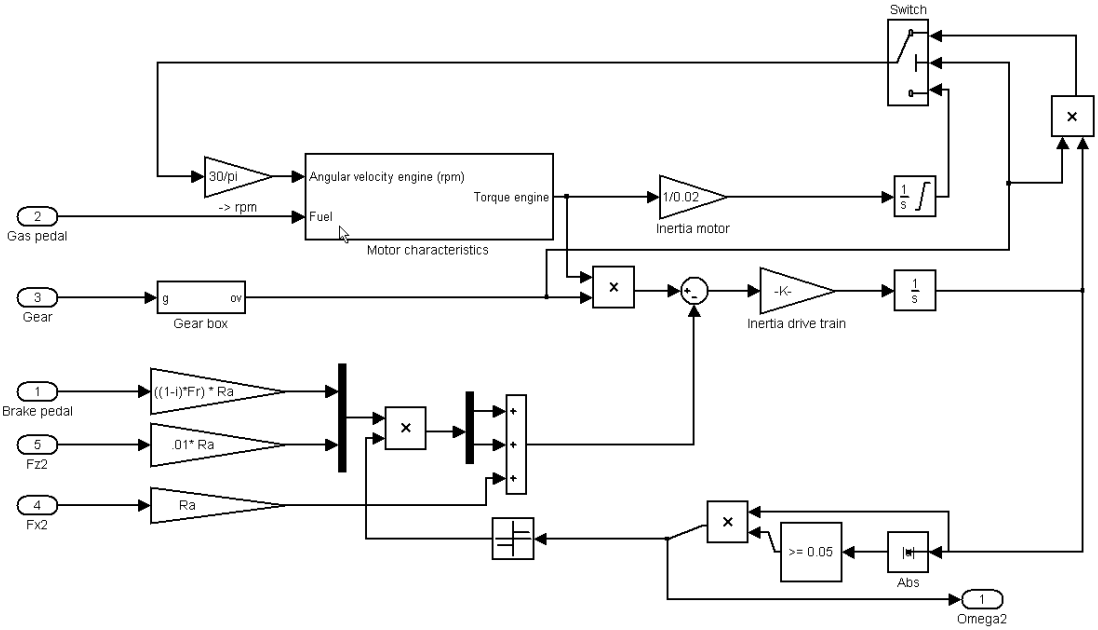

Figure 2.5: Rear drive train

2.5 Gearbox

The biggest challenge in implementing the gearbox was shifting up and down. The intention was to make it in such a way, that it is impossible to end up in a gear below neutral or above gear 6. Moreover it should shift only one gear at a time, even if the shift button is held for some time. To reach these goals, use has been made of Stateflow. The Stateflow chart, connected with the Simulink model and expanded, is shown in figure 2.6.

The Stateflow chart consists of basic blocks, the states Neutral to Sixth, connected to each other through transition channels. These channels are, in this case, activated through the trigger signals gear_up and gear_down. The trigger is set to rising, so when the signal changes from zero to one, the channel is triggered and the state changes. The action a state has to perform is written inside it. These states, or gears actually, only have a constant as output according to the gear. The output type is set to *during*. This type in combination with the third trigger, a periodic signal with a period-time equal to the simulation step time, assures that the output equals the gear at all time. For other output types one should consult the MATLAB help files [3]. The third trigger signal is also responsible for the initialization into neutral gear. Now the right gear is selected, the corresponding transmission ratio is selected using a look up table.

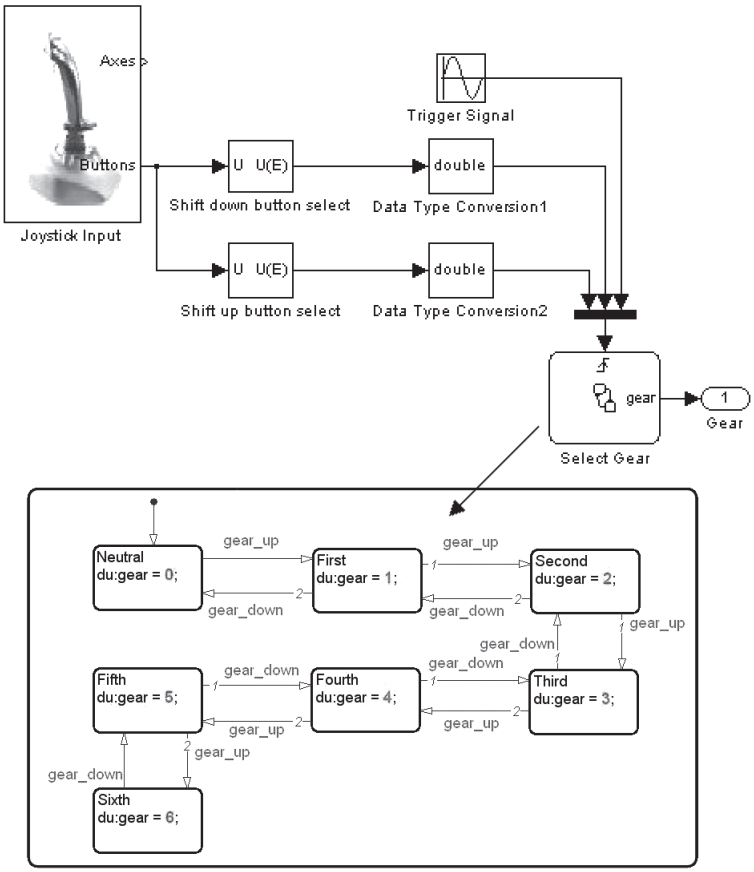

Figure 2.6: Stateflow chart of gear changing.

Chapter 3

Steering Wheel Input

For the realization of the input of the Race Simulator an input device is chosen. In order to realistically use the dynamical model of the car the signals of throttle, steering and braking needs to be continuous. Force feedback is also used in the simulator, see chapter 6, and therefore required. Another requirement is compatibility with MATLAB Simulink. Taking these requirements in consideration a Logitech® MOMO® Racing Force Feedback Wheel with gas and brake pedals is chosen, see figure 3.1.

The Joystick Input block in MATLAB Simulink provides a convenient interaction between the model and the virtual world associated with the Virtual Reality Toolbox block. The Joystick Input block uses axes and buttons. This block can be used as any other Simulink source block. The Joystick Input block can also be given an input for force-feedback.

Figure 3.1: Logitech® MOMO® Racing Force Feedback Wheel with gas and brake pedals.

Chapter 4

Visualization

One of the keys to a successful simulator is the visualization. Even if the dynamical model is incredibly accurate, if the visualization fails to show a natural movement, the simulator will be considered unrealistic. Therefore it is necessary to take a close look at the possibilities of visualization and in particular the VRML-language. This language can be scripted in a text editor, but there are programs to visualize your creation. For this project it was obvious to use 'V-realm Builder' as a platform to build virtual worlds, because this program is standard delivered with MATLAB/Simulink.

In this chapter the possibilities of the VRML-language will be discussed in combination with 'V-realm Builder'. The necessary parts for a realistic movement of the car are considered as well as the design of the car and its surroundings. For a more complete overview of the abilities of 'V-realm Builder' and the VRML language, take a look at [3].

4.1 VRML coordinates

VRML uses the world coordinate system, in which the y-axis is pointed upwards, the x-axis sidewards and the z-axis indicates the distance to the front of the screen. This is different from the normal coordinate system used by MATLAB, where the z-axis is pointed upwards as can be seen in figure 4.1. Rotations are defined using the right-hand rule, so when you look in a positive direction, the positive rotation is to the right.

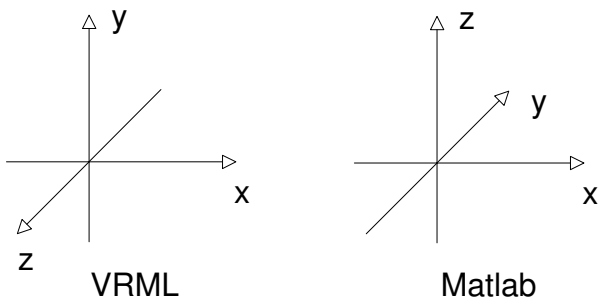

Figure 4.1: Coordinate systems in VRML and MATLAB

4.2 VRML basics

The purpose of the VRML language is to create 3d-objects and make them move around in a 3d-world. The language has too many features to discuss them all in this chapter, so there will be a short overview of the main features and the used options.

The main building block of VRML is a transform. A transform can represent an object and can be oriented and scaled in every desired way. The center of the transform can also be assigned in order to make an object rotate around the desired point. In order to make this transform an object, a shape can be inserted as a child of the transform. How these shapes are defined will be discussed in the next subsection. A transform can also have another transform as its child. In this case the orientation of the child is with respect to the parent, so moving the parent will also move the child. With this feature, objects can be created that consist of many different shapes.

4.2.1 Shapes

Creating shapes with VRML is done by defining coordinates in space and connecting them in order to create surfaces. Some basic shapes are already implemented, so they are easy to insert in your file. For example a box, a cone, a cylinder and a sphere are ready to use. Shapes can also be made by hand. Therefore you need to insert all points of your design and define the surfaces that form the shape together. This can be done in 'V-realm Builder', but that is pretty hard for complicated shapes because of the inability to enter coordinates. So when you want to make complicated shapes, a text editor is more easy to insert the coordinates and create the surfaces.

4.2.2 Extrusion

Another way to visualize an object is an extrusion. An extrusion consists of a basis and a path on which the basis is extruded. Only objects with a constant cross-section can be made with an extrusion. The objects made with extrusions can also be made by creating a shape, but an extrusion requires less effort and is therefore recommended if applicable.

4.2.3 Appearance

The appearance of an object can be adapted by changing its color, or by inserting a texture. Different colors can be selected as well as the shining color and the shininess. An object can also be made transparent, but this option must be enabled in the VRML viewer. Textures can be inserted in order to create realistic surfaces. These surfaces are stored as pictures and are reflected on an object. On large surfaces, textures are stretched and the pixels of the image are clearly visible. In order to avoid this a 'TextureTransform' can be inserted and some properties of the texture can be adjusted.

