Road Vehicle Simulation Infrastructure

Abstract

The subject of this report is a development process and infrastructure that facilitates the creation and verification of vehicle dynamics models. The infrastructure provides facilities for generating verified computer code that can be easily incorporated into any other simulation environment that can interface with C++ libraries. The author of this report performed a simplified dynamics analysis on a road vehicle model and used Simulink to create a system and interface for simulating and verifying that model. The simulation verification system has also been designed to facilitate the addition of more complex vehicle models and other vehicle types to allow for continued development in the future.

Contents

List of Figures

List of Tables

Nomenclature

q	10x1 vehicle position vector, $[m]$
dq	10x1 vehicle velocity vector, $[m/s]$
$t1, t2, t3$	euler angles of vehicle when passed to a coordinate transformation function, $[rad]$
F_t	tire force vector, $[N]$
k_t	tire stiffness coefficient, $[N/m]$
c_t	tire damping coefricient, $[Ns/m]$
δ_s	suspension displacement $[m]$
$\dot{\delta_s}$	rate of change of suspension displacement $[m/s]$
F_c	cornering force vector, $[N]$
α	slip angle, $[rad]$
C_α	tire stiffness coefficient $[N/rad]$
θ	angle between the vehicle's x-axis velocity of a wheel in the xy-plane of the vehicle, $[rad]$
δ	steering angle, $[rad]$
X, Y, Z	global coordinate system axes
x, y, z	local coordinates system axes

1 Introduction

In the creation of a software vehicle simulation, the development of mathematical models and computer code to mimic them is only a portion of a larger process. In order to complete the simulation, a method of verification is necessary, which is a matter of determining whether the mathematics and code behave as the designer intended. When calculating vehicle model mathematics, verification is easy to neglect until the model becomes very complex. If there is no infrastructure laid out to verify the model, then fixing problems with unclear causes can be very difficult and time consuming.

The purpose of this project was to develop a system to facilitate the simulation verification process. It uses common software tools - Microsoft Visual C++ and MATLAB's Simulink - to provide a powerful and interoperable interface for simulating and testing vehicle dynamics models. It separates each component of the simulation into a separate block that can have its inputs and outputs analyzed and recorded. The power of this system is that any component of the simulation can be interchanged with another simpler or more complex model, and by anyone with knowledge of the component's interface. After a model has been verified, C++ code that performs the same calculations as the Simulink model can be generated in a form that allows the model to be simulated within any software environment compatible with C++ code libraries.

1.1 Project Background

The creation of the simulation development software in this project is part of a larger multi-year multi-disciplinary university simulation project, which is the development of a multifunctional and reconfigurable motion simulator. Because of the project's multi-year nature, some contributions to this project were not developed this year. However, all of the mathematics and software development discussed in this report were completed by one student in this year's project, and it will be made clear where an aspect of the project was contributed to by an external source. Additionally, as much as possible, effort was made to allow the work started in this project to be easily continued by other students in subsequent years of the project.

1.2 Software Tools

There are many existing software packages that allow the equations of motion of various physical systems to be automatically generated. In this project, such tools were avoided in order to allow complete control over the simulation mathematics. MATLAB's Simulink was the primary software tool used in this project because of its widespread use in universities and in industry. Simple mathematical calculations are performed in Simulink, and complex ones use C++ to extend Simulink's capabilities. Microsoft Visual Studio .NET 2003 was used for all C++ development. Finally, two extension toolkits for Simulink were used. The first, the Virtual Reality toolkit, was used for 3D visualization, and the second, Real-Time Workshop, was used to generate C++ code based on the Simulink model.

1.3 Outline

There were two main components to this project. The first was the development of road vehicle model mathematics to be simulated. The second component was a system to incorporate those mathematics into a computer software system where they could be tested and verified, and then be able to be utilized in any other simulation system.

2 Road Vehicle Modelling

2.1 Introduction

In the development of the simulation environment, it was decided that a road vehicle model (RVM) would be used to test each component of the simulation. A road vehicle dynamics model consists of three main components: a series of models that calculate the external forces acting on the vehicle, the governing equations of the vehicle that use the external forces to calculate accelerations, and a time-integration component to obtain updated vehicle velocities and positions from those accelerations.

There are many external forces acting on a road vehicle. These forces consist of gravitational, suspension, tire compression, propulsion, braking, aerodynamic, cornering, drawbar and rolling resistance forces. For

this project two of these forces were modelled in detail and four others were used in a simplified form or obtained from an outside source. Simple models were used for gravitational, suspension, and braking forces. The propulsion model was developed by another student in a related project. Complex models were developed for tire compression forces and cornering forces, and will be discussed in more detail in the following sections. The reason these two were chosen is because it is difficult to determine simple models that mimic their behavior, and between them only vehicle motion in the x-direction is not affected. (Coordinate conventions can be seen in Figure 1.) Tire compression forces affect the vehicle motion in the z-direction, its roll and its pitching motion, and cornering forces affect the vehicle's motion in the y-direction and its yawing motion. Once all of the rotational degrees of freedom have been stabilized, moving the vehicle along linear axes is much easier to model, and these two external forces models dominate the vehicle's angular accelerations.

Each external force derivation consists of three main components: establishing the mathematical equations that govern its behavior, determining a step-by-step process that describes how to calculate all of the variables in the mathematical equations based on available data, and development of C++ code that mimics that process. This process was carried out for the two complex models.

2.2 Governing Equations

The road vehicle model consists of ten degrees of freedom. Six degrees of freedom come from the main vehicle body, and the additional four are present because each of the four wheels is permitted relative motion to the main body in one direction. The governing equations used in the road vehicle model were developed in a related graduate student project. Kane's method was used to derive the equations for each of the vehicle's five bodies. The C++ code used in the governing equations was developed outside of this project.

2.3 Coordinate System Conventions

It is important to clearly define coordinate system conventions in a mathematical model of a physical system.

All of the road vehicle modelling leads to resultant accelerations, velocities, and positions in ten degrees of freedom. These ten values are a combination of global and local coordinates. The ten values stored in the calculations consist of X, Y, Z, roll, pitch, yaw, and the four wheel suspension compressions. There are two coordinate system conventions. One set of coordinate axes is global and the vehicle moves relative to it. The second set of coordinate axes are fixed to the vehicle's centre of gravity and translates and rotates with the vehicle. These axes are the ones illustrated in Figure 1.

The X, Y, and Z coordinates are all in the global coordinate system. The roll, pitch and yaw angles are Euler angles in the vehicle-fixed coordinate system and are performed in the order yaw-pitch-roll. The last four degrees of freedom are the compressions of the suspensions for wheels A, B, C, and D respectively. A value of zero for any of these compressions means that the suspension spring is uncompressed, and the distance from the centre of mass of the vehicle body to the tire centre in the z-direction is equal to $h2$. Therefore, if a spring compression is positive then in order to determine the z-location of the centre of the tire in the vehicle-fixed coordinate system the compression must be subtracted from $h2$. The location of $h2$ and all of the other vehicle dimensions are shown in Figure 1 and are defined in a vehicle header file that is included in all of the RVM C++ code.

