ABSTRACT

Title of Document: Virtual Reality Simulation of a Car Suspension with Active Control Capability

Jason James Smoker
Master of Science 2009

Directed By: Professor Amr M. Baz
Department of Mechanical Engineering

This thesis presents the evolution of a full car model into virtual reality environment to visually demonstrate the dynamics of a car resulting from various inputs controlled both passively and actively. The model is a seven degree of freedom system that can be configured to be excited by either a bump or harmonic input. Active controls available to the system include the well known Linear Quadratic Regulator (LQR) as well as a new Nonlinear Energy Absorber (NEA) which utilizes both nonlinear springs and nonlinear damper. The mathematics of the plant, the kinematics of the system, and the visual specifications of the scene are integrated into a three-dimensional environment where the user can be immersed in the environment and witness in real-time the response of a specific configuration. This project was developed with the mindset that dynamic models of systems can be better understood through visual realization and interaction.

VIRTUAL REALITY SIMULATION OF A CAR SUSPENSION WITH ACTIVE CONTROL CAPABILITY

By

Jason James Smoker

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2009

Advisory Committee:
Professor Amr M. Baz, Chair and Advisor
Professor Balakumar Balachandran
Assistant Professor Nikhil Chopra

Table of Contents

List of Tables

List of Figures

1 Introduction

The motivation of this thesis is to create a configurable and comprehensive simulation of a car suspension system with a well known and a novel active control. The virtual nature of the model allows for greater range of options than a physical model would. This includes the ability to instantaneously change parameters and observance of physical reactions that would normally be mechanically or monetarily intensive to configure. The virtual environment brings flexibility into demonstrations as well as repeatability. Akhtar[1] and Kim et al.[2] showed that virtual simulations created an environment that facilitated design and promoted understanding of a physical system respectively.

Due to the nature of the project, comparisons are also an important aspect of the environment. The user is allowed to run a simulation with one set of parameters, then immediately change the parameters to run it again, and compare the results of the simulation. This allows for a demonstration of how changes in certain parameters effect changes in response and how an optimal set of parameters for one input is not the optimal set for another.

The name given to the virtual environment used in this realization is recursive acronym CAVE, which stands for the CAVE Automatic Virtual Environment. The CAVE used is a collection of audio and visual equipment and position sensors that are connected to an Onyx2 computer. The Onyx2 processes information and outputs to a room with three walls and a floor that are used as video screens. This allows the user to have the feeling of immersions in the virtual environment. The user also has a joystick-like device called a “wand” to navigate through the virtual environment and interact with it.

The software used to program the three dimensional environment is written in C++ utilizing OpenGL Performer graphical libraries. Three dimensional objects in environment can either be created from points, line, and polygons using the graphical libraries, or imported from files created with certain three dimensional modeling software. The following pages detail a brief background of virtual reality technologies, the creation and integration of the dynamics and kinematics of the model, and the simulation results.

1.1: Introduction to Virtual Reality

Depending on the point of view, virtual reality has been described in many ways such as some who choose to define it from by the tools that are used. But this can be limiting since many input and output devices can be interchangeable, especially in an ever developing field. In terms of functionality, however, virtual reality can be defined as an interactive environment designed around a graphical interface. Burdea and Coiffet[3] encompass the components of virtual reality as the "Three 'I's of Virtual Reality" specifically immersion, interaction, and imagination.

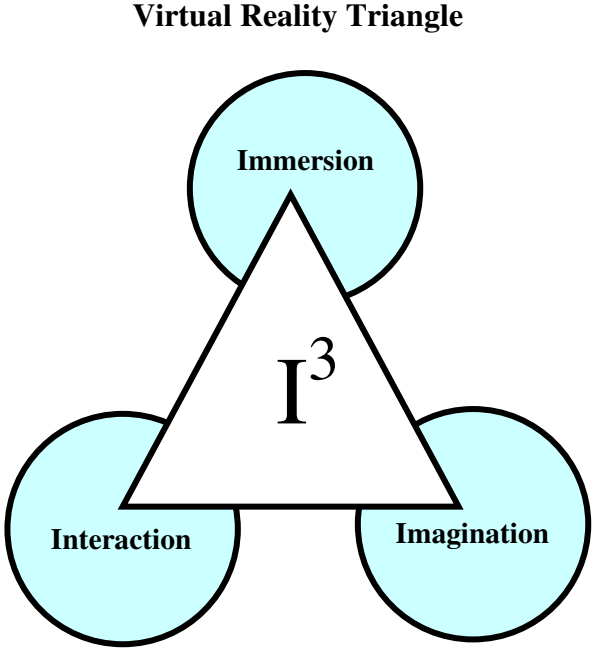

Figure 1: Components of Virtual Reality (Burdea and Coiffet[3])

Sherman and Craig[4] also describe these same three components, while renaming "Imagination" to "Virtual World" but also delineate a fourth component i.e. "Sensory Feedback".

The first hardware for Virtual Reality can be traced back to 1962 with the patent of the "Sensorama Simulator" which used analog images taken by a pair of 35 mm

cameras side by side. In 1968, Ivan Sutherland describes the development for one of the first Head Mounted Displays (HMD) designed for virtual reality. By the end of the decade, the graphical interface evolved to digital graphics reaching 200 polygons per scene. Through the 70s and 80s, the military and NASA took the lead with virtual reality technology. For the military, portability of the software was a key factor for the flight simulators, since new planes were being constantly developed. NASA's interest came from the need for simulations of extra terrestrial conditions. In 1984 NASA hired Scott Fischer to create the Virtual Interface Environment Workstation (VIEW) lab through a cooperative effort with several other companies including Fakespace, Inc., the supplier of the original CAVE lab at the University of Maryland. By the late 1980s computers began to be powerful enough to replace the wireframe images with flat shaded surfaces. By 1989, Autodesk announced their *CyberSpace* project which allowed 3D world creation on personal computers. In 1992, the SIGGRAPH computer graphics conference first débuted the CAVE developed by the Electronic Visualization Lab at the University of Illinois. 1995 ushered the first sub $1,000 HMD which includes head tracking developed by Virtual I/O. By 2007, developers were taking advantage of readily available computation power tracing particles. Scirski et al.[5] developed hybrid systems for use with particle tracing. That same year Waldner et al.[6] developed a system allowing multiple users to have independent perspectives simultaneously using polarized light. Figure 2 shows the increase of the VR market since 1993.