4.3 Dynamic VRML in Simulink

To work with a VRML world in Simulink, the 'Virtual Reality Toolbox' must be installed. In this toolbox the so-called 'VRsink' is included, which makes it possible to display VRML files. When the desired VRML-file is loaded in the block, the VRML world is shown and all parts with which the model is built are shown in a box. Here must be denoted what parameters are variable and

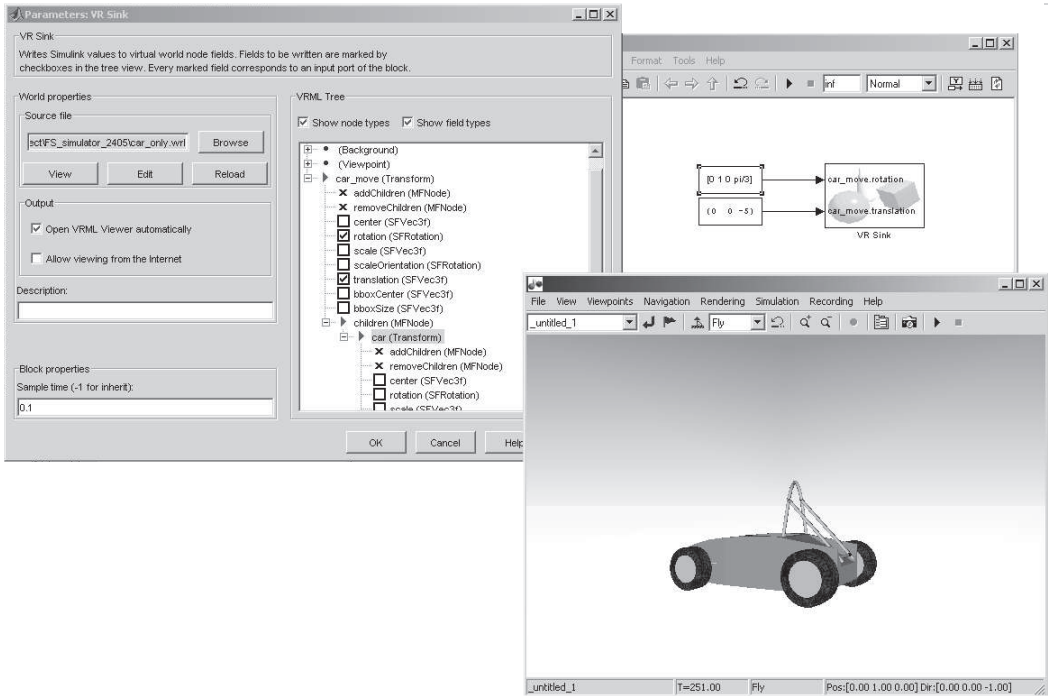

Figure 4.2: Impression of the 'VRsink' in Simulink

must be adjusted from Simulink. These variables then become an input in the 'VRsink'. The only constraint on being able to change a variable is that its parent must have a custom name. In figure 4.2 you can get an impression on how the 'VRsink' roughly works. The size of the input signals are multi dimensional. Translation inputs consist of the X-, Y-, and Z-values in the virtual world, while rotation inputs consist of the axis around which will be rotated and the rotation angle in radians. In figure 4.2, a screenshot of a car is shown with a translation of five meters in the Z-direction and a rotation of 60 degrees around the Y-axis.

4.3.1 Coordinate transformation

Because the output of the dynamical model is not expressed in the main directions of the world coordinate system, a coordinate transformation is required in order to make the correct movement of the car visible. How to do this is described in [7] and can be implemented in Simulink with standard building blocks. The angular velocities of the car must be integrated to obtain rotations. These rotations should be multiplied with the rotation matrix to obtain rotations in the right coordinate system. The translation of the car can then be obtained by multiplying the output of the rotation matrix with the velocities and integrating this signal. When rotations around multiple axes are introduced, tools from the 'Virtual Reality Toolbox' are required, for example the Rotation Matrix to VRML Rotation block. In figure 4.3 is shown how this can be implemented.

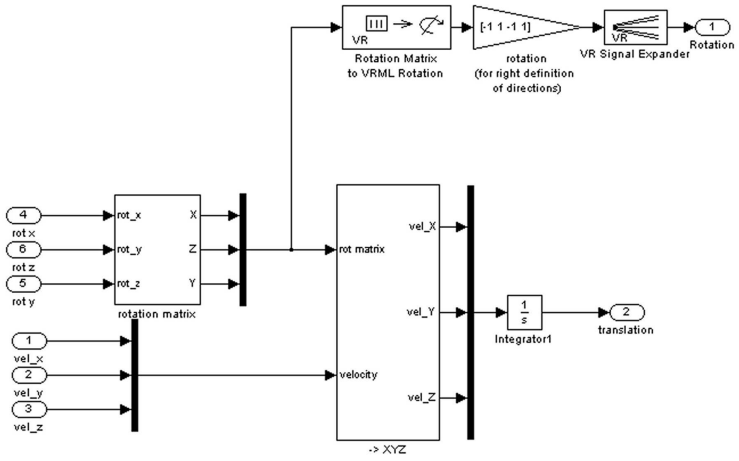

Figure 4.3: Implementation of coordinate transformation with rotation around multiple axes

4.4 Car design

4.4.1 Export from Unigraphics

In order to make a realistic simulator, the virtual car has to represent the real racing car , figure 4.4, as good as possible. Because there exists a complete 3D model of the car in Unigraphics, figure 4.5, the most realistic car is obtained by exporting this file to VRML. However, the level of detail of this model caused the resulting VRML file to be larger than 15MB. A file of this magnitude would not be favorable for the performance of the final simulator, because the car has to move around. A solution would be to adapt the Unigraphics file and remove all unnecessary details, but even with this adapted file, the VRML file was too large. Therefore this method is not used to design the car in the final simulator.

4.4.2 Simple car design

The main idea of creating a car by hand is making one parent and define all different parts as its children in order to be able to make the whole car move as one. Not all parts of the car are included in the design, simply because it takes a lot of time to make very detailed objects by hand. The car consists of five main parts, chassis, wheels, suspension, roll bar and the steering wheel. The chassis and the steering wheel are designed manually with the use of shapes and the roll bar is created using an extrusion. The wheels and suspension are created using standard cylinders in different formats. All parts are given a color, textures are only added to the tyres to visualize the rotation of the tyres. The result of this design is shown in figure 4.6.

4.5 Circuit design

The circuit should be one of the circuits on which the Formula Student team will be racing. One of these circuits is Bruntingthorpe in England and this circuit will be the racetrack in the final

Figure 4.4: Car at the official presentation

Figure 4.5: Car design in Unigraphics

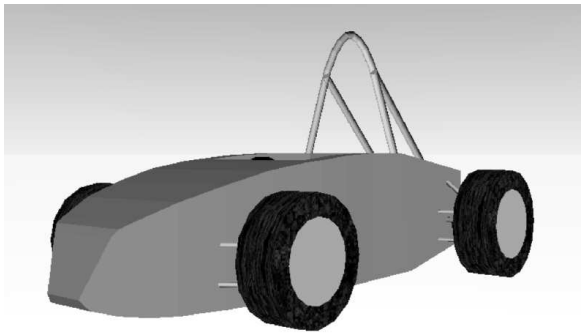

Figure 4.6: Simple car design in VRML

simulator.

Because a circuit is nothing more than an extruded line, it is obvious to choose an extrusion as the tool for designing it. One problem is to determine the center line of the track over which the line should be extruded. The only available data of the track is a GPS plot derived map, from [6]. Although it is an impression of the racing line of the Honda-driver, this picture will be used as an impression of the center line of the racetrack. A short overview of the used algorithm is following, but the complete MATLAB-script is included in appendix A.

The picture of the circuit is loaded into MATLAB and is converted to a line with a width of one pixel. MATLAB can then fit a polynomial through these data points, this results in a good fit of the circuit. This polynomial can then be divided into a desired amount of points on the track, and the final step would be to transform these points into coordinates of the virtual world. These coordinates must be inserted in the path of the VRML extrusion and the width of the circuit can also be set. The result of this approach is put together with the available GPS-map in figure 4.7.

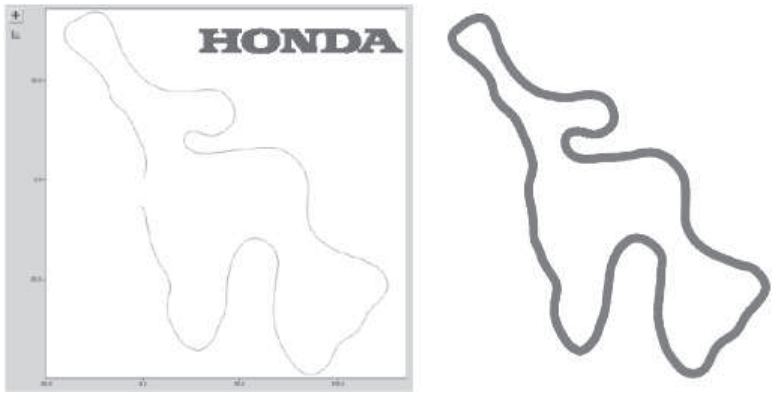

Figure 4.7: Bruntingthorpe circuit, left: real track, right: VRML design.

4.6 Graphical User Interface

An essential part for the experience of a simulator is the graphical user interface. While driving the car it is nice to know how fast you are driving or how many revolutions the motor is making, and the laptime is also an important given on a race track. To make the simulator more appealing for the audience, these signals have to be shown on the screen. In the next subsections the individual parts of the GUI are discussed and the final GUI is shown in figure 4.8.

4.6.1 Gauges

To display the essential parameters of the car, the gauges blockset in Simulink offers a lot of usable gauges. These gauges are pretty realistic and can easily be connected to the model, but because it is more appealing to have all visualization parts on one screen this option is not used.

Visualizing a speedometer or a revolution counter in VRML can be done by creating a round face and giving it a texture with the appearance of the gauge. The needle is a rectangular shape with its center assigned at the end, and placed in the middle of the gauge. To make it display the right output, the angle of the the needle must be adapted according to the input. The conversion between for example the velocity of the car and the angle of the needle must be calibrated according to the appearance of the gauge. In figure 4.8 the final gauges are shown.

4.6.2 Displays

Displaying text in VRML is done by inserting text as a child of a transform. However, text can not be denoted as a changing variable in Simulink and therefore it can not be used to display the lap time. One of the other options for the lap time display is an analogous clock that can be created the same way as the speedometer, but a digital clock is preferred, so another way to display dynamic text must be found.

One of the features in VRML that is not discussed yet, is the switch. A switch can be given children and with the 'SwitchChoice' option, a choice can be made which child should be visible. A digital clock can be made with this feature by inserting different text transforms as children. To make a digital clock with four digits, four of these switches are needed. In order to be able to display time in this manner, time in Simulink must be divided into minutes, ten seconds, seconds and tenth of seconds. In the 'VRsink' must be denoted that the switch choice is variable and the time digits form the input that changes the visible child of the switch. This results in dynamic text. For the gear display only one digit is needed.