2.4 Tire Compression Forces Model

Whenever the vehicle is on the ground there are forces acting on it from the ground. These forces must always be continuously recalculated because any force acting on the vehicle that generates a moment will change the forces acting on the tires. This acts as a safety net for balancing any unrealistic forces that may be applied to the vehicle during model development, but if the tire forces mathematics is not performed correctly, then the vehicle simulation will not remain stable for more than a few seconds at the most.

In the tire forces model the tire is assumed to be a spring-damper system, and the vehicle behavior is analyzed by calculating the reaction of the vehicle body when it is placed on the ground. The basic theory behind the tire compression forces model is that if the location of the terrain and the location of the road

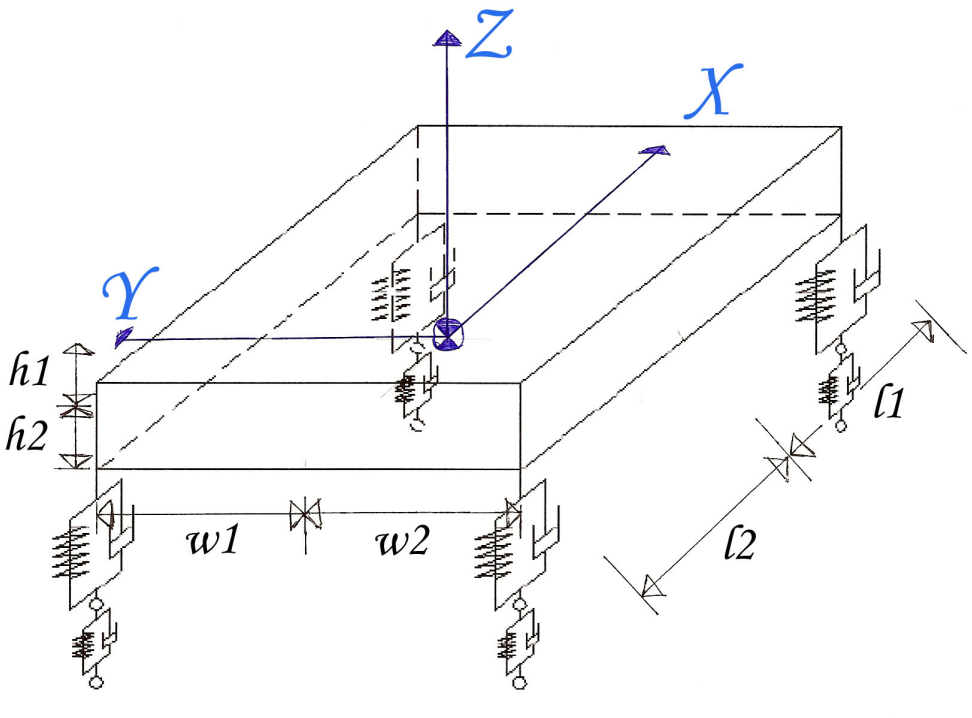

Figure 1: Road vehicle model local coordinate system and vehicle dimensions

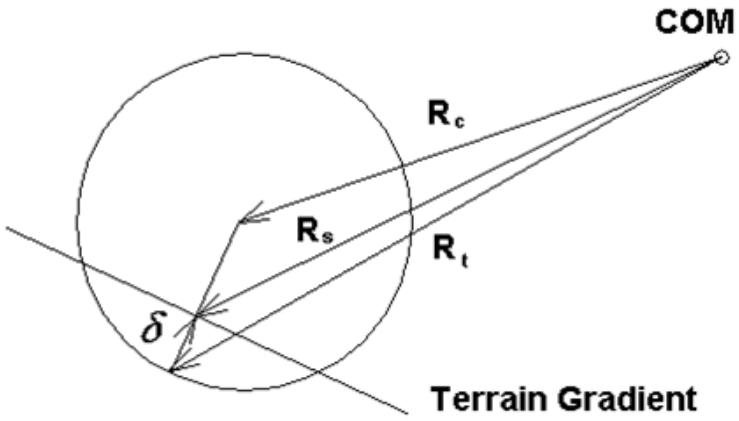

Figure 2: Tire forces model diagram

vehicle's tires are known, it can be determined whether its tires are on the ground or not. If a tire overlaps with the terrain, as is the case in Figure 2, then its distance within the terrain can be represented as δ_s. The force acting upward on the vehicle at the tire is then simply:

$$F_t = k_t \delta_s + c_t \dot{\delta_s} \tag{1}$$

In this equation, k_t is the spring coefficient of the tire and c_t is the damping coefficient of the tire. In the current implementation of the road vehicle, the values used for these variables are 248,000 N/m and 1000 Ns/m respectively. While this tire forces model is difficult to stabilize, the theory behind it is sound.

2.4.1 Tire Forces Calculation Procedure

This road vehicle model assumes that the terrain is completely flat, and that the point of contact between the tire and the ground is at the bottom of the tire in the vehicle's coordinate frame. The calculation procedure is as follows:

1. Determine the absolute location of each of the tires in the global coordinate system. Since only the relative location of each wheel with respect to the vehicle centre of gravity in the body-fixed coordinate system is stored, a transformation matrix must be used in this calculation.

2. Based on the location of each wheel with respect to the elevation of the terrain surface, the distance between the terrain and each wheel can be calculated.

3. If the distance is greater than 0, the tire is not on the ground and the tire force on that tire is zero.

4. If the distance is less than 0, the tire is under the ground, and thus must be compressed.

5. The magnitude of the tire force is shown in Equation 1 and is calculated to be the sum of the distance multiplied by the spring constant and the damping force multiplied by the rate of change of that distance.

6. The damping force requires the rate at which the tire is compressing with respect to the ground. This rate is estimated as the Z-velocity of the vehicle body plus the velocity of the wheel with respect to the vehicle.

2.5 Cornering Forces

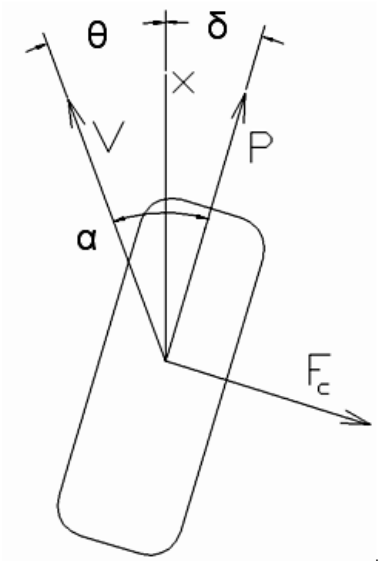

Figure 3: Cornering forces model diagram

The basic theory behind the cornering forces model is very simple. It states that the cornering force acting on any wheel, F_c, is simply the cornering stiffness, C_α, which is a vehicle property, multiplied by the slip angle, α. The value of C_α currently being used is 1000 N/rad.

$$F_c = C_\alpha \alpha \tag{2}$$

The slip angle, α, is the angle between the velocity vector of the wheel and the pointing vector of the tire. This angle can be determined easily since the dot product of two vectors is equal to the product of their magnitudes and the cosine of the angle between them. Unfortunately determining the direction of the

resultant cornering force in three dimensions is not as trivial so a simplified 2D analysis is used to determine the slip angle instead.