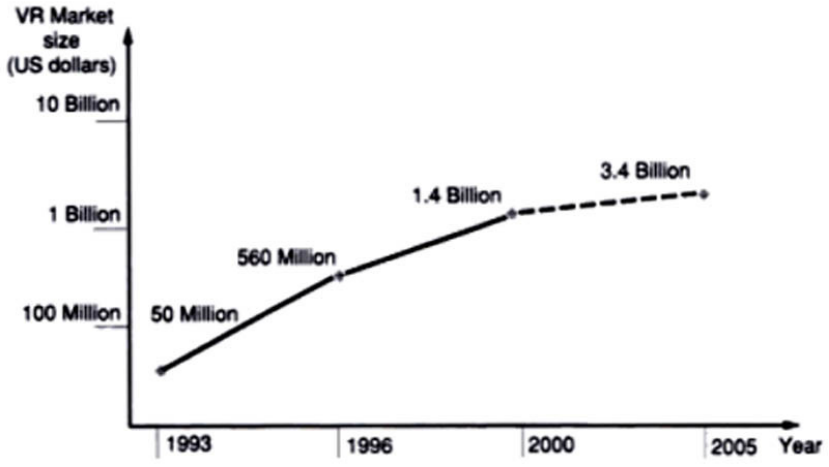

Figure 2: Size of VR Market (Burdea and Coiffet[7])

Virtual reality technology has evolved today to such a point that multiple variations of interaction have involved. A classification system was introduced by Brill[8] then adapted by Biocca and Levy[9]. The adapted a classification broke the interactions into 6 variations:

Table 1: Classification of Systems (Biocca and Levy[5])

Types	*Description*[2]
Window systems	A computer screen provides a window or portal onto an interactive, 3-D virtual world. Desktop computers are often used and users sometimes wear 3-D glasses for stereoscopic effects.
Mirror systems	The users look at a projection screen and see an image of themselves moving in a virtual world. Video equipment is used to record the user's body. A computer superimposes a cut-out image on a computer graphic background. The cut-out images of themselves on the screen mirrors their movements, hence the name *mirror systems*.
Vehicle-based systems	The users enter what appears to be vehicle (e.g., tank, plane, car, space ship, etc.) and operate controls that simulate movement in the virtual world. The world is most often projected on screens. The vehicles may include motion platforms to simulate physical movement.
Cave systems	Users enter a room or enclosure where they are surrounded by large screens that project a nearly continuous virtual scene. 3-D glasses are sometimes used to enhance the sense of space.
Immersive virtual reality systems	Users wear displays that fully immerse a number of the senses in computer generated stimuli. The stereoscopic head-mounted displays (HMD) are a distinctive feature of such systems.
Augmented reality systems	Users wear a visual display (e.g., transmissive HMD) that superimposes 3-D virtual objects on real-world scenes.

One can see that the fourth item on Table 1, is the system utilized for this project. Details of the CAVE layout and hardware as it applied to the University of Maryland Mechanical Engineering CAVE can be found in Appendix A: CAVE Lab and operation of said CAVE can be found in Appendix B: Operation of the CAVE.

1.2: Introduction to Car Suspension Modeling with Control

Many variations of the car model have been utilized depending on the focus of the application. Quarter car models are often used in educational material such as Maxwell[10], Garner[11], and Rao[12]. These tend to be 1 or 2 degree of freedom (DOF) systems. Others, still, use it as a baseline when exploring new control systems such as Vakakis[13] or for reducing computation costs such as Zheng et al.[14]

Yao[15] uses a 4 DOF half car model as a starting point to demonstrate an egomotion estimation algorithm for use with autonomous navigation as does Taghirad and Esmailzadeh[16] to design a controller that minimizes driver and passenger acceleration. Yoshimura et al.[17] use a 6 DOF half car model to demonstrate an active suspension using dynamic absorbers.

Full car models usually occur with 7 or 10 DOF. The 7 DOF model is usually sufficient for cases without forward or lateral acceleration, in other words, straight and level driving with constant velocity. Zheng et al.[18,19], Lu[20], Yao[10], and Kruzek and Stribrsky[21] all use 7 DOF models for their respective papers on nonlinear energy sinks, multi-objective H_2 and H_∞ control, an algorithm for autonomous navigation and an analysis on roll and pitch due to breaking and cornering, an exception to the aforementioned cases. Kruzek and Stribrsky[21] take an additional step and model effect of the ride on the passenger. Cases that take into account longitudinal and lateral accelerations are usually are dealt with 10 DOF systems. Kim and Ro[22] simplify that common 10 DOF system by categorizing the degrees of freedom into those that most affect the ride and those that most affect handling then, using reducing a technique, simplify the system to 7 DOF. As an example of the preceding models, Appendix C overviews preliminary programs created using quarter car and half car models.

The type of car modeled will be dependent on the parameters associated with the car. A car with a center of gravity close to the front and a loose suspension system will could correspond to an empty truck. A vehicle with a center of gravity closer to the center and a tight suspension system could correspond to a sportscar. Allowing these parameters to be chosen by the user as in this model will allow flexibility in the design process.

Many of these papers many of these papers focus on control designs and implementation of specific types of active suspension systems discusses later in this paper. However, though out of the scope of this paper, these, too, can be modeled independently of the car such as in Groom and Schaffner.[23,24]

2 Mathematical Modeling of the Dynamics of a Full Car

The model described below is a seven degree of freedom system including heave x, pitch θ, roll φ and wheel displacements w_1, w_2, w_3, w_4. Each of the four wheels is designated with its own input i.e. y_1, y_2, y_3 and y_4. All spring constants k between the wheels and car frame are equal to each other as are the damper constants c. Tire is modeled with a high spring constants are denoted by k_t and negligible damping. The l_1, l_2, l_3, and l_4 lengths are relative to the center of gravity. The system diagram and equations are shown below[12,15,25].

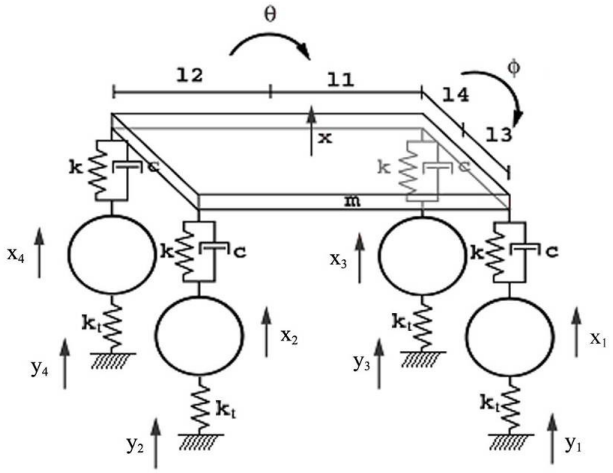

Figure 3: Passive Full Car Model

The controller for the car will be added between the chassis and the each of the wheels as shown below.