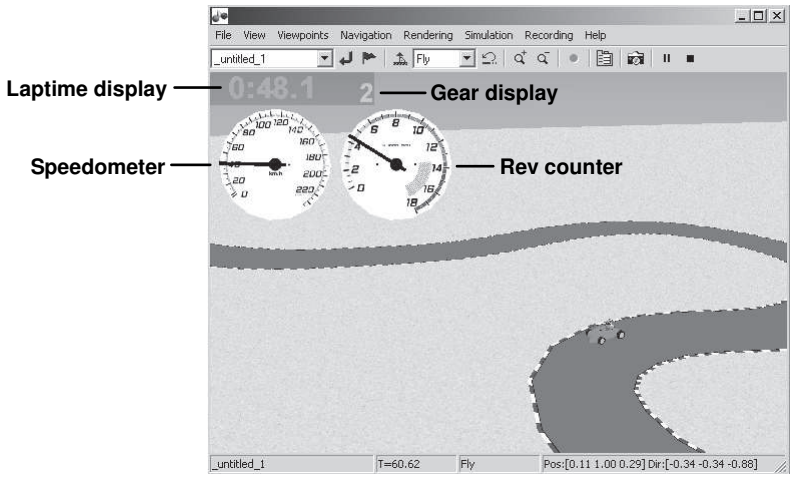

Figure 4.8: Final Graphical User Interface.

4.7 Viewpoints

Another feature to make the simulator more lively is changing viewpoints. A viewpoint can be inserted to change the point of view in a VRML world rapidly. The orientation of a viewpoint can

easily be set in the 'V-realm builder' and there is no limit to the amount of viewpoints. Changing from one viewpoint to another during a simulation can easily be done by clicking PageUp or PageDown, but that is not an option while one is racing with the steering wheel. Therefore another option to change view is necessary for the simulator. A viewpoint has the option 'set bind' that can be true or false depending on the activity of the viewpoint. This option can be adjusted from Simulink if you denote it in the 'VRsink', but while simulating an error occurs. That is why another method is implemented to change viewpoints during simulation.

The solution is found in creating one viewpoint as a child of a transform, which is also the child of a transform. Translation of the first parent transform indicates the position of the viewpoint and the translation and rotation of the second parent indicate the movement of the viewpoint. Translation and rotation of these transforms are denoted as variable in the 'VRsink', so they are adjustable from Simulink. This option has the advantage that the point of view can be changed without adapting the VRML file. To be able to change from one point of view to another, a 'multiport switch' is used in Simulink, figure 4.9. With this block, a choice can be made between different inputs, so when the translation and rotation of a viewpoint are given as the inputs, switching viewpoints is possible. Another advantage of this method is the possibility to make a viewpoint fly around in the virtual world. When a transfer function is inserted between the output of the viewpoint switch and the 'VRsink', the impression of flying around is raised.

4.7.1 Implemented viewpoints

In the final simulator three viewpoints are implemented, but this can be adjusted. Figure 4.9 shows an impression of the different viewpoints. The first viewpoint is the 'inside car view' and represents the view of the driver of the car. This view is positioned just above the middle of the car and the movement of the viewpoint is exactly the same as the movement of the car.

The next implemented view will be mostly used while racing and is called the 'behind car view'. Because the whole car is in front of the viewpoint, there should be a good impression of the whole behavior of the car. The only problem with this view is creating a realistic movement, because when the movement of the car is given directly translated to this viewpoint, the view is always right behind the car and the race impression is very poor. Therefore some kind of difference between the rotation of the car and the viewpoint must be introduced. This can be done by inserting a delay between the two rotations, what results in a more dynamic view while cornering. The best way of making this viewpoint rotate is by letting it face in the same direction as the car's movement. This method makes it better visible when the car is drifting around a corner and is implemented by adding the vehicle side slip angle to the car's rotation.

Another viewpoint is the 'view from above'. This view gives a good overview of the race, because it is placed at a great distance above the track. The movement of this viewpoint consists of only the translation of the car and to get a more dynamic impression, a delay is inserted.

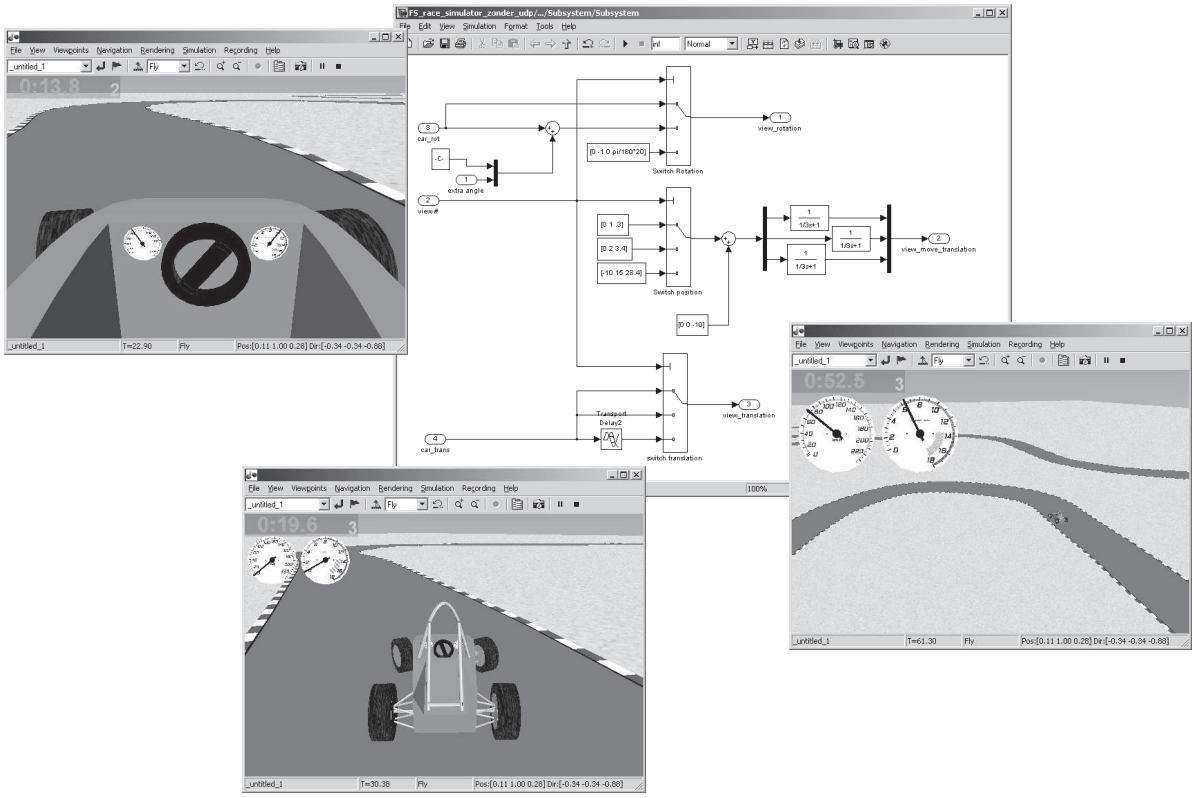

Figure 4.9: Implementation of viewpoint switching and an impression of the viewpoints.

Chapter 5

Sound

For a realistic race experience sound is a very important aspect. Two types of sound are implemented in the simulator. The first is engine sound, the second the sound of the slipping wheels.

5.1 Engine

Especially the sound of the engine is important. If it is realistic enough this sound will help you determine when it is time to change gears.

Since the frequency and volume of the sound should vary continuously it is almost impossible to use recorded sound samples. The sound has to be generated online. The signal that will be sent to the sound card should be some sort of wave form. The basic frequency of this wave should be the same as the engine speed.

To realize this, the engine speed is integrated and sent through a sine function. This generates a sine wave having amplitude one and the same frequency as the engine. Since engine noise consists of more than one frequency also a signal of double, half and quarter frequency is made. These signals are added up using different gains. Real engine sound is not made up of just sine functions. To make up for this, the sound is made noisier using clipping of the signal. This makes the sound more realistic. In figure 5.1 the implementation of this can be seen.

The volume of the sound should depend on the position of the accelerator. This is done in a

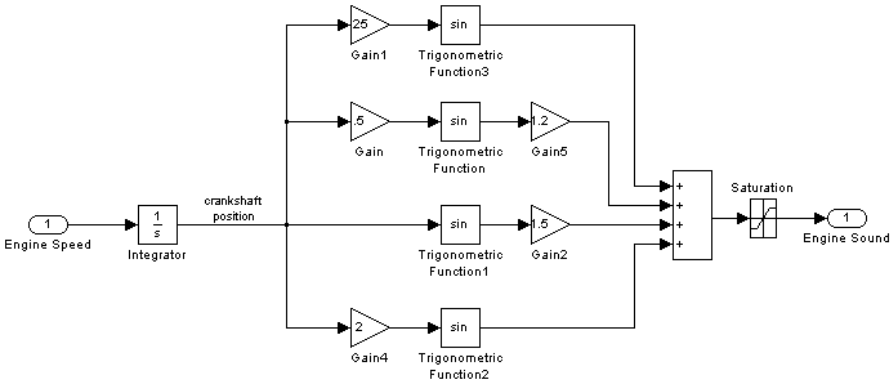

Figure 5.1: Implementation of the sound generator.

simple linear way. When the accelerator is completely floored, the sound is loudest. When the accelerator is not pushed at all, the volume of the sound is set to 30% of the maximum value.

5.2 Wheel slip

From a sound sample of a slipping wheel a sound vector is obtained. This vector is loaded in the MATLAB workspace and used as an input in Simulink. The same vector that plays for about a second is played repetitively. This gives a continuous wheel slip sound. To get the slip sound to react to the simulation the volume of the sound depends on the amount of slip the tyres have. The σ^* values of the front and rear tyre represent the amount of combined slip. The values for the front and rear tyre are added up. The result is put through a gain and is used to modify the volume of the sound. This gives no sound at all when there is no slip. When the slip level increases the volume of the slip sound increases also.

Chapter 6

Force Feedback

To get an even more realistic race experience force feedback has been implemented in the simulator. This force feedback is actually a moment on the steering wheel. In our simulator two kinds of force feedback are applied. First the self aligning moment is explained, second the force from driving off the track is elaborated.

6.1 Self aligning moment

A possible way to describe the aligning moment is by using the so called pneumatic trail and side force F_y. For our simulator this is not a good option. When this is applied, the steering wheel can give rather high forces when driving in an almost straight line. This is caused by the way the slip conditions are calculated. This is not very realistic.

Therefore another method is used. Instead of the real side slip angle and real side force, the speed and steer angle are used to give an estimation of the self aligning moment. Though the physical foundation of this approach is not very solid, this does give a larger force when the side force increases.