In this cornering forces model the absolute velocity of each tire is first calculated in the global coordinate frame. This is done using rigid body dynamics mathematics:

$$v_A = v_O + v_{A/O} = v_O + w_O \times r_A + v_{A_{xyz}} \tag{3}$$

This equation states that the velocity of a point ($_A$) attached to a rigid body is equal to the velocity of the rigid body's centre of rotation ($_O$) plus the velocity of the point relative to the rigid body. This relative velocity is then equal to the cross product of the angular velocity of the rigid body and the distance between the rigid body's centre of rotation and the point plus the velocity of the point in its own coordinate frame. In this model the value of $v_{A_{xyz}}$ has been neglected since it is simply the rate at which the vehicle's suspension is changing. This value not only requires a complicated coordinate transformation to bring it into the global coordinate system but is also acting perpendicular to the orientation of the xy-plane of the vehicle and thus does not contribute to our 2D analysis.

Next the velocity vector of each wheel is transformed into the local coordinate system of the vehicle, and only its components in the xy-plane of the vehicle are considered. The pointing vector of the wheel is determined in the plane of the vehicle.

There is an additional simplification in the cornering forces model that is not immediately apparent. In road vehicle physics, the relationship between the slip angle and the force is not exactly linear. However, this relationship can be assumed to be linear for modelling purposes up to a slip angle of approximately six degrees. At six degrees the magnitude of the force reaches a maximum and afterwards the force starts to decrease. In this model, the assumption used is that for slip angles greater than six degrees, the cornering force is equal to the force that would be present if the slip angle was still six degrees. This means that if the steering angle is greater than six degrees the cornering force will not increase, and only the propulsive force

will have an additional effect on the yaw rate of the vehicle.

2.5.1 Cornering Forces Calculation Procedure

Most of the cornering forces model has been described in the previous section, but the exact procedure that was used to calculate cornering forces is listed below.

1. Determine the velocities of each of the wheels in the global coordinate frame.
2. Transform the wheel velocities into the xy-plane of the vehicle body.
3. Using the x-axis of the vehicle as a zero-degree reference axis, determine the angle of the pointing vector of each wheel, δ.
4. Calculate the angle of the velocity vector of each wheel using the x-axis as a zero-degree datum using $\theta = tan(\frac{v_y}{v_x})$. If either the x or y component of that velocity is equal to zero, increment it by a small amount to avoid singularities in the mathematics. Since the tangent function gives values between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ and the area around the vehicle ranges from $-\pi$ to π, check the signs of the x and y components of velocity to see if θ needs to have π added or subtracted from it.
5. Next the direction in which the cornering force acts needs to be determined. This is done by comparing the angle of the velocity vector (θ) and the tire pointing vector (δ).
6. If $\theta > \delta$ and $\theta < \delta + \pi$, then the angle of the cornering force is $\delta - \frac{\pi}{2}$.
7. If $\theta < \delta$ and $\theta > \delta - \pi$, then the angle of the cornering force is $\delta + \frac{\pi}{2}$.
8. The magnitude of the cornering force is calculated as the product of the absolute value of the difference between δ and θ and the cornering stiffness of the tire.
9. Given the magnitude of the cornering force and the angle that it acts at the force's components are separated into their components in the x and y-axes of the vehicle coordinate frame.

10. Repeat the previous four steps for each wheel of the vehicle that is on the ground.

3 Road Vehicle Model Development Infrastructure

3.1 Introduction

This section of the report discusses RVSIM, a development environment for the road vehicle model (RVM). It uses C++ and Simulink to provide a powerful, yet easily reconfigurable simulation infrastructure for developing the road vehicle model. Once the expected performance of the road vehicle model has been verified, the Simulink model can then be exported to code form using Real-Time Workshop. This guarantees that the generated code version of the simulation will work exactly as expected. A method for compiling this code into a dynamic link library (DLL) was developed which can then be easily accessed by any other C++ program.

3.2 Design Requirements

The design requirements for RVSIM were based on the behavior of an unsuccessful attempt to use Simulink to simulate a road vehicle model in a previous year. The components of that simulation that were used in this year's project are only the C++ code for the vehicle' s governing equations and the Simulink interface between the vehicle mathematics and a VRML 3D visualization Simulink block.

The basic idea of RVSIM was to take the governing equations and visualization of the existing model, and to add to it an infrastructure in Simulink for separately calculating each external forces model and to visually manage how those forces are added together on the vehicle's five bodies. Each forces model could then be connected to Simulink's data display and plotting tools so that the vehicle mathematics could be monitored during the simulation.

Based on the existing components, and future goals, design requirements were specified and are listed below.

- Before being transferred to the governing equations of the road vehicle model, all of the forces acting

on the road vehicle model's five bodies shall be added together in a 30x1 double array in Simulink. (30 = six degrees of freedom for each of the five bodies.)

- Vehicle operator inputs shall be managed separately from the road vehicle model subsystem and their values will be fed directly into each external forces model as required. The primary motivation behind this is that it will mimic the way vehicle input is handled when the RVM is implemented in the simulator's high level architecture (HLA).

As work on RVSIM progressed through the year, some additional design requirements were defined.

- The road vehicle model shall use a right-hand coordinate system, instead of the left-hand system that was used up until the end of 2005. This shall facilitate current and especially future mathematical modelling and coordinate system transformations.
- The external forces calculations shall be time-invariant, in order to facilitate the process of time-integrating the resultant accelerations.
- The Simulink model shall be designed such that it will be compatible with Real-Time Workshop.
- All vehicle-specific parameters shall be stored in a single header file.

Design guidelines for the C++ code that was to be written were also created. They are listed below.

- Each model function shall be independent of its class and gather data in one of 3 ways:
    1. constants located within the function (local) or in the header file of the class (global);
    2. variables/constants passed into the function as arguments; or
    3. values derived within the function based on items 1 and 2.
- All model functions and intermediate calculation functions shall be static functions.

- Intermediate calculation functions shall be private within the class that contains the vehicle forces functions.
- All model functions shall have a return type of void and return their results as function arguments. This may be changed in the future for debugging purposes.
- The first and second arguments of a model function shall be **q** and **dq** (position and velocity of bodies in the RVM). The third argument shall be the function return value and any model specific inputs shall be included 4th or later.

All of these programming design requirements were closely adhered to in RVSIM.

Once the road vehicle's performance in Simulink was satisfactory, it became important to develop an infrastructure that would allow the road vehicle model in Simulink to be exported to C or C++ code that could be easily incorporated into an external simulation framework. A set of design requirements were decided upon to govern how this process would be developed.

- The road vehicle model shall be compiled into a dynamic link library (DLL) for ease of portability and implementation.
- A single time-step calculation shall be made possible by calling a single function in the DLL.
- Concerning the integration of Real-Time Workshop automatically generated code, the generated code shall not be modified and will be accessed through the use of a wrapper file that will call the DLL's accessible functions. This will allow updates to the code to be performed as efficiently as possible.

3.3 Infrastructure Development

There were two major infrastructure accomplishments made in this project with respect to the road vehicle model. The first was the expansion of the C++-to-Simulink infrastructure that facilitated the verification of the road vehicle model mathematics. The second was the process by which the road vehicle model in Simulink

could be exported into code that could be incorporated an external simulation environment. Following is a detailed description of each stage of the development process, which is illustrated in Figure 4.