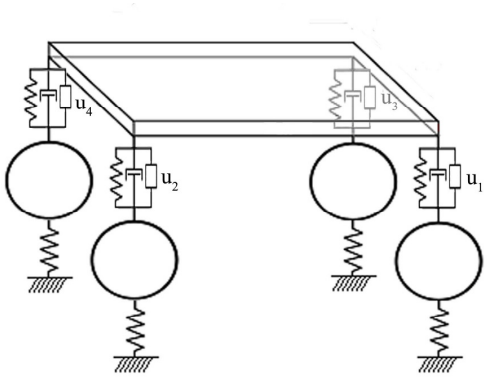

Figure 4: Control Input

2.1: Governing Equations and Series of Equations Transformation

For completeness, control inputs, denoted by u_1, u_2, u_3, and u_4, are added between the car wheels and car body. The resulting governing equations of the car body heave, pitch, roll, and wheel displacement is as follows.

$$M\ddot{X}+k_1(X-l_1\theta-l_3\phi-x_1)+b_1(\dot{X}-l_1\dot{\theta}-l_3\dot{\phi}-\dot{x}_1)+k_2(X+l_2\theta-l_3\phi-x_2)+b_2(\dot{X}+l_2\dot{\theta}-l_3\dot{\phi}-\dot{x}_2) \tag{2.1}$$
$$+k_3(X-l_1\theta+l_4\phi-x_3)+b_1(\dot{X}-l_1\dot{\theta}+l_4\dot{\phi}-\dot{x}_3)+k_4(X+l_2\theta+l_4\phi-x_4)+b_4(\dot{X}+l_2\dot{\theta}+l_4\dot{\phi}-\dot{x}_4)=u_1+u_2+u_3+u_4$$

$$J_{c1}\ddot{\theta}-k_1(X-l_1\theta-l_3\phi-x_1)l_1-b_1(\dot{X}-l_1\dot{\theta}-l_3\dot{\phi}-\dot{x}_1)l_1+k_2(X+l_2\theta-l_3\phi-x_2)l_2+b_2(\dot{X}+l_2\dot{\theta}-l_3\dot{\phi}-\dot{x}_2)l_2 \tag{2.2}$$
$$-k_3(X-l_1\theta+l_4\phi-x_3)l_1-b_3(\dot{X}-l_1\dot{\theta}+l_4\dot{\phi}-\dot{x}_3)l_1+k_4(x+l_2\theta+l_4\phi-x_4)l_2+b_4(\dot{X}+l_2\dot{\theta}+l_4\dot{\phi}-\dot{x}_4)l_2=-l_1(u_1+u_3)+l_2(u_2+u_4)$$

$$J_{c2}\ddot{\phi}-k_1(X-l_1\theta-l_3\phi-x_1)l_3-b_1(\dot{X}-l_1\dot{\theta}-l_3\dot{\phi}-\dot{x}_1)l_3-k_2(X+l_2\theta-l_3\phi-x_2)l_3-b_2(\dot{X}+l_2\dot{\theta}-l_3\dot{\phi}-\dot{x}_2)l_3 \tag{2.3}$$
$$+k_3(X-l_1\theta+l_4\phi-x_3)l_4+b_3(\dot{X}-l_1\dot{\theta}+l_4\dot{\phi}-\dot{x}_3)l_4+k_4(X+l_2\theta+l_4\phi-x_4)l_4+b_4(\dot{X}+l_2\dot{\theta}+l_4\dot{\phi}-\dot{x}_4)l_4=-l_3(u_1+u_2)+l_4(u_3+u_4)$$

$$m_1\ddot{x}_1-k_1(X-\theta l_1-\phi l_3-x_1)-c_1(\dot{X}-\dot{\theta}l_1-\dot{\phi}l_3-\dot{x}_1)=k_t(y_1-x_1)-u_1 \tag{2.4}$$

$$m_2\ddot{x}_2-k_2(X+\theta l_2-\phi l_3-x_2)-c_2(\dot{X}+\dot{\theta}l_2-\dot{\phi}l_3-\dot{x}_1)=k_t(y_2-x_2)-u_2 \tag{2.5}$$

$$m_3\ddot{x}_3-k_3(X-\theta l_1+\phi l_4-x_3)-c_3(\dot{X}-\dot{\theta}l_1+\dot{\phi}l_4-\dot{x}_3)=k_t(y_3-x_3)-u_3 \tag{2.6}$$

$$m_4\ddot{x}_4-k_4(X+\theta l_2+\phi l_4-x_4)-c_4(\dot{X}+\dot{\theta}l_2+\phi l_4-\dot{x}_4)=k_t(y_4-x_4)-u_4 \tag{2.7}$$

Isolating the second derivative results in the following equations:

$$\begin{gathered} M\ddot{X} = -(k_1+k_2+k_3+k_4)X - (b_1+b_2+b_3+b_4)\dot{X} \\ +\left(k_1l_1 - k_2l_2 + k_3l_1 - k_4l_2\right)\theta + \left(b_1l_1 - b_2l_2 + b_3l_1 - b_4l_2\right)\dot{\theta} \\ +\left(k_1l_3 + k_2l_4 - k_3l_3 - k_4l_4\right)\phi + \left(b_1l_3 + b_2l_4 - b_3l_3 - b_4l_4\right)\dot{\phi} \\ +k_1w_1 + c_1\dot{w}_1 + k_2w_2 + c_2\dot{w}_2 + k_3w_3 + c_3\dot{w}_3 + k_4w_4 + c_4\dot{w}_4 \\ +u_1 + u_2 + u_3 + u_4 \end{gathered} \tag{2.8}$$

$$\begin{gathered} J_c\ddot{\theta} = \left((k_1+k_3)l_1 - (k_2+k_4)l_2\right)X + \left((b_1+b_3)l_1 - (b_2+b_4)l_2\right)\dot{X} \\ -\left((k_1+k_3)l_1^2 + (k_2+k_4)l_2^2\right)\theta - \left((b_1+b_3)l_1^2 + (b_2+b_4)l_2^2\right)\dot{\theta} \\ -\left(k_1l_1l_3 - k_2l_2l_3 - k_3l_1l_4 + k_4l_2l_4\right)\phi - \left(b_1l_1l_3 - b_2l_2l_3 - b_3l_1l_4 + b_4l_2l_4\right)\dot{\phi} \\ -k_1w_1l_1 - c_1\dot{w}_1l_1 + k_2w_2l_2 + c_2\dot{w}_2l_2 - k_3w_3l_1 - c_3\dot{w}_3l_1 + k_4w_4l_2 + c_4\dot{w}_4l_2 \\ -l_1(u_1+u_3) + l_2(u_2+u_4) \end{gathered} \tag{2.9}$$