6.2 Track tracking

In order to add some features to the simulator, it is nice to know if the car is on the track or not. When this information is available, it is possible to make the steering wheel vibrate as soon as the car leaves the track to get a more realistic race impression.

To achieve this goal, the location of the car and a map of the track must be available. The location of the car in the virtual world is an input of the visualization in Simulink, so this parameter is already known. A map of the track must be made available in the form of coordinates at which the car is on the track.

The desired structure of the map is a matrix with its axes representing the main directions in the virtual world. Coordinates at which the car is on the track, must have the value zero, whereas the other coordinates must have the value one. When this matrix is available, a lookup table in Simulink is usable to compare the position of the car with the right value in the matrix. How this matrix is created, is described in A. The outcome of the lookup table is a one, as soon as the car leaves the track. To make sure that the vibrations occur when the wheels of the car are off the track, the width of the track in the tracking matrix is smaller compared to the width of the track.

6.2.1 Lap time

Another feature that can be introduced now, is recording the lap time. In the map of the track can be indicated where the start of the track is located. This is done by increasing the value of the matrix at these locations to three. In Simulink can be implemented that the lap time must be set to zero as soon as the output of the lookup table has the value three.

Chapter 7

Making a working simulator

Now that all the dynamics, the joystick-inputs and the visualizations are implemented, the only thing left is to combine all these parts into one working total. The mutual connections are made in a straight forward way as can be seen in figure 7.1. The only problem left is to run the model in real-time.

7.1 Real Time

Standard Simulink will run any model as fast as the computer speed allows. For a driving simulator it is important that the simulation time and actual time are the same. This can be done by artificially slowing down the simulation if the model runs faster than real time. There are a few options for doing this. The easiest way to do this, is to adjust the sampling time such that the model will run in realtime. An advantage of this approach is that the model is executed in the most accurate way the computer can handle. A drawback however is that it is never exactly realtime and the sampling time has to be adjusted and tuned every time the model is altered or another computer with another clock speed is used. Another way to get the computer to run in realtime is to use the Real Time Workshop. How this can be done can be found in [3]. The disadvantage is that not all Simulink blocks are compatible with the Real Time Workshop, e.g. the block for playing sound and the joystick input block don't work in combination with the Real

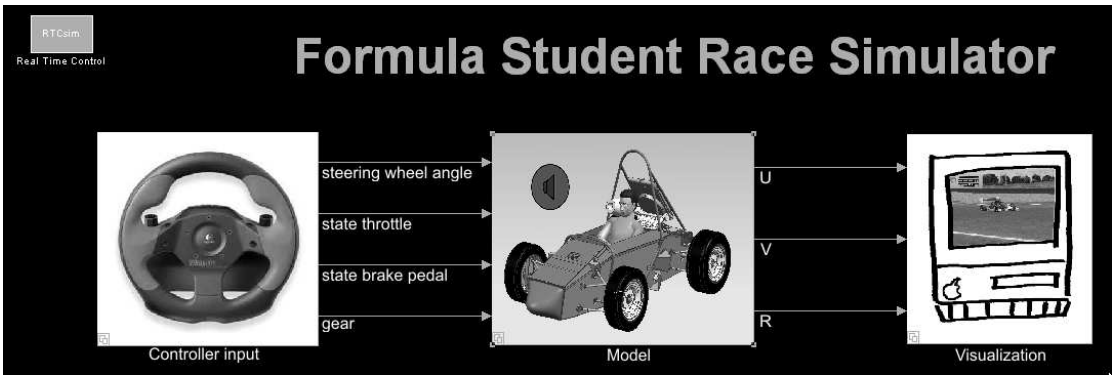

Figure 7.1: Connection of all separate parts.

Time Workshop. After a search on the internet a third option turned up. A specialized toolbox was developed by Werner Zimmerman [8]. This toolbox contains the RTCsim block. This block compares the simulation time to the system time and adapts, on the basis of that, the sampling time. So it is actually an automated version of option one. This approach makes it possible to run the simulation in real-time, however it is only possible to slow down the simulation.

7.2 Dividing workload

It turned out that after connecting the entire model it was not possible to run this model, including visualization and sound on one of the test laptops in real-time using the approach stated above. The processors of these laptops are simply not fast enough. There are a few ways to cope with this problem. The first one is to buy new computers with better performance, but this option is not applicable in this case. The second option is to implement the models in a more efficient way, such that less computer performance is needed. This consideration is taken into account; the models are implemented as efficient as possible, but the processors are still not able to run the program in real-time. Therefore the third option is chosen for: distribute the workload over different computers. In this case it means that the model is cut into parts and than divided over separate computers. It appeared that there were two parts of our design which required a lot of process-time: the visualization and the sound processing. Therefore the model is split into three parts. The first part deals with the joystick-input, the car dynamics and the coordinate transformation for the visualization. This part subsequently sends the concerning parameters to a computer which deals with the sound and an other computer which deals with the visualization. Because of this it is possible to run the dynamic model in combination with the sound and visualization in real-time. An additional advantage is that al the three separate parts can be improved and extended, because after the distribution the models do not need full processor time.

7.2.1 UDP

In this case UDP has been used to communicate between the separated models. UDP stands for user datagram protocol. There are also other methods to communicate as can be read in appendix B.2, but here is chosen for UDP because it gives fast communication and it is easy to implement. One drawback of UDP however is that you will never be sure whether a message arrives at the other computer or not. In this case this is not a very important issue because when a message accidentally gets lost, it will not have a major impact on the visualization or sound.

The implementation of UDP can be seen in figure 7.2. At first, all signals have to be packed into one binary signal, using the Pack block from the Simulink xPc-Target toolbox. This block requires that you indicate the type of every signal. After this the packed signal can be sent to another IP adres using the Send block. The other pc can now receive the packed data using the receive block in combination with the right IP address. It is also necessary to indicate the size of the received signal in number of bytes. There are two output ports at the receive block. The first one contains the packed signal which can be unpacked using the unpack block. This one requires the same signal information as the pack block. The second one is a flag indicating whether any new data has been received. This port outputs a value of 1 for the sample when there is new data and a 0 otherwise. If no new data is received, the receive block uses the previous output.

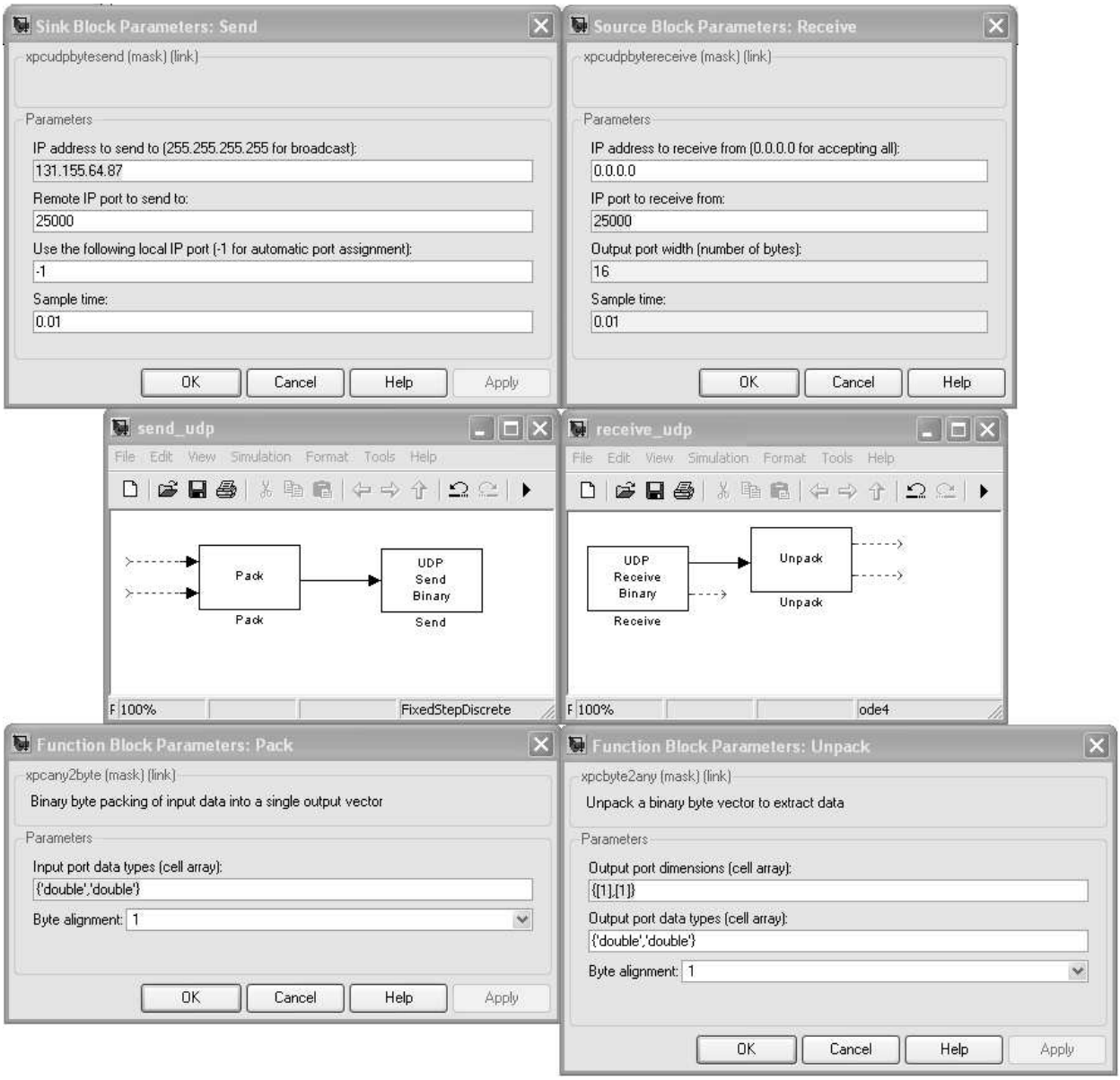

Figure 7.2: Implementation of communication using UDP.

Chapter 8

Conclusions and Recommendations Part I

To persuade potential sponsors a real-time simulator has been developed. With this simulator it is possible to drive a three dimensional formula student car in a three dimensional world using external steering wheel input. Moreover a rather realistic car model is implemented using MATLAB Simulink, to experience some real life driving behavior.

The dynamical model of the car is based on the bicycle model, which defines a mass and its center of gravity. The forces are passed on to the car through the tyres. The tyres are modelled in such a way that they include combined slip, power oversteer, limited corner speed, wheel spin and wheel lock. The drive train is modelled using an engine characteristic and instantaneous gear changes.