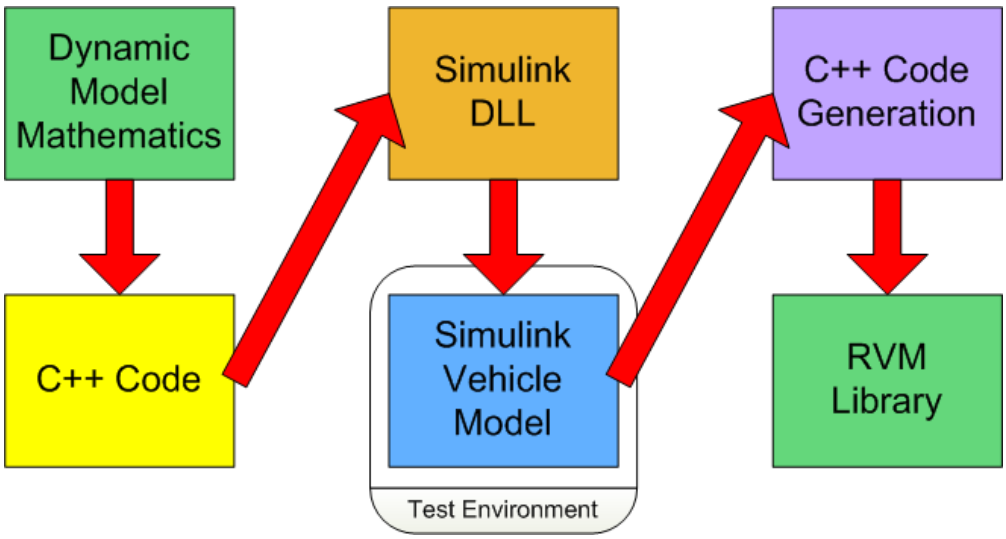

Figure 4: Road vehicle model development process

3.4 C++ Code Mathematics

In the first step of the road vehicle development process the three-dimensional mathematics of each external force are written in C++. This is done in C++ instead of MATLAB because the simulator that the vehicle model is being designed for uses software written in C++. The major disadvantage of not using MATLAB is not being able to utilize its ability to work with matrices. This was overcome by using the freeware version of a C++ matrix library for matrix multiplication and inversion. Matrix multiplication is used in coordinate transformations and inversion is used in the vehicle's governing equations.

3.5 Simulink S-Function Implementation

Performing complex mathematical calculations can be very difficult and time consuming to program in Simulink. For this reason, 'S-Function' blocks were used in Simulink to import C++ code. These blocks

are created by inserting custom C++ code into an S-Function C++ template provided with MATLAB. The template is compiled using Microsoft Visual Studio in order to produce a dynamic link library (DLL) which can be loaded into Simulink.

3.6 RVSIM

RVSIM is the name of the Simulink testing environment for the road vehicle model. An image of RVSIM being used to test the road vehicle model can be seen in Figure 5.

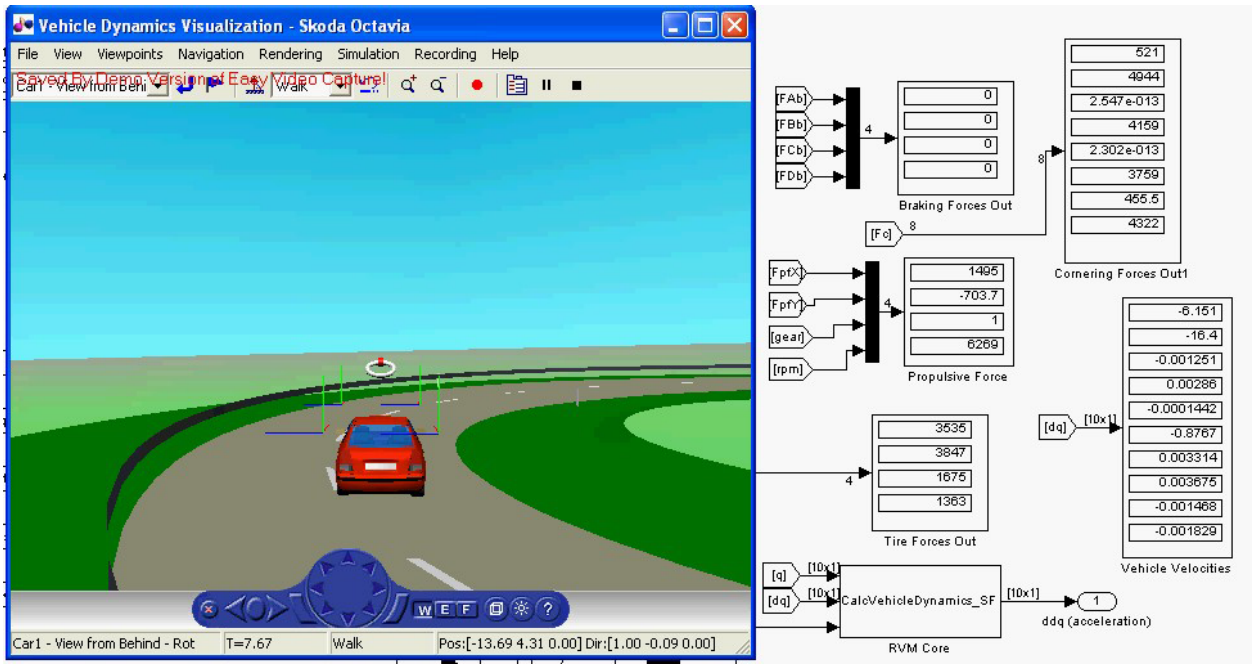

Figure 5: RVSIM demonstration

3.6.1 Information Flow

Simulink is basically a visual programming language. Thus, program components are separated into blocks, which are the equivalent of C++ functions or FORTRAN subroutines. The input and output variables of each block are then connected to other blocks so that one can easily visualize the flow of information in their

program.

One of the disadvantages of a program flow of this type is that it can become very cluttered with connection paths between blocks. In the road vehicle model Simulink program, "goto" and "from" blocks have been used extensively in order to manage the large amount of information that is computed. A quick example of this is in Figure 6. This picture shows the joystick input subsystem completely isolated, yet the values it calculates are being processed by other blocks. This is because the $deltaIn$, $pbIn$ and $ppIn$ blocks are all "from" blocks, and the corresponding "goto" blocks are located inside the joystick subsystem shown in Figure 7. Another advantage of this connection method is that if one had an alternate joystick input subsystem in another model file, the subsystems could be easily swapped by cutting and pasting the blocks between files.

The purpose of the joystick subsystem is not only to grab the input from a joystick, but to transform that input to the data ranges that are expected by the road vehicle model. Table 1 displays those data ranges. Delta is the steering angle of the wheels and not the steering wheel. The transformation from steering wheel angle to actual wheel angle is not taken into account anywhere in the road vehicle model at the moment. This is because the steering wheel input device does not give actual steering wheel inputs so its values are directly translated to wheel angles. The steering angle units are radians.