$$\begin{gathered} J_c\ddot{\phi} = \left((k_1+k_2)l_3 - (k_3+k_4)l_4\right)X + \left((b_1+b_2)l_1 - (b_3+b_4)l_4\right)\dot{X} \\ -\left(k_1l_1l_3 - k_2l_2l_3 - k_3l_1l_4 + k_4l_2l_4\right)\theta - \left(b_1l_1l_3 - b_2l_2l_3 - b_3l_1l_4 + b_4l_2l_4\right)\dot{\theta} \\ -\left((k_1+k_2)l_3^2 + (k_3+k_4)l_4^2\right)\phi - \left((b_1+b_2)l_3^2 + (b_3+b_4)l_4^2\right)\dot{\phi} \\ -k_1x_1l_3 - c_1\dot{x}_1l_3 - k_2x_2l_3 - c_2\dot{x}_2l_3 + k_3x_3l_4 + c_3\dot{x}_3l_4 + k_4x_4l_4 + c_4\dot{x}_4l_4 \\ -l_3(u_1+u_2) + l_4(u_3+u_4) \end{gathered} \tag{2.10}$$

$$m_1\ddot{x}_1 = k_t(y_1 - x_1) + k_1(X - \theta l_1 - \phi l_3 - x_1) + c_1(\dot{X} - \dot{\theta}l_1 - \dot{\phi}l_3 - \dot{x}_1) - u_1 \tag{2.11}$$

$$m_2\ddot{x}_2 = k_t(y_2 - x_2) + k_2(X + \theta l_2 - \phi l_3 - x_2) + c_2(\dot{X} + \dot{\theta}l_2 - \dot{\phi}l_3 - \dot{x}_2) - u_2 \tag{2.12}$$

$$m_3\ddot{x}_3 = k_t(y_3 - x_3) + k_3(X - \theta l_1 + \phi l_4 - x_3) + c_3(\dot{X} - \dot{\theta}l_1 + \dot{\phi}l_4 - \dot{x}_3) - u_3 \tag{2.13}$$

$$m_4\ddot{x}_4 = k_t(y_4 - x_4) + k_4(X + \theta l_2 + \phi l_4 - x_4) + c_4(\dot{X} + \dot{\theta}l_2 + \phi l_4 - \dot{x}_4) - u_4 \tag{2.14}$$

Transforming the second order equations into a series of first order equations results in the following.

$$\eta_1 = X \qquad \dot{\eta}_1 = x_2$$

$$\eta_2 = \dot{X} \qquad \dot{\eta}_2 = \frac{1}{M}\begin{bmatrix} -(k_1+k_2+k_3+k_4)X-(b_1+b_2+b_3+b_4)\dot{X} \\ +\left(k_1l_1-k_2l_2+k_3l_1-k_4l_2\right)\theta+\left(b_1l_1-b_2l_2+b_3l_1-b_4l_2\right)\dot{\theta} \\ +\left(k_1l_3+k_2l_3-k_3l_4-k_4l_4\right)\phi+\left(b_1l_3+b_2l_3-b_3l_4-b_4l_4\right)\dot{\phi} \\ +k_1x_1+c_1\dot{x}_1+k_2x_2+c_2\dot{x}_2+k_3x_3+c_3\dot{x}_3+k_4x_4+c_4\dot{x}_4 \\ +u_1+u_2+u_3+u_4 \end{bmatrix}$$

$$\eta_3 = \theta \qquad \dot{\eta}_3 = x_4$$

$$\eta_4 = \dot{\theta} \qquad \dot{\eta}_4 = \frac{1}{J_{c1}}\begin{bmatrix} \left((k_1+k_3)l_1-(k_2+k_4)l_2\right)X+\left((b_1+b_3)l_1-(b_2+b_4)l_2\right)\dot{X} \\ -\left((k_1+k_3)l_1^2+(k_2+k_4)l_2^2\right)\theta-\left((b_1+b_3)l_1^2+(b_2+b_4)l_2^2\right)\dot{\theta} \\ -\left(k_1l_1l_3-k_2l_2l_3-k_3l_1l_4+k_4l_2l_4\right)\phi-\left(b_1l_1l_3-b_2l_2l_3-b_3l_1l_4+b_4l_2l_4\right)\dot{\phi} \\ -k_1x_1l_1-c_1\dot{x}_1l_1+k_2x_2l_2+c_2\dot{x}_2l_2-k_3x_3l_1-c_3\dot{x}_3l_1+k_4x_4l_2+c_4\dot{x}_4l_2 \\ -l_1(u_1+u_3)+l_2(u_2+u_4) \end{bmatrix}$$

$$\eta_5 = \phi \qquad \dot{\eta}_5 = x_6$$

$$\eta_6 = \dot{\phi} \qquad \dot{\eta}_6 = \frac{1}{J_{c2}}\begin{bmatrix} \left((k_1+k_2)l_3-(k_3+k_4)l_4\right)X+\left((b_1+b_2)l_1-(b_3+b_4)l_4\right)\dot{X} \\ -\left(k_1l_1l_3-k_2l_2l_3-k_3l_1l_4+k_4l_2l_4\right)\theta-\left(b_1l_1l_3-b_2l_2l_3-b_3l_1l_4+b_4l_2l_4\right)\dot{\theta} \\ -\left((k_1+k_2)l_3^2+(k_3+k_4)l_4^2\right)\phi-\left((b_1+b_2)l_3^2+(b_3+b_4)l_4^2\right)\dot{\phi} \\ -k_1x_1l_3-c_1\dot{x}_1l_3-k_2x_2l_3-c_2\dot{x}_2l_3+k_3x_3l_4+c_3\dot{x}_3l_4+k_4x_4l_4+c_4\dot{x}_4l_4 \\ -l_3(u_1+u_2)+l_4(u_3+u_4) \end{bmatrix}$$

so

$$\eta_7 = x_1 \qquad \dot{\eta}_7 = x_8 \tag{2.15}$$

$$\eta_8 = \dot{x}_1 \qquad \dot{\eta}_8 = \frac{1}{m_1}\left(k_1X+c_1\dot{X}-k_1l_1\theta-c_1l_1\dot{\theta}-k_1l_3\phi-c_1l_3\dot{\phi}-(k_1+k_t)x_1-c_1\dot{x}_1+k_ty_1-u_1\right)$$