The visualization is implemented using the Simulink Virtual Reality Toolbox, which uses VRML. The visualization consist of three major parts, the car, the track and the graphical user interfaces. The car had to be redrawn by hand, because the existing Unigraphics models were too large to visualize. It follows the design of the FSRTE02. This is done as realistic as possible. The track is based on the existing race track Bruntingthorpe in England. There are namely the upcoming races. Finally GUI's are added to give the driver some information about speed, engine speed, gear and lap time.

To increase the driving experience even further, sound and force feedback are added. Sound is created using an engine and slip dependent signal. The force feedback also consists of two signals, an aligning moment and a disturbance signal which becomes active when the car leaves the track.

The last problem to be dealt with is real-time simulation. If the whole model would be implemented on a single computer, the model would not be able to run in real-time, because the calculation time became too large. Therefore the model is split up into three parts and distributed over separate computers. There is one master computer, which calculates the model and then sends the concerning parameters for sound and visualization to the other two computers. This data is sent over the network using UDP. With this adjustment the models can run even faster than real-time, but that too is not desired. Therefore a special Simulink block was added which dynamically adjusts the simulation step time in such a way that the simulation time equals the actual time.

The result of all these parts together is a sophisticated real-time simulator of the FSRTE02 which can be used for promotional purposes. This one is realized with the use of standard MAT-

LAB/simulink and some additional toolboxes. There is however always room for improvement, such as:

- During the development of this simulator not all parameters were known. Thus to increase realism the real car parameters should be implemented.
- The current dynamic model of the car could be extended e.g. with a four wheel model with a more advanced weight distribution. If the model then becomes too large, it could be distributed over multiple processors.
- The aligning moment for the force feedback is now not very realistic implemented. For a better implementation see Pacejka [5].
- When the car leaves the track, it should experience more resistance and less grip.
- Collisions with objects in the vrml world should be taken into account.
- The visualization of the three dimensional world and car could be improved.
- The simulator should keep track of the fastest lap times.
- More racetracks could be added.
- For better promotion results, the simulator could be installed in an old formula student race car with a video screen in front of it.

Part II

Lap Time Simulation

Chapter 9

General Model Description

Since the design of next years car has not even begun, very limited information about it is available. Therefore only a very simple vehicle model can be used to describe it. In this case the vehicle consists of a point mass and one tyre having limited grip. The grip this tyre has, only depends on the mass it has to support which is constant in this model.

While driving along the track, a perfect driver is assumed. At all points the vehicle will undergo optimal acceleration. This is achieved by either having maximum combined lateral and longitudinal acceleration, or using the full power of the engine.

The gearbox to be modelled is a perfectly continuous gearbox. Therefore the engine can always provide maximum power and no shifts are needed. Air drag will also be modelled. This will limit the top speed and the acceleration once the vehicle is at speed. This air drag does have a positive influence on the deceleration while braking.

After reading this, one may think that all these assumptions are rather optimistic and the lap times such a simulation will provide can never be reached in practise. And indeed, all these assumptions are optimistic. For the process of making an evaluation of the best of the two alternatives only the differences between the two cars have to be reliable. The lap times this simulation will give can be considered as a lower limit on the actual time that can be achieved.

Chapter 10

Implementation

To get lap times from this point mass vehicle model some calculations are needed. As a first step, the model explained in the previous chapter has to be converted to a mathematical model. The basis for this model is Newton's second law. At all points on the track the force in lateral direction is calculated. From this force the possible longitudinal acceleration can be calculated. Using this acceleration the speed at the next point and the step in time can be calculated.

The first limiting factor on the acceleration is the grip of the tyres. This grip is assumed to be equal in all directions and can be characterized by a friction coëfficiënt μ. This μ depends on the force it has to support. Since the car has four tyres every tyre has to support a quarter of the vehicle weight. The formula provided in [1] is used to get:

$$\mu = 1.74 - 0.000128 \cdot \frac{9.81 \cdot m}{4} \tag{10.1}$$

The available grip has to be divided in a lateral and a longitudinal grip force. The lateral force can be calculated using:

$$F_y = mu^2 R^{-1} \tag{10.2}$$

For this the curvature R^{-1} has to be known at all calculation points. This curvature follows from the circuit to be driven. This curvature is zero when driving in a straight line, constant when driving on a circle and varying when driving on a general circuit.

Now that the lateral force is known the maximum longitudinal force that can be transmitted by the tyres can also be calculated. When braking, all tyres can be used and the longitudinal force will become:

$$F_{x,grip} = \sqrt{(\mu mg)^2 - F_y^2} \tag{10.3}$$

When accelerating only the rear tyres can be used. Taking this into account an estimated 65% of $F_{x,grip}$ can be used for accelerating.

The cars to be considered have only a finite amount of power available. Therefore the grip available might not be put to full use. The maximum longitudinal force following from the engine power equals:

$$F_{x,engine} = \frac{P}{u} \tag{10.4}$$

The acceleration the car will undergo only depends on the smallest of $F_{x,grip}$ and $F_{x,engine}$. Next to this force another force acts on the vehicle. This is the air drag. This force can be calculated using:

$$F_{x,drag} = \frac{1}{2} C_D \rho A u^2 \tag{10.5}$$

The resulting force now becomes the minimum of $F_{x,grip}$ and $F_{x,engine}$ minus $F_{x,drag}$.

Now that the longitudinal force is known the acceleration and the speed at the next step can be calculated. For the updating of the speed a constant acceleration is assumed.

$$\begin{aligned} a &= \frac{\min(F_{x,grip}, F_{x,engine}) - F_{x,drag}}{m} \\ u_{k+1} &= u_k + a \cdot \Delta t \\ \Delta s = \int_{\Delta t} u dt = \int_{\Delta t} u_k + a \cdot t dt &= u_k \Delta t + \tfrac{1}{2} a \Delta t^2 \rightarrow \Delta t = \frac{-u_k + \sqrt{u_k^2 + 2a\Delta s}}{a} \\ u_{k+1} = u_k + a \cdot \frac{-u_k + \sqrt{u_k^2 + 2a\Delta s}}{a} &= \sqrt{u_k^2 + 2a\Delta s} \end{aligned} \tag{10.6}$$

In this equation Δs is the distance between two points on the track. When the car at all points undergoes the acceleration according to equation 10.6 it might come across a corner at a speed too high for the curvature. If this happens the car will not be able to corner fast enough. Therefore the speed of the car has to be lowered before entering such a corner. The problem is: to know when to start braking.

A solution for this problem is to first run a similar simulation in reverse. The engine power for this reverse simulation is no longer relevant and the air drag now has the same direction as the speed of the car. When, in this reverse simulation the car comes across a corner at a speed that is too high the speed can be modelled to decrease to the maximum corner speed instantaneously. From this simulation a maximum speed profile along the track is determined. When the car brakes maximally at this speed it will exactly be at the maximum corner speed when it reaches the next corner. If the speed of the car is above this maximum speed line the car will not be able to brake hard enough to make the next corner. Therefore the speed at a simulation step should be the minimum of the speed following from 10.6 and the speed from the reverse simulation.

To summarize this chapter the calculation steps are outlined here:

1. Calculate the lateral force using the curvature and speed (10.2)
2. Calculate how much grip there is left in longitudinal direction (10.3)
3. Calculate the force the engine can deliver (10.4)
4. Calculate the air drag (10.5)
5. Calculate the resulting longitudinal force
6. Update the speed (10.6)
7. Check if the speed is not too high according to the reverse simulation
8. Update the time and position and continue at step 1

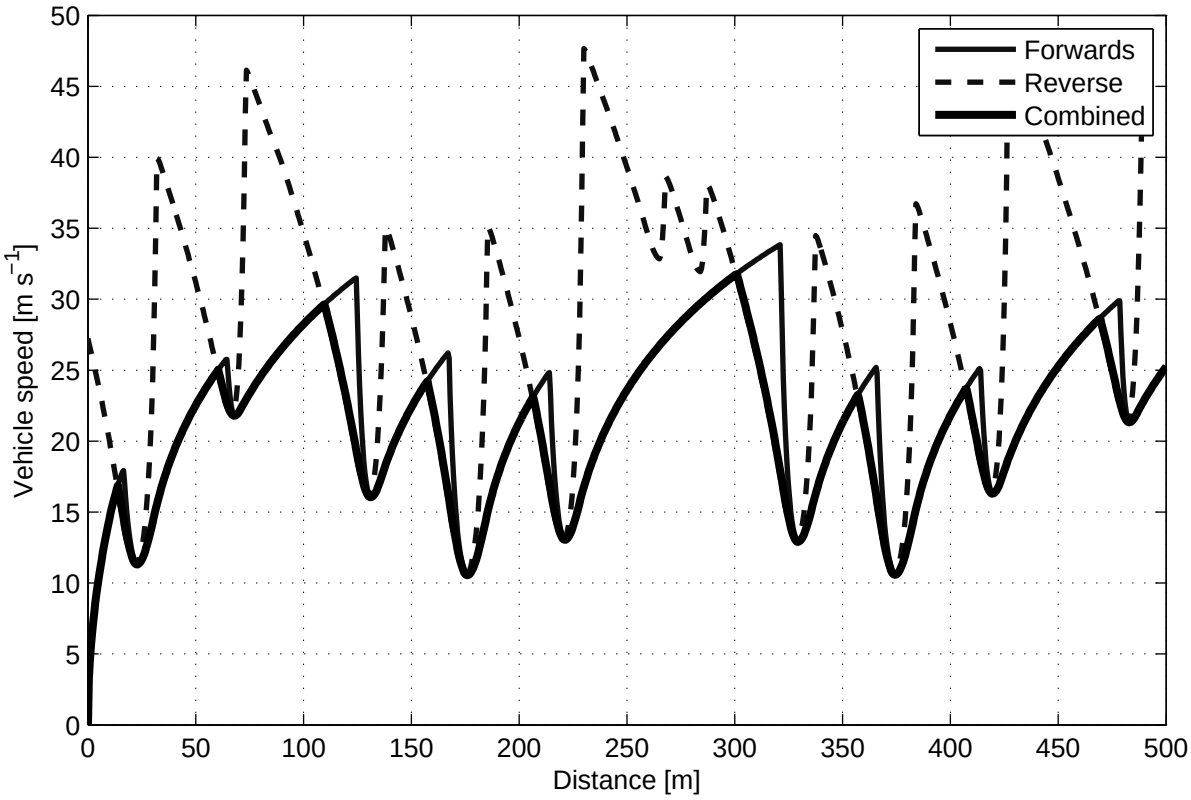

Figure 10.1: Forward, reverse and combined vehicle speed

The way the speed is determined is shown in figure 10.1; taking the minimum of the forward and backward simulation as mentioned in step 7. After this simulation the time, speed and acceleration are known at every position on the track. In appendix C the MATLAB implementation is shown. The results for the cars to be considered will be presented in the next chapter.