Table 1: Expected input value ranges to the vehicle model

Input	Min	Max
Gas	0.0	1.0
Brake	0.0	1.0
Delta	-0.44	0.44

3.6.2 RVSIM

Figure 8 displays the highest level of RVSIM, and is the first model window that will appear when RVSIM is loaded. It consists of three subsystems and a VRML virtual reality sink. The subsystem in the lower left

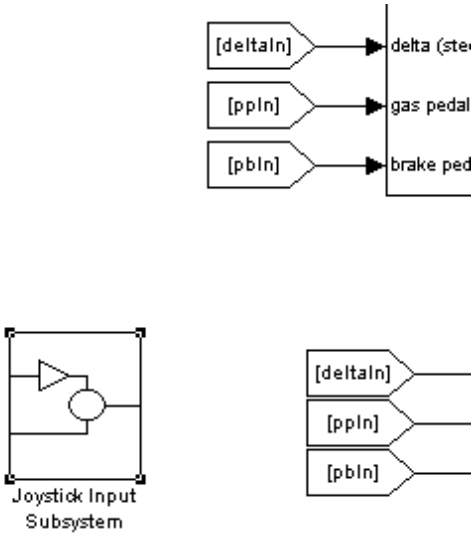

Figure 6: Modular joystick subsystem connections

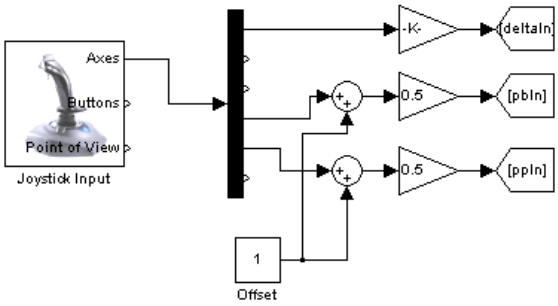

Figure 7: Customized joystick subsystem with goto blocks

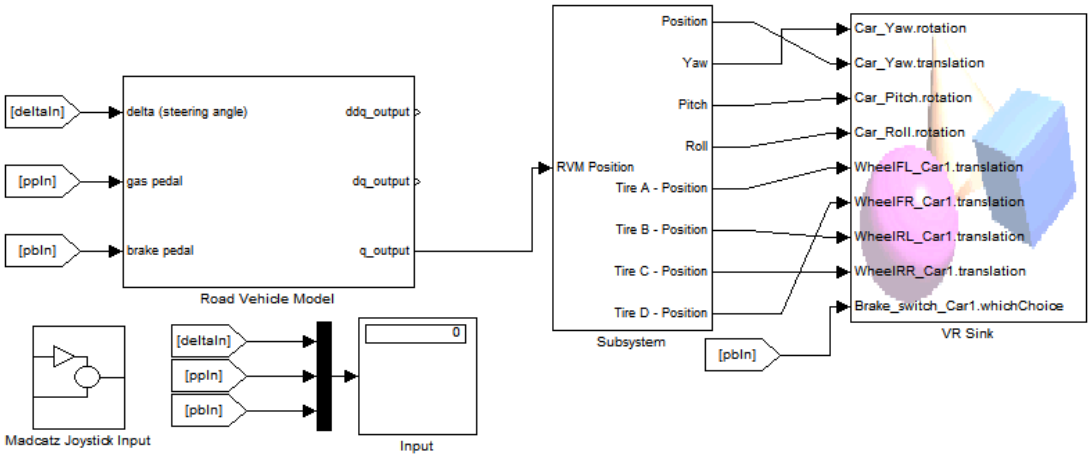

Figure 8: Top level of RVSIM

manages joystick input and can be seen in more detail in Figure 7. The road vehicle model subsystem is where all of the road vehicle calculations are performed and will be discussed in detail in later sections. The output of this subsystem is an updated vehicle position. It also outputs vehicle acceleration and velocity which can be connected to display sinks in order to view their values during a simulation in this window if desired. The vehicle position is connected to the third subsystem which transforms the vehicle positions into the format required by the VR sink. The VR sink is the final block in this program and its output can be viewed by double-clicking it. The VRML model used in this project is a simple racecar model that is included with Simulink.

3.6.3 Road Vehicle Model

The Simulink subsystem that performs all of the road vehicle calculations can be viewed in Figure 9. It is definitely complex, but consists of only four primary components. The first is the external forces models on the left side of the window. All of them are C++ S-Functions except for suspension and gravity. The next component is the force summation in the centre of the figure. The third component is the arrangement of blocks that the force summation feeds in to, which are the governing equations of the road vehicle model and

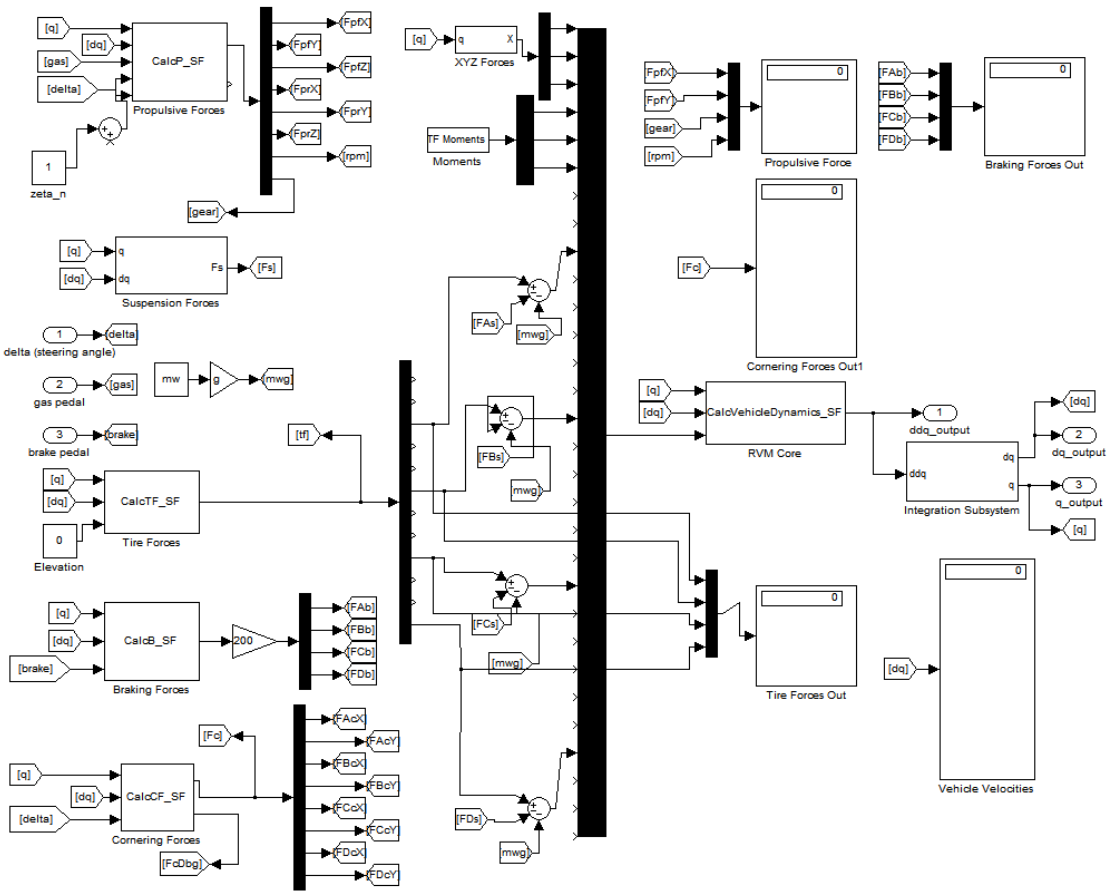

Figure 9: Primary subsystem of RVSIM

blocks which integrate the resultant accelerations to obtain velocities and positions. The last component is the arrangement of display sinks on the right side of the figure. They are used to display the values of the external forces models and can be monitored during a simulation's execution.