$$\eta_9 = x_2 \qquad \dot{\eta}_9 = x_{10}$$

$$\eta_{10} = \dot{x}_2 \qquad \dot{\eta}_{10} = \frac{1}{m_2}\left(k_2X+c_2\dot{X}+k_2l_2\theta+c_2l_2\dot{\theta}-k_2l_3\phi-c_2l_3\dot{\phi}-(k_2+k_t)x_2-c_2\dot{x}_2+k_ty_2-u_2\right)$$

$$\eta_{11} = x_3 \qquad \dot{\eta}_{11} = x_{12}$$

$$\eta_{12} = \dot{x}_3 \qquad \dot{\eta}_{12} = \frac{1}{m_3}\left(k_3X+c_3\dot{X}-k_3l_1\theta-c_3l_1\dot{\theta}+k_3l_4\phi+c_3l_4\dot{\phi}-(k_3+k_t)x_3-c_3\dot{x}_3+k_ty_3-u_3\right)$$

$$\eta_{13} = x_4 \qquad \dot{\eta}_{13} = x_{14}$$

$$\eta_{14} = \dot{x}_4 \qquad \dot{\eta}_{14} = \frac{1}{m_4}\left(k_4X+c_4\dot{X}+k_4l_2\theta+c_4l_2\dot{\theta}+k_4l_4\phi+c_4l_4\dot{\phi}-(k_4+k_t)x_4-c_4\dot{x}_4+k_ty_4-u_4\right)$$

The equations from (2.15) can then be used to build the state space form of the system

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{Gy} \tag{2.16}$$

where **A** is the coefficient matrix to the system variables, **B** is a ones matrix corresponding to the control inputs and **G** is a matrix of k_t's relating the ground input to the system.

where the input disturbances for harmonic oscillations can be described as:

$$y_1 = H_H \sin\left(\frac{2\pi Vt}{L}\right) \qquad y_2 = H_H \sin\left(\frac{2\pi Vt}{L} + \sigma_d\right)$$
$$y_3 = H_H \sin\left(\frac{2\pi Vt}{L} + \sigma_l\right) \qquad y_4 = H_H \sin\left(\frac{2\pi Vt}{L} + \sigma_d + \sigma_l\right) \tag{2.17}$$

while a bump excitation can be described by:

$$y_1 = \frac{H_B}{2}\left(1 - \cos\left(\frac{2\pi Vt}{L}\right)\right) \qquad y_2 = \frac{H_B}{2}\left(1 - \cos\left(\frac{2\pi Vt}{L} + \sigma_d\right)\right)$$
$$y_3 = \frac{H_B}{2}\left(1 - \cos\left(\frac{2\pi Vt}{L} + \sigma_l\right)\right) \qquad y_4 = \frac{H_B}{2}\left(1 - \cos\left(\frac{2\pi Vt}{L} + \sigma_d + \sigma_l\right)\right) \tag{2.18}$$

where H_B and H_H are the bump height and harmonic amplitudes respectively, V is the velocity of the vehicle t is the time past, and L is the length of the bump and the length of a period in the harmonic input.

The σ_d and σ_l variables assume that all four wheels travel over the same bump displacement at different times. The σ_d accounts for the interval between the front wheels and the rear wheels, while σ_l accounts for shift between the right set of wheels and the left set of wheels.

The series of equations in (2.15) in conjunction with (2.17) or (2.18), are solved using Runge-Kutta-Fehlberg numerical method. The solution is solved in real-time at approximately 100 solutions per second.

3 Active Control of Car Suspension

Active control of suspension systems are increasingly becoming a common factor in new cars. Commercially available active suspensions for passenger cars can be traced back to 1987 with the Electronically Controlled Suspension in the Mitsubishi Galant.[26] Today, many other car companies have their own version of the active suspension system including Mercedes Benz, Toyota, Infinity, Jaguar, Chevrolet, and Cadillac. For example, Chevrolet and Cadillac utilize a Magneto-Rheological (MR) damper called MagneRide[27] developed by Dephi in a few of their cars including the Cadillac Seville STS, XLR, SRX, and the Chevrolet Corvette.

Active suspensions come in two forms, purely active suspensions which exert an independent force such as hydraulic actuated and electromagnetic recuperative (ER) and semi active suspensions which actively change the damping coefficient of the suspension such as valve actuated and the aforementioned MR damper. These designs add comfort to passengers and improve the road handling abilities of the vehicle. A summary of various techniques can be found in references.[14,28,29,30,31]

A few specific examples of Linear Quadratic Regulator (LQR) and Linear Quadratic Gaussian (LQG) control are demonstrated in Taghirad and Esmailzadeh[16] as well as Sam, Ghani, and Ahmad[32]. In addition, use of Nonlinear Energy Sink (NES) has been proposed by Zheng, Baz, and Li[18,19] stemming from vibration control demonstrated by Vakakis.[13,33]

In the following paragraphs, the process and design of an LQR and a Nonlinear Energy Absorber (NEA) controller will be described. A straightforward system model without noise is implemented, thus all states will be known making an LQG controller unnecessary.

3.1 Linear Quadratic Regulator (LQR) Design

The previously described full car model can be described in the state space form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{G}\mathbf{y} \tag{3.0}$$

where state vector $\mathbf{x}$ is given by

$$\mathbf{x} = \begin{bmatrix} X & \dot{X} & \theta & \dot{\theta} & \phi & \dot{\phi} & x_1 & \dot{x}_1 & x_2 & \dot{x}_2 & x_3 & \dot{x}_3 & x_4 & \dot{x}_4 \end{bmatrix}^T \tag{3.1}$$

road disturbance **w** on each wheel is given by

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T \tag{3.2}$$

and control input **u** representing four actuators is given by

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T . \tag{3.3}$$

The LQR control of this Linear Time-Invariant (LTI) system is to be designed for a Fixed-Final State. Conventionally, determining the control matrix is based on minimization of performance index below where weighting matrices **Q** and **R** are positive semi-definite and positive definite, respectively.

$$J = \int_0^\infty \left(\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \right) dt . \tag{3.4}$$

However, Taghirad and Esmailzadeh[16] showed that an Acceleration Dependant Method (ADM) can be implemented by including passenger accelerations in the form

$$\mathbf{z} = \begin{bmatrix} \ddot{x}_{p1} & \ddot{x}_{p2} \end{bmatrix}^T . \tag{3.5}$$

which can be visualized in Figure 5.