Chapter 11

Results and Comparison

In this chapter the model described in chapter 10 is used to perform laptime simulations. The two designs already presented in the introduction will be compared with each other and with two other interesting Formula Student designs. These other designs are the DUT04, a very light car built by Delft University, and the winner of last year by Toronto University.

To be able to perform a simulation, first the curvature of the track must be available. For the acceleration event this is easy, the curvature is zero everywhere and the track length is 75 m according to the regulations [4].

The skid-pad consists, according to the same regulations, of two circles in a figure eight pattern having their centers separated by 18.25 m. These circles have a diameter of 15.25 m. From these two diameters an effective diameter is estimated. This effective diameter is chosen to be 18 m, giving a curvature of $R^{-1} = \frac{2}{18}$.

For the endurance race it is more difficult to determine the curvature. The curvature for this event is anything but constant. Therefore a computer program analyzing the curvature of the track has been written. This program gives the curvature of the Formula Student race track as a function of the distance. The details can be found in appendix A.

Apart from the track also some car parameters have to be known to perform a simulation. To determine the air drag, a drag coëfficiënt and frontal surface have to be given as input. The drag coëfficiënt is estimated to be 0.35 and the frontal surface is estimated to be 1 m^2. These parameters will be taken equal for all cars. Two very important parameters are the mass of the vehicle including its driver and the power of its engine. These parameters will be different for each vehicle. The driver's weight including driving clothes and helmet is held equal for all simulations and is chosen to be 80 kg.

When all these parameters are inserted in the MATLAB function as can be found in appendix C, the times of table 11.1 result. For the two Eindhoven designs, the results are also graphically depicted in figures 11.1 and 11.2. In these figures, the time, acceleration, track curvature and vehicle speed are presented.

Table 11.1: Times in seconds for some cars on different formula student driving events.

	carbon chassis Aprilia engine 205 kg, 41 kW	Aluminium chassis Suzuki engine 295 kg, 65 kW	DUT04 205 kg, 30 kW	Toronto 293 kg, 60 kW
Acceleration	3.9325	3.9013	4.1113	3.9290
Skid-pad	21.1548	21.3332	21.1548	21.3291
Endurance	23.6380	23.7448	23.9943	23.7722

From the results in table 11.1 it is obvious that both designs from Eindhoven University of Technology are very competitive and even faster than the other cars in this simulation. The powerful design using the Suzuki engine is the fastest in the acceleration event. The lightweight car using the Aprilia engine is the fastest in the skid-pad event and the endurance race. The differences are however very small. Therefore it seems sensible, to also look at some other factors when making a decision.

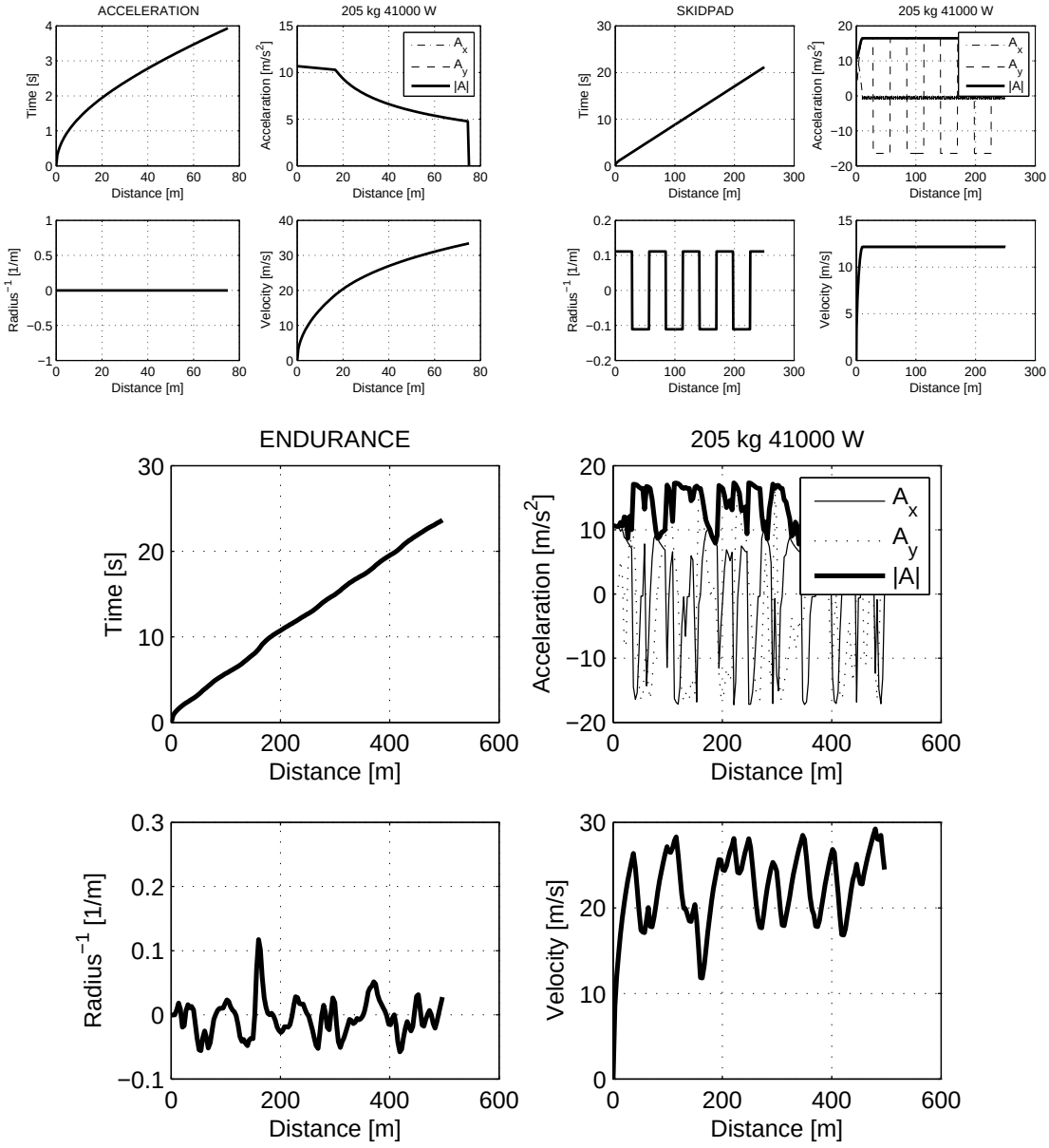

Figure 11.1: Graphical representation of the simulation on different tracks using a 41 kW 205 kg vehicle.

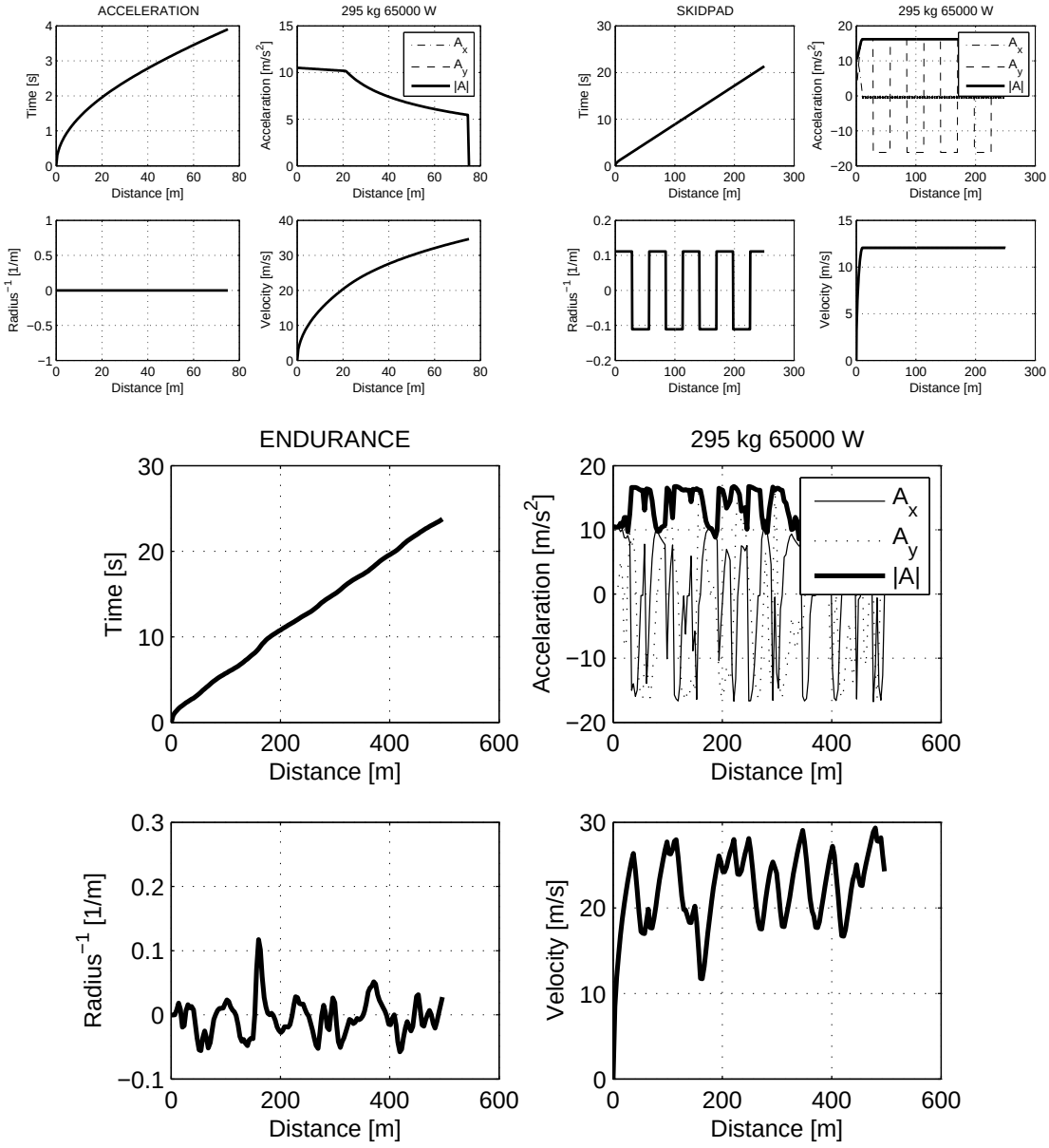

Figure 11.2: Graphical representation of the simulation on different tracks using a 65 kW 295 kg vehicle.

Chapter 12

Sensitivity Analysis

To find out how sensitive the results from table 11.1 are to small changes in parameters, the same calculations have been repeated using somewhat changed parameters. The parameters for the standard car are 250 kg weight including its driver, 50 kW engine power and a C_w value of 0.35.