3.6.4 External Forces Models

Each external forces model in RVSIM will now be discussed in detail. All of them that require subsystems for their calculations take [q] and [dq] as their first two inputs, even if their values are not required. This is because of the design requirements that were discussed in Section 3.2. The mathematics of these external forces models are discussed in Section 2.

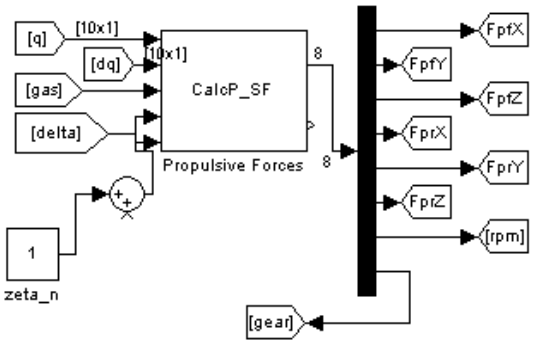

Figure 10: Simulink propulsive forces

Figure 10 displays the place where the propulsive forces are calculated. The inputs are gas pedal, brake pedal, steering angle, and current gear. In the future this gear input could be used to change the transmission gear if the propulsion S-Function was modified to model a manual transmission engine. The block with $CalcP\_SF$ is an S-Function block. Its first output contains eight elements which consists of the propulsive force in three dimensions at the front wheels, the rear wheels, the engine RPM and the current transmission gear setting. The propulsion mathematical model currently being used was developed by an external source but the C++ code for it was written as part of this project.

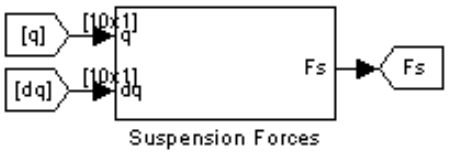

Figure 11: Simulink suspension forces

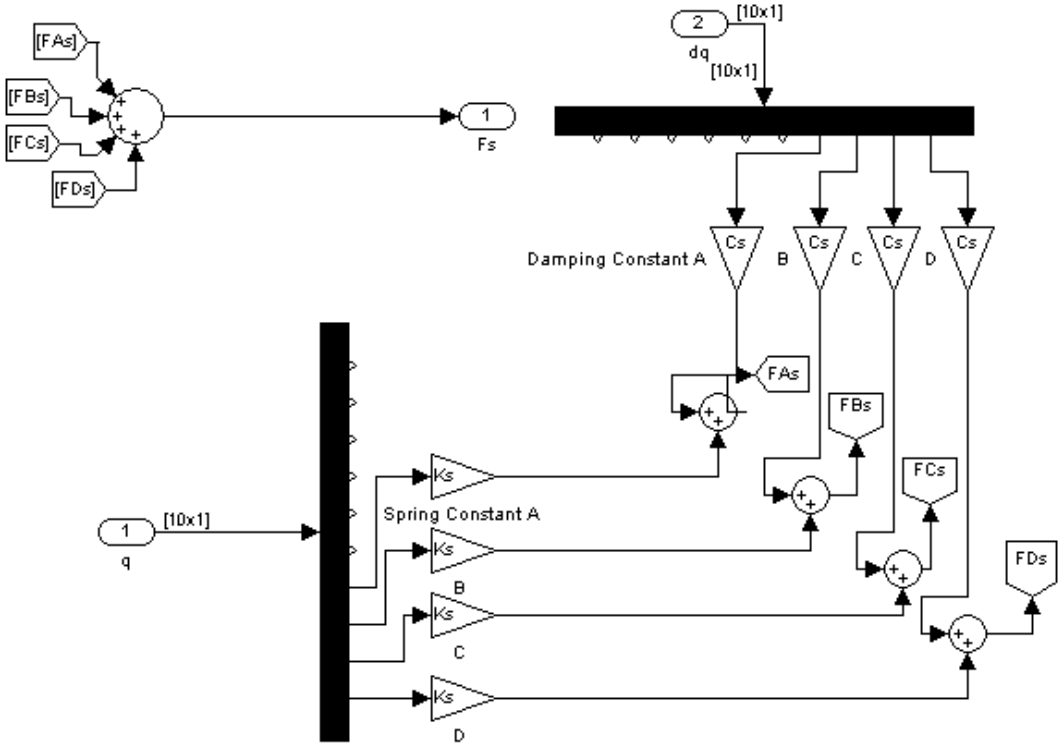

Figure 12: Simulink suspension forces subsystem

Figure 11 is of the suspension forces subsystem. It has no additional inputs, and outputs only the suspension force acting on the main vehicle body Fs, which in this case is a global "goto" block because its value is used inside another subsystem. Since the suspension forces calculations are simple they are all performed within Simulink. If the subsystem in Figure 11 is double-clicked, the Simulink model in Figure 12 will appear. This subsystem calculates the suspension forces acting on each of the wheels and sums them together to calculate the suspension force acting on the vehicle body. "Goto" blocks are used for the forces acting on each of the wheels since their values are used in the force summations around each wheel as well as the moment summation on the main vehicle body.

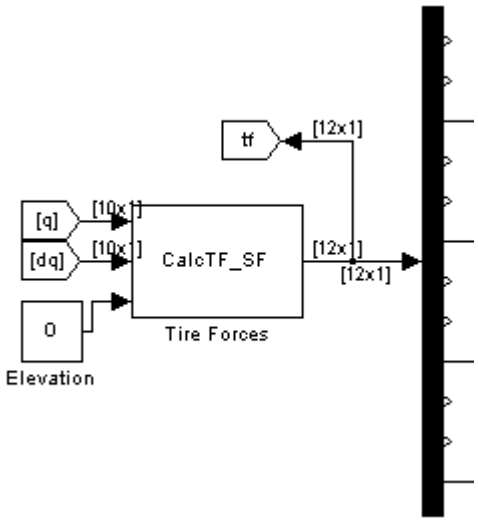

Figure 13: Simulink tire compression forces blocks

The Simulink blocks that calculate the tire compression forces acting on the road vehicle are displayed in Figure 13. Since this model assumes that the terrain surface is completely flat the only additional input is the elevation of that terrain. The output of the model is the force vector acting on each wheel in three dimensions. Again, since the model assumes that the terrain is flat, only the Z-component of each force is connected to a force summation.

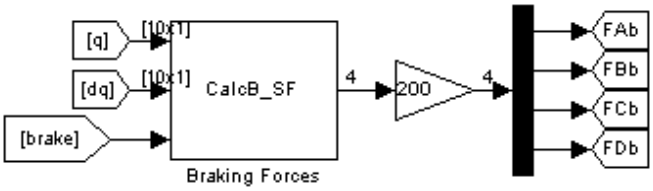

Figure 14: Simulink braking force blocks

Figure 14 displays the braking model that is currently implemented in RVSIM. This model is inaccurate and was only implemented to provide rudimentary braking ability for road vehicle demonstrations.