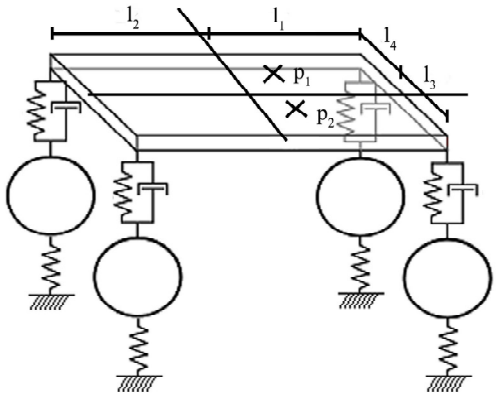

Figure 5: Passenger Locations

For this system, these represent the vertical acceleration of the front two passengers. The corresponding performance integral is

$$J = \int_0^\infty \left(\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} + \mathbf{z}^T \mathbf{S} \mathbf{z} \right) dt \tag{3.6}$$

where the weighting matrix $\mathbf{S}$ is given by

$$\mathbf{S} = \begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix}. \tag{3.7}$$

However, the passenger accelerations are linearly dependant on the state variable and thus (3.6) can simply be represented as

$$J = \int_0^\infty \left(\mathbf{x}^T \mathbf{Q}_n \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \right) dt \tag{3.8}$$

where the new $\mathbf{Q}_n$ is given by

$$\mathbf{Q}_n = \left[\mathbf{Q} + \mathbf{v_1}^T S_1 \mathbf{v_1} + \mathbf{v_2}^T S_2 \mathbf{v_2} \right] \tag{3.9}$$

and linear dependence is described with transformation vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ such that

$$\begin{aligned} \ddot{x}_{p1} &= \mathbf{v_1} \cdot \mathbf{x} \\ \ddot{x}_{p2} &= \mathbf{v_2} \cdot \mathbf{x} \end{aligned} \tag{3.10}$$

where the transformation can be defined by

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{A}_{2i} + \mathbf{A}_{4i} \cdot \left(\frac{l_1 - l_2}{2} - 0.5 \right) + \mathbf{A}_{6i} \cdot 0.5 \\ \mathbf{v_2} &= \mathbf{A}_{2i} + \mathbf{A}_{4i} \cdot \left(\frac{l_1 - l_2}{2} - 0.5 \right) - \mathbf{A}_{6i} \cdot 0.5 \end{aligned} \tag{3.11}$$

The resulting control is represented in state feedback form

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{3.12}$$

where $\mathbf{K}$ is the state feedback gain matrix determined by

$$\mathbf{K} = \mathbf{R}^{-1} \mathbf{B}^{\mathbf{T}} \mathbf{P} \tag{3.13}$$

and $\mathbf{P}$ is the solution of the Algebraic Riccati Equation.

$$0 = \mathbf{A}^{\mathbf{T}}\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}(\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathbf{T}})\mathbf{P} + \mathbf{Q}_n \tag{3.14}$$

3.2 Solutions to the Algebraic Riccati Equation (ARE)

Many papers currently take the solution of the Algebraic Riccati Equation for granted by using by using packaged LQR solvers[16,18,19,24,32] in software such as MATLAB and MATRIXx. In an attempt to call attention to the numerous methods to solve an ARE, this section reviews a few of these methods. It will then end with the method used in this simulation.

The Continuous Algebraic Riccati Equation (CARE) is the more difficult case to solve as opposed to the Discrete Algebraic Riccati Equation (DARE). One of the earlier and still popular methods was developed by James E. Potter.[34] This method involves reorganizing values of the ARE in to a single matrix as follows.

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{Q}_n \\ (\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathbf{T}}) & -\mathbf{A}^{*} \end{bmatrix} \tag{3.15}$$

The eigenvectors of $\mathbf{M}$ are then found and each eigenvector is written in the form

$$\mathbf{a} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix} \tag{3.16}$$

where $\mathbf{b}$ and $\mathbf{c}$ are the upper and lower halves of the eigenvector $\mathbf{a}$ respectively. The solution to the ARE can then be described as

$$\mathbf{P} = \mathbf{b} \bullet \mathbf{c}^{-1} \tag{3.17}$$

Laub[35] later improved on this method using the computation of Schur vectors.

These methods are fairly straightforward; however, it becomes computationally expensive for larger systems with computational effort growing at a rate around n^3.[34]

Two years later, D.L. Kleinman[36] developed an iterative technique which reduced the Riccati equation into a linear algebraic equation similar to the continuous Lyapunov equation form. That same year, R.A. Smith[37] wrote about an iterative solution to solve

the matrix equation $\mathbf{XA} + \mathbf{BX} = \mathbf{C}$ for which the continuous Lyapunov equation is a special case i.e. where $\mathbf{B} = \mathbf{A}^*$. Later, Banks and Ito[38] would combine these methods for an iterative method suitable for large but finite dimensional problems.

For this project, the discrete solution was deemed preferable since it minimized matrix operations to addition, subtraction, multiplication, and finding an inverse and eliminated the need to find eigenvalues of a high order matrix.

The system model and performance index for the discrete system is described by:

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k$$

$$J_i = \frac{1}{2}\mathbf{x}_N^T \mathbf{S}_N \mathbf{x}_N + \frac{1}{2}\sum_{k=i}^{N-1}\left(\mathbf{x}_k^T \mathbf{Q}_{nk} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R}_k \mathbf{u}_k\right) \tag{3.18}$$

where $\mathbf{A}_k$ and $\mathbf{B}_k$ were found using Taylor Series estimation.

$$\mathbf{A}_k = e^{\mathbf{A}T} = \left(\mathbf{A} \cdot T + \mathbf{I}\right) + H.O.T. \approx \left(\mathbf{A} \cdot T + \mathbf{I}\right)$$

$$\mathbf{B}_k = \int_0^T e^{\mathbf{A}\tau} \mathbf{B} d\tau = \left(\mathbf{B} \cdot T\right) + H.O.T. \approx \left(\mathbf{B} \cdot T\right) \tag{3.19}$$

The corresponding solution for the discrete ARE can be found using the following:[39]

$$\mathbf{Q}_{nk} \geq 0 \qquad \mathbf{R}_k > 0 \qquad \mathbf{S}_k \geq 0 \qquad \mathbf{S}_N \text{ given}$$

$$\mathbf{S}_k = \mathbf{A}_k^T \left[\mathbf{S}_{k+1} - \mathbf{S}_{k+1}\mathbf{B}_k \left(\mathbf{B}_k^T \mathbf{S}_{k+1} \mathbf{B}_k + \mathbf{R}_k \right)^{-1} \mathbf{B}_k^T \mathbf{S}_{k+1} \right] \mathbf{A}_k + \mathbf{Q}_{nk} \qquad k < N \tag{3.20}$$

$$\mathbf{K}_k = \left(\mathbf{B}_k^T \mathbf{S}_{k+1} \mathbf{B}_k + \mathbf{R}_k \right)^{-1} \mathbf{B}_k^T \mathbf{S}_{k+1} \mathbf{A}_k \qquad k < N$$

It can be seen that as opposed to the continuous ARE, the solution here requires the extra weighting matrix $\mathbf{S}_N$ as seen above which provides a starting point for the solution of the ARE. For this model, the identity matrix is used.