The parameter changes are chosen 10% of the standard values. All changes are made such that the effect will be an increase in lap time. This way four simulations are carried out. One using the standard values and three using simulations with one of the parameters changed 10%.

From table 12.1 it can be seen that the 10% changes have no rigorous effects. Especially the changes due to the increased air drag are very small.

The change in power has somewhat more effect on the times. Surprisingly this change has no effect at all on the skid-pad time. This can be explained by investigating figures 11.1 and 11.2. During longitudinal acceleration the engine power is never the limiting factor, so an increase in power won't make the car any faster. The time on the 75 m acceleration is the slowest of all, it is slightly slower than the time resulting from an increase in weight.

The change in weight gives the most changes. All times increase. The time on the endurance race and skid-pad is the slowest of all. Therefore it is clear, that one should be careful adding weight to the car.

Table 12.1: Times in seconds for cars having changing parameters on different formula student driving events.

	Standard car 250 kg, 50 kW, 0.35	Increased weight 275 kg	Decreased power 45 kW	Increased air drag 0.385
Acceleration	3.9343	3.9816	3.9850	3.9385
Skid-pad	21.2432	21.2930	21.2432	21.2438
Endurance	23.7088	23.8317	23.7994	23.7179

Chapter 13

Validation of the Model

To check how close the used model comes to the real situation, the outcome of the simulations has been compared with data collected by Honda using two real Formula Student cars by Leeds University and Oxford Brookes. This data comes from [6]. The data gives the speed of the car along the track. Since the simulation can give this output as well this can be compared. The data from Honda is shown in figure 13.1.

A comparison of figure 13.1 with the speeds during the endurance race in figures 11.1 and 11.2 shows that there is some similarity between the measurements and the simulation. But the similarity might be not as striking as one might expect since the simulation is done using a similar car on the same track. The speed of the car in the simulation is quite a bit faster than the speed in the actual lap. This difference can be explained by considering that the utopian driver used in the simulation might be quite a bit better than the one actually driving the car around the track. Moreover, in real life, even with a perfect driver, you can still not make as much use of the tyres as is done in the simulation. Factors as load transfer have not been accounted for in the simulation. Also the use of a real gearbox instead of the ideal continuous transmission in the simulation will give some ground for the reduced speed.

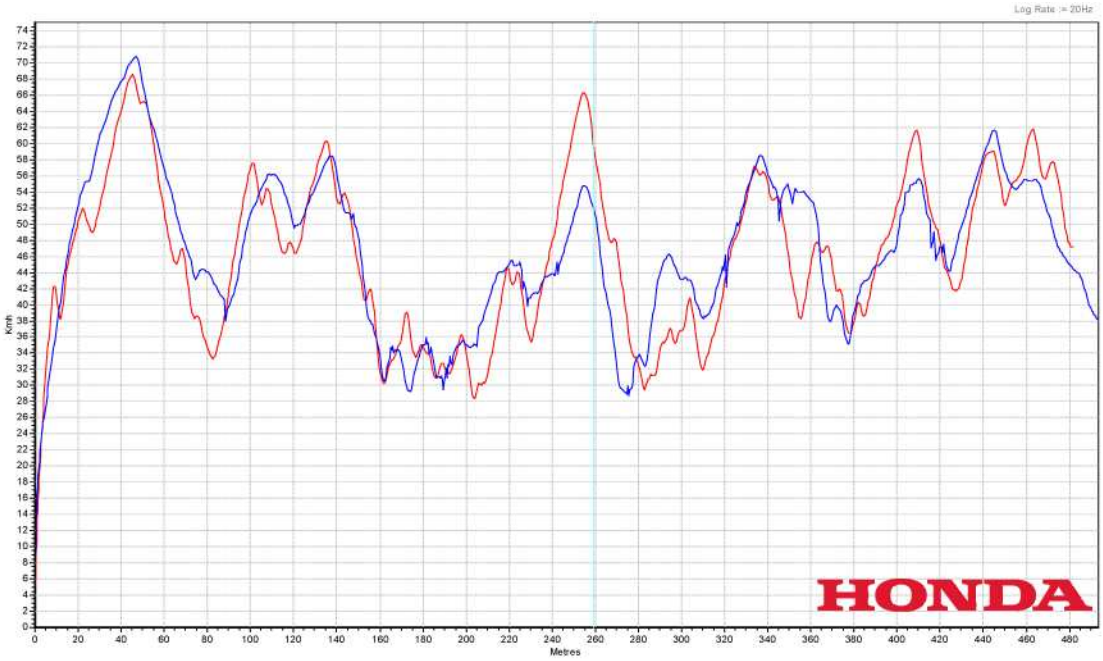

Figure 13.1: Distance traveled in meters on the X-axis versus velocity in kilometers per hour on the Y-axis. The track used is the Bruntingthorpe Formula Student track and the cars are the Leeds 5 car and the Oxford Brookes 13 car.

Chapter 14

Conclusions and Recommendations Part II

To make an objective choice between two different Formula Student designs a lap time simulation has been carried out. The two designs are: a very light car having a light engine and a somewhat heavier car having a strong but heavier engine. Since on the basis of this information it is impossible to directly determine which car would perform best, there was a need for a lap time simulation.

The lap time simulation consists of a simple physical model of a car. Since little information is available the car is modelled as a point mass. The driver is modelled to drive the car to its maximum at all times, i.e. either grip or engine power limit the acceleration. The circuit is modelled by defining the curvature at all points.

Solving this model is done in an iterative way. Small steps in the path are used to go from this model to lap times. These lap times provide an objective criterion to decide which car is best suited for the Formula Student event.

From the result of these calculations it is not completely obvious which car is best. The light car is mostly faster, but only by very small margins. Therefore you might question the value of this difference, because it is not know wether it is even possible to design such a light car. The lap times of the two Eindhoven designs have also been compared to simulation lap times using parameters of competing cars. From this simulation it became clear that both designs are very competitive and even faster.

From the results of the sensitivity analysis it became clear that especially the weight of the car has a very big influence. A 10% increase of the weight makes a bigger difference, than the difference between the two Eindhoven designs. From this it is evident that one should carefully examine whether the estimated parameters for both cars can actually be realized.

A comparison of the calculated lap times to the ones realized by a real Formula Student car shows that the simulated lap time is quite a bit faster. This is no reason to think the simulation is not correct. A difference due to different parameters and a lot of idealizations in the model was expected. This difference does make clear that the lap times from the simulation are not likely to be reached in practise.

Appendices

Appendix A

M-Files for Converting the Track Picture

Conversion to curvature

```
clc
close all
clear all

% Define the number of points on the track.
%  Make it larger for a smoother result, smaller
%  for a more detailed result.
r = 8 ;
% Define the distance between 2 points for the
%  first segmentation of the track.
nop = 500 ;

% Calculate the position of the points on the
%  track in the right order and in x- and y-coordinates.
%  Also return the set of polynomials used for the spline.
[xy p] = track_seg('binnenbaan.bmp',r,nop) ;

% Split the polynomials in a X and a Y direction.
X = p.coefs(1:2:length(p.coefs)-1,:) ;
Y = p.coefs(2:2:length(p.coefs),:) ;

% Calculate the derivatives of the polynomials.
xp = [X(:,1)*3,X(:,2)*2,X(:,3)] ;
xdp = [xp(:,1)*2,xp(:,2)] ;
yp = [Y(:,1)*3,Y(:,2)*2,Y(:,3)] ;
ydp = [yp(:,1)*2,yp(:,2)] ;

s = length(X)-1 ;
x = pi*[0:2/s:2] ;

% Calculate the inverse radius of the spline.
for i = 1:s
    k(i) = 2*(X(i,3) * Y(i,2) - Y(i,3) * X(i,2)) / ( ((X(i,3))^2 + (Y(i,3))^2 )^(3/2) ) ;
end

% Make the outcome smoother and plot the inverse radius.
k = smooth(k,'lowess') ;
figure
plot(x(1:s)/2/pi*500,k)
```

Convert the track for VRML

```
close all
clear all
clc

% Define the number of points on the track.
nop = 300 ;
% Define the distance between 2 points for the
%  first segmentation of the track.
r = 5;

% Calculate the position of the points on the
%  track in the right order and in x- and y-coordinates.
xy = track_seg('binnenbaan3.bmp',r,nop) ;

% Define the startingpoint of the circuit.
deltax = xy(1,1) ;
deltay = xy(1,2) ;

% Move all points in such that the startingpoint
%  of the circuit lays at the origin.
xy(:,1) = xy(:,1) - deltax ;
xy(:,2) = xy(:,2) - deltay ;

% Make the start- and endpoint the same.
xy(nop,:) = xy(1,:) ;

% Draw a picture of the circuit.
figure ; axis equal ; hold on
plot(xy(:,1),xy(:,2),'.')
plot(xy(:,1),xy(:,2))

% Store the coordinates such that they can be
%  inserted in the VRML path of the circuit.
p(:,1) = xy(:,1) ;
p(:,3) = 0 ;
p(:,2) = xy(:,2) ;
```

Creating a map of the track

```
clc
clear nop r xy circuit deltax deltay x_start x_end y_start y_end

% Define the number of points on the track.
nop = 900 ;
% Define the distance between 2 points for the
%  first segmentation of the track. Make it larger for a smoother result,
%  smaller for a more detailed result.
r = 5;

% Calculate the position of the points on the
%  track in the right order and in x- and y-coordinates.
xy = track_seg('images\circuit.bmp',r,nop) ;

% Shift the values with a value of one.
xy = xy+1 ;

% Define a value for increasing the accuracy of track tracking. One
%  place in the final matrix represents 1/a meter in the virtual world.
a = 4 ;

% Make the start- and endpoint the same.
xy(nop,:) = xy(1,:) ;

% Multiply the coordinates with the accuracy value.
```

```
xy = a*xy ;

% Define the startingpoint of the circuit.
deltax = xy(1,1) ;
deltay = xy(1,2) ;

% Round the coordinates in order to insert them in a matrix.
x = round(xy(:,1)) ;
y = round(xy(:,2)) ;

% Define a matrix with ones.
sc = 1000 ;
circuit = ones(sc,sc) ;

% Define the desired width of the track in meters.
b = 3 ;

% Create a square of zeros with the same width of the track around each
% received coordinate on the track.
for i = 1:nop
    circuit(y(i)-b*a:y(i)+b*a,x(i)-b*a:x(i)+b*a) = 0 ;
end

% Define the startingline at the startingpoint of the track. To garantee
% the detection of each crossing, the width of this line is set to 4.
circuit(y(5)+1,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+2,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+3,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+4,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+5,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+6,x(5)-3*a:x(5)+3*a) = 3 ;
circuit(y(5)+7,x(5)-3*a:x(5)+3*a) = 3 ;


% Define the range which must be inserted in the lookup table.
x_start = -round(deltax)+1 ;
x_end = x_start+sc-1 ;
y_start = -round(deltay)+1 ;
y_end = y_start+sc-1 ;


clear deltax deltay i nop p r sx sy xy x y...
     x1 x2 x3 x4 x5 y1 y2 y3 y4 y5 x6 x7 y6 y7 sc

save circuit
```