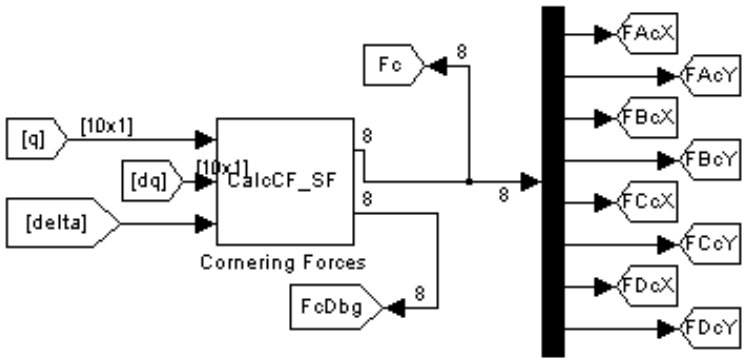

Figure 15: Simulink cornering forces model

The Simulink blocks that calculate cornering forces are displayed in Figure 15. This model requires the steering wheel input and has two output ports. The first output port contains the cornering forces acting on each wheel in the x and y-axis of the vehicle. The second output port of the model stores values for debugging the model and can be customized as required. At the moment it outputs slip angles and velocity vector angles.

Finally Figure 16 displays the blocks that calculates the gravitational force acting on each wheel. The gravitational force acting on the body is located in the subsystem that sums forces on the RVM's linear axes.

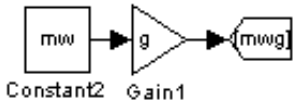

Figure 16: Simulink gravitational force

3.6.5 Forces Summation

After all of the external forces have been calculated they need to be summed together around the vehicle. This is done with the large vertical mux block in the middle of Figure 9. As can be seen, at the moment the forces are only applied directly to the ten degrees of freedom that contain the variables that are being solved for. This is not the ideal way of applying these forces but is a method that works.

The mux block represents the six degrees of freedom for the main vehicle body and each of the four wheels. Therefore the current configuration applies external forces on the Z-axis component of each wheel and on all six degrees of freedom for the main body. The ideal way of doing this would be to apply forces that act on the wheels directly at the wheels. (This is referring to the input port of the mux that represent the linear axes of the wheels.) Unfortunately, when this method was first attempted it resulted in unexpected behavior of the road vehicle and has not been attempted again since. However, simple experimentation since that time has shown that the forces are able to be applied to the model using that method, and the unexpected behaviour observed earlier was because in the governing equations the linear forces are not applied through the centre of gravity of the vehicle as originally assumed.

The reasoning behind applying the wheel forces directly to the main body is centered around the fact that they are only needed to determine the moments acting on the vehicle body and do not affect the relative motion between the wheels and the vehicle body. Thus along the x and y-axes of the vehicle, the body and wheels are considered to be a rigid body.

The outputs of all of the external forces models are all linear forces in the local coordinate system of the

vehicle body. This means that the conventions in Figure 1 are followed and the x-axis is the longitudinal axis of the vehicle, the y-axis is the lateral axis and the z-axis is the vertical axis. However, one is not interested in the changes of vehicle accelerations in this coordinate system. Therefore, before the external forces can be inputed into the external forces vector mux they must be transformed into the global coordinate system.

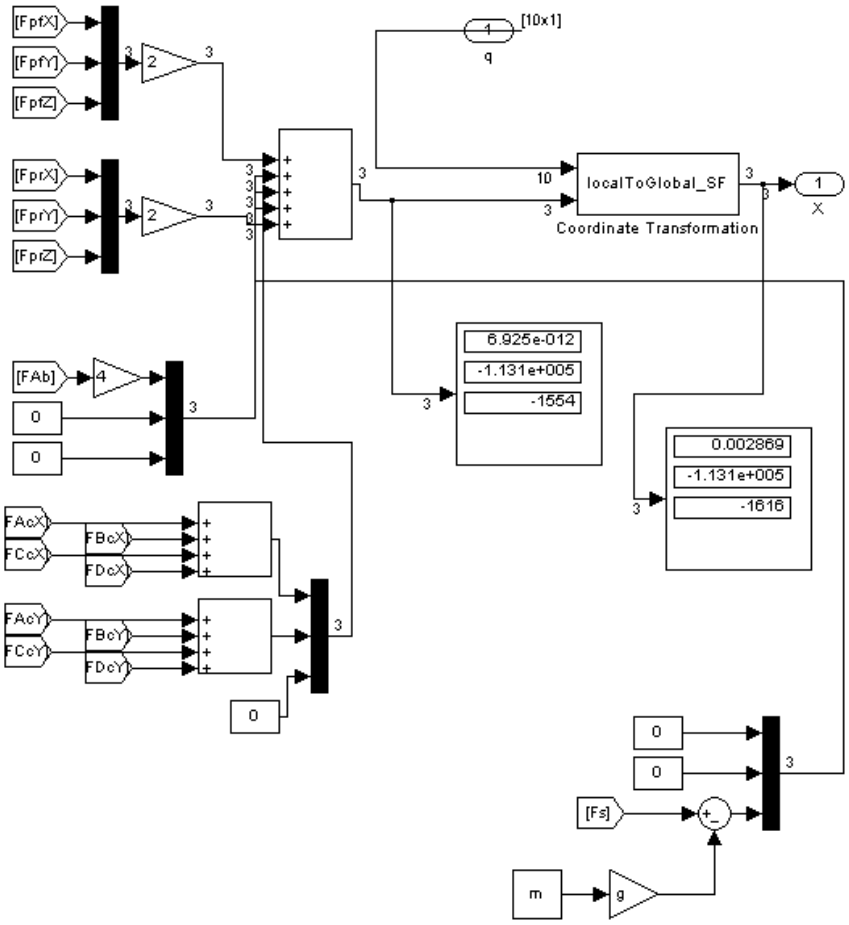

Figure 17: Linear forces summation and coordinate transformation

Figure 17 shows the force summations along the X, Y, and Z axes on the main vehicle body in the global coordinate system. The forces are first added together in the local coordinate system. They are then

transformed into the global coordinate system before being outputted. The display sinks allow the user to monitor the resulting force summation and the effect of the coordinate transformation. One final feature of this subsystem is that it is the location where the gravitational force of the main vehicle body is calculated since it only contributes to vehicle acceleration in one degree of freedom. (For now the moment force acting on the vehicle due to gravity is neglected as the vehicle is symmetrical).

The linear force summations on the four wheels are performed directly in front of the mux. Since the resulting accelerations of interest are the relative accelerations between the vehicle body and wheels these are not transformed into the global coordinate system.