3.3 Nonlinear Energy Absorber (NEA) Design

The Nonlinear Energy Absorber is an incremental step to that of the Nonlinear Eergy Sink developed by Vakakis[13]. This model redistributes energy to lower the amplitude peaks of the system. Zheng et al. proposed to apply the control to car suspension systems in two ways. In one paper[19], one controller was presented for a half car model. In the other[18], four independent controllers were proposed for a full car model. This section describes the addition of a nonlinear spring as well as the nonlinear damper presented by Vakakis. This allows the system not only to effectively redistribute the energy, also dissipate it as well. Just as with the LQR, all calculations were to be made within the program to streamline the virtual design interface.

Below are three tables that compare the energy dissipation of a Vakakis type system, a linear system, and the new system presented with both nonlinear spring and nonlinear damper. It can be seen that the energy dissipation of the Vakakis system is the same as that of the linear system. However, the new system adds an extra term of dissipation and therefore dissipates more energy than either the Vakakis system or the linear system.

Table 2: Nonlinear Spring (Vakakis type system)

System Equation	$m\ddot{x} + b\dot{x} + kx^3 = 0$
Energy Equation	$E = \frac{1}{2}m\dot{x}^2 + \frac{1}{4}kx^4$
Energy Decay	$\dot{E} = (m\ddot{x} + kx^3)\dot{x} = -b\dot{x}^2 = \dot{E}_v$

Table 3: Linear System

System Equation	$m\ddot{x} + b\dot{x} + kx = 0$
Energy Equation	$E = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}kx^2$
Energy Decay	$\dot{E} = (m\ddot{x} + kx)\dot{x} = -b\dot{x}^2 = \dot{E}_{linear} = \dot{E}_V$

Table 4: Nonlinear Damper and Nonlinear Spring

System Equation	$m\ddot{x} + b\dot{x} + kx^3 + d\dot{x}^3 = 0$
Energy Equation	$E = \frac{1}{2}m\dot{x}^2 + \frac{1}{4}kx^4$
Energy Decay	$\dot{E} = (m\ddot{x} + kx^3)\dot{x} = -b\dot{x}^2 - d\dot{x}^4 = \dot{E}_s$ $\therefore \lvert\dot{E}_s\rvert > \lvert\dot{E}_V\rvert = \lvert\dot{E}_{linear}\rvert$

Since the NEA relies on nonlinear springs and nonlinear dampers, the addition of new state variables are required as shown in (3.20) where iterations $i = 1 \to 4$ correspond to each of the four controllers as mentioned earlier

.

$$\mathbf{g}_i = \left[\left(z_{bi} - x_i \right)^3 \quad \dot{z}_{bi} \quad \left(z_{bi} - x_i \right) \quad \dot{x}_i^3 \right]^T \tag{3.21}$$

where x_i and $\dot{x}_i$ corresponds to the position and velocity of each of the wheels and z_{bi} and $\dot{z}_{bi}$ are the points of contact above the wheels on the chassis defined by (3.22) below.

$$
\begin{aligned}
z_{b1} &= X - l_1\theta - l_3\phi & \dot{z}_{b1} &= \dot{X} - l_1\dot{\theta} - l_3\dot{\phi} \\
z_{b2} &= X + l_2\theta - l_3\phi & \dot{z}_{b2} &= \dot{X} + l_2\dot{\theta} - l_3\dot{\phi} \\
z_{b3} &= X - l_1\theta + l_4\phi & \dot{z}_{b3} &= \dot{X} - l_1\dot{\theta} + l_4\dot{\phi} \\
z_{b4} &= X + l_2\theta + l_4\phi & \dot{z}_{b4} &= \dot{X} + l_2\dot{\theta} + l_4\dot{\phi}
\end{aligned}
\tag{3.22}
$$

One can see in equation (3.20) that the nonlinear stiffness element is $\left(z_{bi} - x_i\right)^3$ and the nonlinear damping is $\dot{x}_i^3$. Thus the feedback input is

$$\mathbf{u}_i = -\mathbf{N}_i\mathbf{g}_i \tag{3.23}$$

where $\mathbf{N}_i$ Is the gain matrix of the NEA. This would give sixteen gains, however, we can assume that the center of gravity will remain constant laterally and reduce the gain to eight by declaring the gains for the two front controllers equal an the two back controllers equal.

The calculation of the gain parameters in $\mathbf{N}_i$ are numerically found by minimization of the cost function given in (3.8). In the papers by Zheng *et al.,* the Matlab function **fminunc** was utilized as the minimization algorithm. Since, in the final version of the program, Matlab was not to be used, one was written for the program (see Appendix D: Main Source Code). A random excitation is created and stored for the minimization. The system is the repeatedly subjected to the excitation while the gain parameters are adjusted to minimize the cost function.

4 Matlab Derived Control Solutions[40]

Though the final version of the simulation was to be stand-alone, the initial calculations for the 7 DOF system were solved using Matlab software. Interaction and passive control parameters could be made in real-time within the virtual environment, but gain matrices were calculated using Matlab software separately. Below are results of the results based on

4.1 Controller Calculations

The performance of the LQR and NEA controllers were evaluated using the parameters from table 2.

Table 5: Vehicle Parameters

Parameter	Unit	Value
M_s	kg	1583
I_{xx}/I_{yy}	$kg\,m^2$	531/2555
$M_{w1}/M_{w2}/M_{w3}/M_{w4}$	kg	48/48/74/74
$K_{s1}/K_{s2}/K_{s3}/K_{s4}$	KN/m	35/35/34/34
$C_{s1}/C_{s2}/C_{s3}/C_{s4}$	Ns/m	400/400/200/200
$K_{t1}/K_{t2}/K_{t3}/K_{t4}$	KN/m	220/220/220/220
l_{xf}/l_{xr}	m	1.116/1.438
$l_{ylf}/l_{yrf}/l_{ylr}/l_{yrr}$	m	0.77/0.77/0.765/0.765

Using an unbounded optimization function in Matlab, feedback parameters of (3.20) corresponding to the minimum energy for the nonlinear energy sink was found to be:

$$N_1 = N_2 = [20000, 2203.5, 205.2, -1500]$$

and

$$N_3 = N_4 = [20000, 2803.5, 805.2, -1500]$$

such that

$$\mathbf{N}_i = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix}^T$$

Using the built in LQR solver in Matlab, the corresponding LQR solution of the feedback gain $\mathbf{K}$ in (3.11) for the same parameters were found to be:

$$k_1 = k_2 = [-1762.6, 846, 789.6, 3.3]$$

$$k_3 = k_4 = [-2716.8, 1078.2, 1486.1, -58.9]$$

such that

$$\mathbf{K} = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix}^T$$

The following graphs in figure 4 show the frequency and time domain comparisons between the LQR and NEA responses.