Function used in these m-files

```
function [xy p] = baan_seg(afbeelding,r,nop) ;

fig = imread(afbeelding) ;
fig = double(fig) ;
fig = fig-fig(1,1) ;
fig = abs(fig) ;
sz = size(fig) ;
baan = zeros(sz) ;

x = 1 ;
y = 1 ;
while y <= sz(1)
    while x < sz(2)
        if fig(y,x+1) < fig(y,x) & fig(y,x-1) <= fig(y,x)
            baan(y,x) = 1 ;
        end
        x = x+1 ;
    end
```

```
    y = y+1 ;
    x = 1 ;
end

x = 1 ;
y = 1 ;
while x <= sz(2)
    while y < sz(1)
        if fig(y+1,x) < fig(y,x) & fig(y-1,x) <= fig(y,x)
            baan(y,x) = 1 ;
        end
        y = y+1 ;
    end
    x = x+1 ;
    y = 1 ;
end
[Y,X] = find(baan) ;

y_start = 100 ;
x_start = min(X(find(Y==y_start))) ;

y = y_start ;
x = x_start ;

stop = 0 ;
XY = [x_start y_start] ;
while stop < 1

    ymin = max(y-r,1) ;
    ymax = min(y+r,sz(1)) ;
    xmin = max(x-r,1) ;
    xmax = min(x+r,sz(2)) ;

    if y == y_start & x == x_start
        yr = ymin:y ;
    else
        yr = ymin:ymax ;
    end
    xr = xmin:xmax ;
    [yp,xp] = find(baan(yr,xr)) ;

    yp = yp+ymin-1 ;
    xp = xp+xmin-1 ;

    d = sqrt((yp-y).^2+(xp-x).^2) ;
    i = find(d == min(abs(d-r))+r) ;
    if length(i) < 1
        i = find(d == (r - min(abs(d-r)))) ;
    end
    if length(i) > 1
        i = i(floor(length(i)/2)) ;
    end
    XY = [ XY ; xp(i) yp(i) ] ;

    y = yp(i) ;
    x = xp(i) ;

    baan(yr,xr) = 0 ;
    if (x-x_start)^2+(y-y_start)^2 < r
        stop = 1 ;
        XY(length(XY),:) = [x_start,y_start] ;
    end
end

XY(:,2) = sz(1) - XY(:,2) ;
```

```
s = length(XY)-1 ;
x = pi*[0:2/s:2] ;

p = csaps(x,XY') ;
yy = ppval(p,linspace(0,2*pi,nop)) ;

xy = yy' ;
```

Appendix B

Alternative Approaches

B.1 Joystick input using the Real-Time Workshop

The standard joystick input block from the Virtual Reality Toolbox does not work in combination with the Real-Time Workshop. If you do want to use input from a joystick or steering wheel in real-time some other way has to be found. The solution is using the Real-Time Windows Target. Here there are blocks available called analog input and digital input. These blocks can be configured in such a way that they respond to the steering wheel.

First the device has to be selected using install new board. The joystick can be found under Standard Devices. The other block parameters can be found in figure B.1.

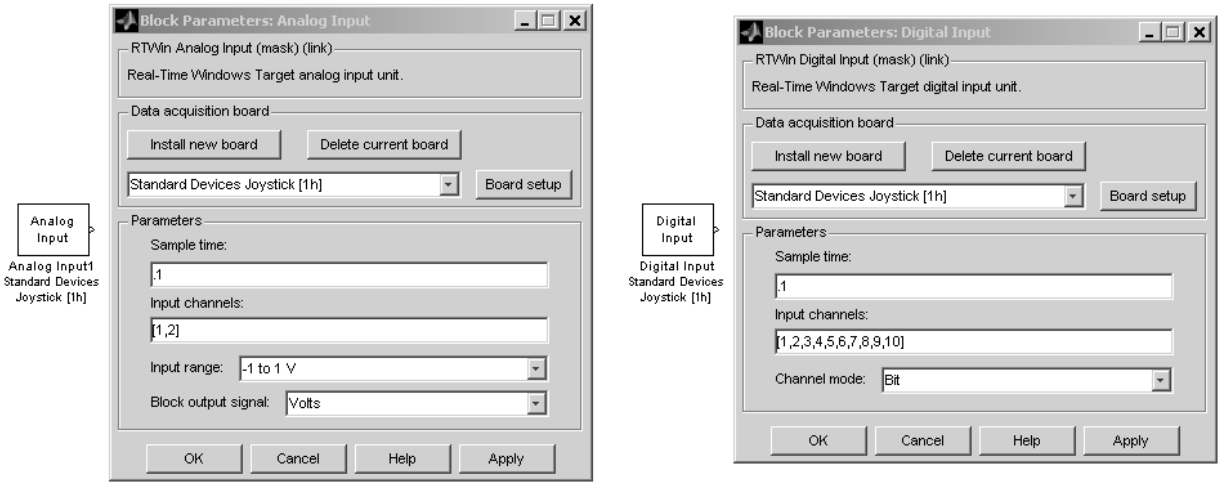

Figure B.1: The configuration of the input blocks.

The method described in this appendix does work, but is not used in the simulator. The simulator does not use the Real-Time Workshop, so the standard joystick block can be used.

B.2 TCP/IP communication

As an alternative for UDP, TCP/IP communication can be used. This communication protocol gives more reliable communication than UDP, because it does not only send the information, but also checks if the information has been received. TCP/IP is not implemented in standard Simulink. To implement it in the simulator, a block from the internet [8] has been used. Though this protocol did work, it was not as convenient as UDP. Using it gave errors from time to time. Since it is not that much of a problem in our simulator if information doesn't arrive, as long as almost all information does arrive, UDP is used in the simulator.

B.3 Gauges blockset

While racing you would like to know some things about the status of your car, e.g. the speed at which you are driving, the engine RPM and the gear. Therefore this information must be made available in the visualization. In Simulink it is easily possible to use displays for this purpose, but to make it look nice and more realistic, the gauges blockset can be used. In this blockset some standard automotive instruments are available, these can be used for the engine speed and vehicle speed. These instruments have been adapted to fit our needs. For the time and gear LED displays have been used.

Appendix C

MATLAB Function for the Lap Time Simulation

```
function [T,V,AX,AY,mu] = Lap_Time_Simulation(m,P,Cd,A,iR,dx)
%% Usage
% Simulate a car characterized by it's mass m, engine power P,
% dragcoëfficiënt Cd and frontal surface A along a track defined by it's
% curvature iR using stepsize dx

%% Define constants
mu = 1.74 - 1.28e-4*m/4*9.81 ; % According to the Master's Thesis of Andries
rho = 1.2929 ; % Densitity of air in kg/m^3

%% Reverse
v = 100 ;d =  max(iR(2,:)) ; Vm = 100 ; % Initialize
while d > 0
    ir = interp1(iR(2,:),iR(1,:),d,'linear') ; % Get the current curvature

    % Calculate the lateral force using the curvature and speed
    Fy = m * v^2 * ir ;

    % Calculate how much grip there is left in longitudinal direction
    Fx_grip = sqrt(mu^2*(m*9.81)^2 - Fy^2) ;
    if imag(Fx_grip) ~= 0 , Fx_grip = 0; end

    Fd = .5 * Cd * A * rho * v^2 ; % Calculate the air drag

    Fx = Fx_grip +  Fd ; % Calculate the resulting lateral force

    v = sqrt(v^2 + 2*Fx/m*dx) ; %Update the speed

    % check if the speed is too high for the current curvature
    if ir ~=0, v=min(v,sqrt(mu*9.81/abs(ir))); end;
    Vm = [v Vm] ;

    % Update the position and continue at step 1
    d = d - dx ;
end

%% Forwards
% Initialize
v = 0 ;d = 0 ; t = 0 ;
T = 0 ;V = 0 ; AX = [] ;AY = [] ;
while d < max(iR(2,:))
    ir = interp1(iR(2,:),iR(1,:),d,'linear') ; % Get the current curvature

     % Calculate the lateral force using the curvature and speed
```

```
    Fy = m * v^2 * ir ;

    % Calculate how much grip there is left in longitudinal direction
    Fx_grip = .65*sqrt(mu^2*(m*9.81)^2 - Fy^2);
    if imag(Fx_grip) ~= 0, Fx_grip = 0; Fy = mu * m * 9.81 * sign(ir); end

    Fx_motor = P/max(v,1) ; % Calculate the force the engine can deliver

    Fd = .5 * Cd * A * rho * v^2 ; % Calculate the air drag

    Fx = min(Fx_motor, Fx_grip) - Fd ; % Calculate the resulting lateral force
    AY = [AY Fy/m] ;
    vo = v ;

    v = sqrt(v^2 + 2*Fx/m*dx) ; %Update the speed

    % Check if the speed is not to high according to the reverse simulation
    Vmax_rem = interp1(iR(2,:),Vm(1,:),d,'linear');
    v=min([v,Vmax_rem]);
    if v == Vmax_rem, Fx = -Fx_grip/.65 - Fd;     end
    AX = [AX Fx/m] ;

    % Update the time and position and continue at step 1
    t = t + dx/( (v+vo)/2 ) ;
    T = [ T t ] ;     V = [V v] ;          d = d + dx ;
end
AX = [AX 0];AY = [AY 0];
```

Bibliography

[1] A. van Berkum. Chassis and suspension design fsrte02. Master's thesis, Eindhoven University of Technology, the Netherlands, 2006.

[2] I. Besselink. Vehicle dynamics 4l150. Lecture Notes Vehicle Dynamics 2005-2006, 2005.

[3] The Matworks, inc. *MATLAB R2006a help files*, 2006.

[4] Society of Automotive Engineers. *2006 FORMULA SAE® RULES*.

[5] H.B. Pacejka. *Tyre and vehicle dynamics*. Elsevier Butterworth Heinemann, 2 edition, 2006.

[6] B. Siegler. Formula student event track data. Technical report, Honda Research and Development Europe (UK) Ltd.

[7] N. van de Wouw. *Multibody dynamics : lecture notes*. Eindhoven : Technische Universiteit Eindhoven, 2005.

[8] W. Zimmermann. Iolib, May 2006.