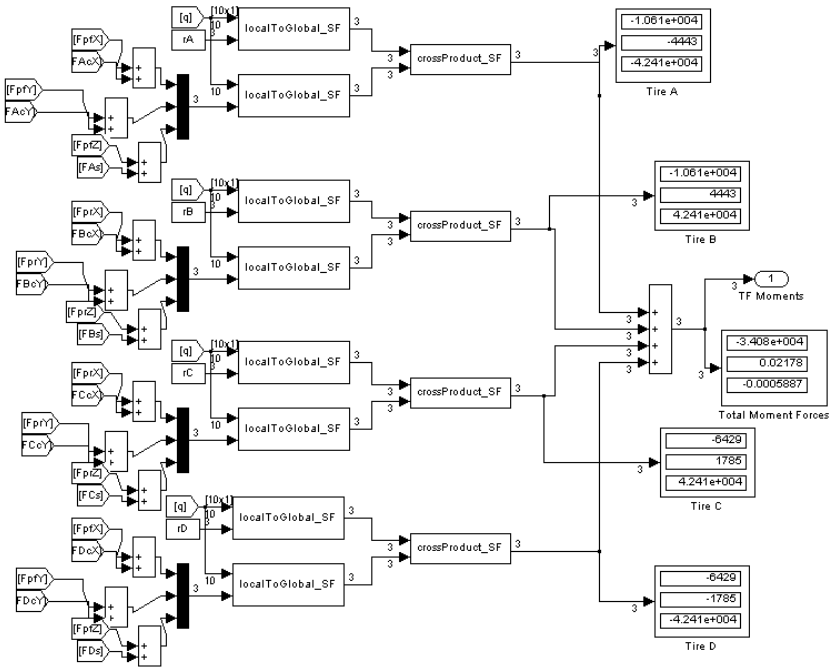

Figure 18: Angular forces summations with coordinate transformations

Once the forces have been summed together along the coordinate axis it is necessary to sum their effects on the angular accelerations of the vehicle. Figure 18 displays the subsystem that performs this summation. This moment summation is much more complicated because it is required to perform four linear force

summations, one for each wheel, determine the resulting moments around the centre of gravity of the vehicle from each wheel, and then sum all of those moments together. It would be an accurate statement to say that at least ninety percent of the problems with the road vehicle in this project have been because of a mistake in an external forces model that contributes to this moments summation. The display sinks in this subsystem are positioned to allow the resulting moment contribution from each wheel to be monitored as well as the final summation of these moments.

3.6.6 Governing Equations

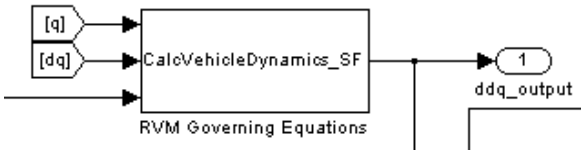

Figure 19: Governing equations S-Function

Once the forces acting on the road vehicle are added together, they are passed to the S-Function $CalcVehicleDynamics\_SF$, which contains the governing equations of the road vehicle model. This subsystem is shown in Figure 19. The outputs are the RVM accelerations in ten degrees of freedom. These accelerations are then passed to an integration subsystem which integrates the accelerations with respect to time in order to determine the vehicle velocities and positions. Figure 20 displays this subsystem.

This subsystem allows the initial conditions of the vehicle velocities and positions to be set. At the moment modifying these values is the only way to set the initial conditions for the code that will be generated for the road vehicle model (which is discussed in the next section). The positions of the road vehicle model are passed out of the road vehicle subsystem so that they can be sent to the virtual reality visualization sink.

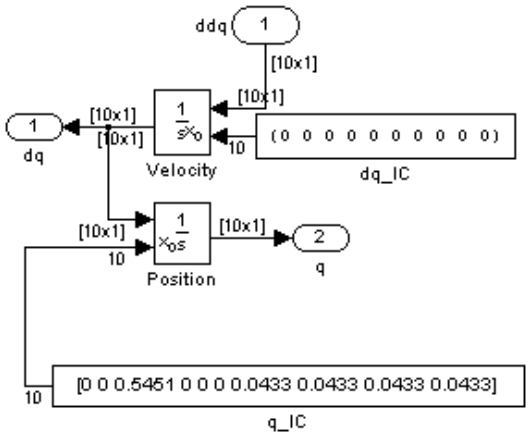

Figure 20: Time-integration subsystem

3.7 Real-Time Workshop Code Generation

There are many ways to use Real-Time Workshop (RTW) to generate code for the road vehicle model. The process documented here is the only one that succeeded during this project's undertaking, and involves using the generic real-time target in RTW to generate a Microsoft Visual Studio makefile. By default that makefile can be used to create an executable program that runs the Simulink block program, but in this case a process has been developed to modify the code so that it will produce a DLL instead.

An important detail about the RVM DLL is that the latest implementation of the DLL uses Simulink's built-in time-integration methods. This means that not only does the RVM DLL output vehicle accelerations as planned, but also updated velocities and positions. This means that even though a time-integration procedure was developed, it was not implemented in the latest RVM DLL code and thus is not documented here. The disadvantage of this is that in order to change the integration method or time step, one must change it in Simulink before using Real-Time Workshop to export the RVM to code.

Also important to note is that one of the design requirements of RVM-RTW code generation process was not met. This requirement was to not need to modify any of the code automatically generated by RTW.

Unfortunately, the current process requires seven lines of code to be added to one of the generated files. These seven lines consist of one additional header inclusion and two functions which allow the Simulink program inputs and outputs to be accessed with pointers. The reason these functions are added to the generated file are because at the moment no other way has been found to work that does not generate an error because of different conventions between C and C++.

After the RVM DLL has been generated it can be used in any other C++ program by calling just three functions. The first is *rvm_initialize*(), the second is *rvm_onestep*() and the third is *rvm_terminate*(). The first and third functions must be called at the beginning and end of a simulation, respectively. The *rvm_onestep*() is used whenever one desires to advance the RVM by one time integration step. At the moment the RVM is configured to operate with a time step of 0.01 s, or 100 Hz. This implementation of the RVM code does not return the absolute elapsed time, so if one wishes to keep track of it, it must be done externally.

The major advantage of using Real-Time Workshop to generate code from the Simulink program instead of compiling the code manually is that the model in Simulink has been verified. This means that the mathematical equations that were incorporated into the various Simulink components are behaving as intended. (This is different from validation, which would determine the accuracy of the modelling compared to reality.) Therefore as long as there are no complications in the code generation process, the road vehicle code is guaranteed to work if the Simulink model works.

4 Simulation Performance

At the end of the project the road vehicle model behaved as expected both within the Simulink program and by using the generated C++ library in an external command-line program. The Simulink model only operated at a speed of two-thirds of real time, but remained controllable through the use of a joystick or steering wheel. The reason the simulation did not run in real time was because the visualization and

modelling mathematics were both performed on the same computer. The code generated from the Simulink model performed the modelling mathematics faster than real time. Therefore if in the future the Simulink model is needed to run in real-time the visualization could be processed on a second computer.

The road vehicle model attains its optimum performance at 100 Hz. This corresponds to a time-step of 0.01s between acceleration calculation iterations. At lower frequencies the vehicle model becomes unstable and the mathematics diverges. At higher frequencies the vehicle model takes additional time to calculate each time step and thus will eventually operate slower than real time. Performing faster than real time is preferred since it allows the complexity of the model to be increased without sacrificing performance.

5 Conclusion

This report outlines the development of a road vehicle mathematical model and a software environment used to verify the model. Six external forces models were incorporated into the road vehicle model and complex models were developed to simulate tire compression and cornering forces. All of the external forces were either programmed in C++ code or directly programmed in Simulink. In Simulink, a modular development environment was created to allow the model to be easily verified to provide a simple interface for expanding the simulation in the future. Finally, using Real-Time Workshop in Simulink, a process was developed to incorporate automatically-generated C++ code into a dynamic link library that can be incorporated into other software simulation environments. The project was a success since the vehicle's behaviour was verified within the Simulink simulation and by using the generated C++ library in an external software environment.