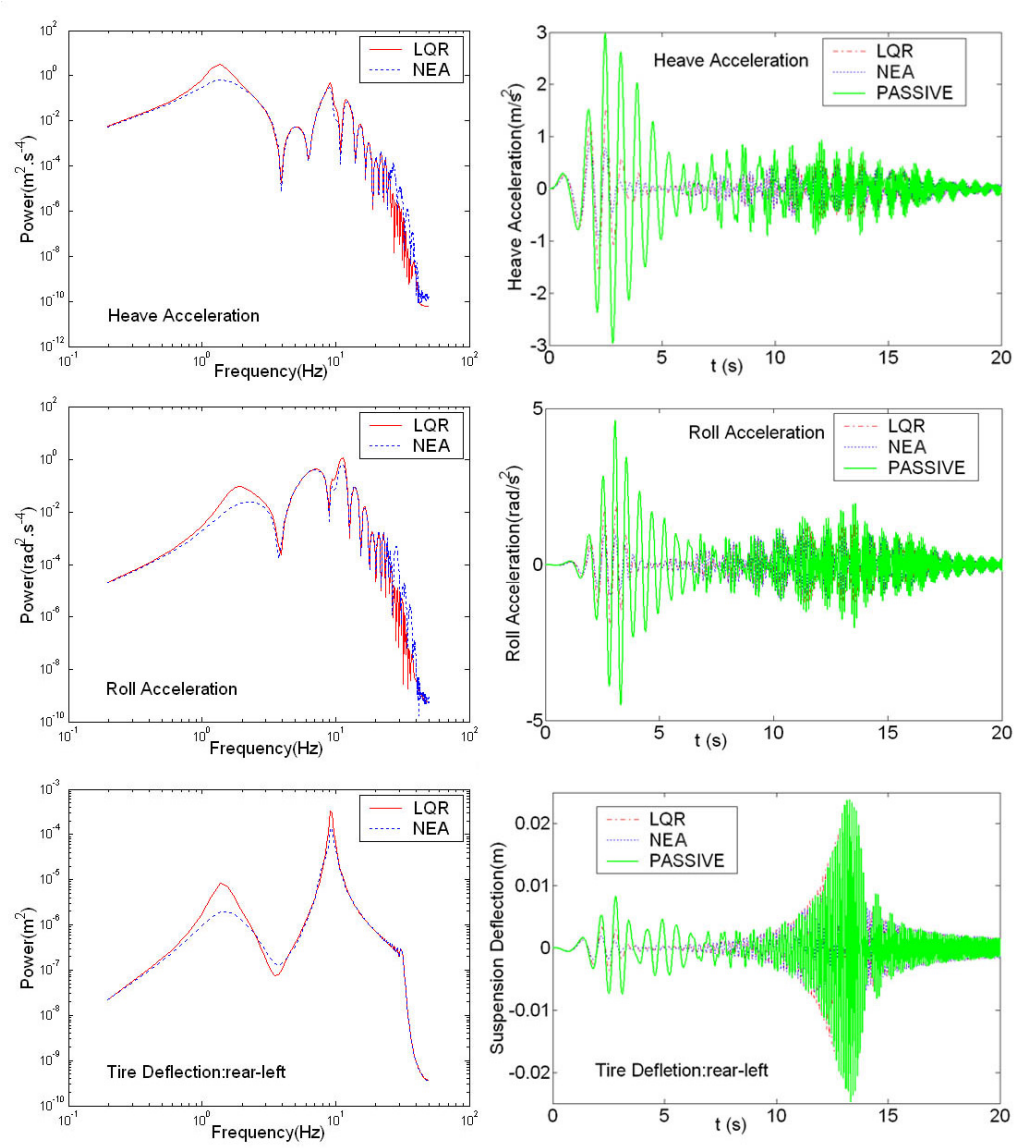

Figure 6: Frequency and Time Domain Graphs

4.2 Implementation in Virtual Reality

The figures above depict the responses for the set of parameters listed in table 2. In figure 5 NEA control has a lower energy but LQR control has a lower maximum.

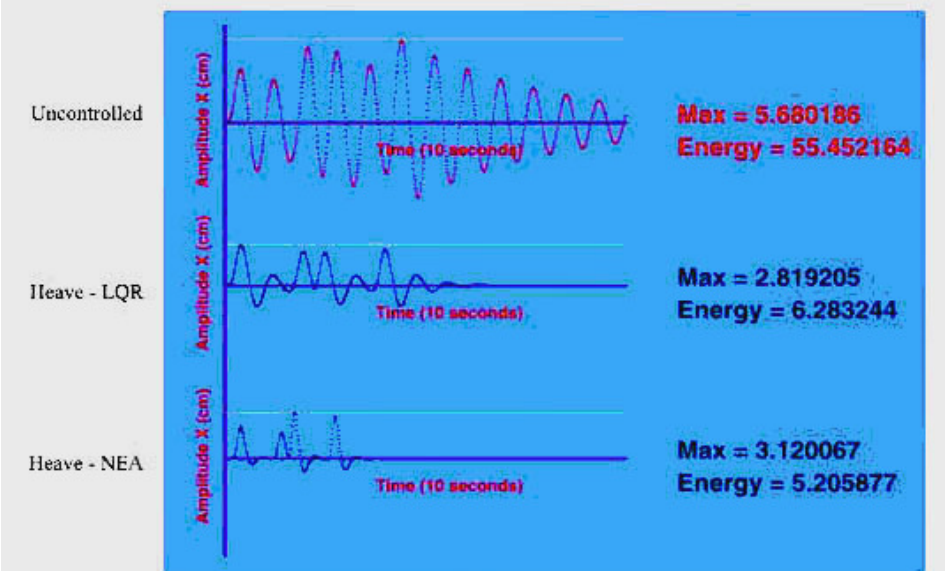

Figure 7: Heave Response

In the roll and pitch responses, NEA control has both lower maximum and energy. The high frequency jittering seen in the uncontrolled graphs of the figures below can be attributed to the limitation of real time integration.

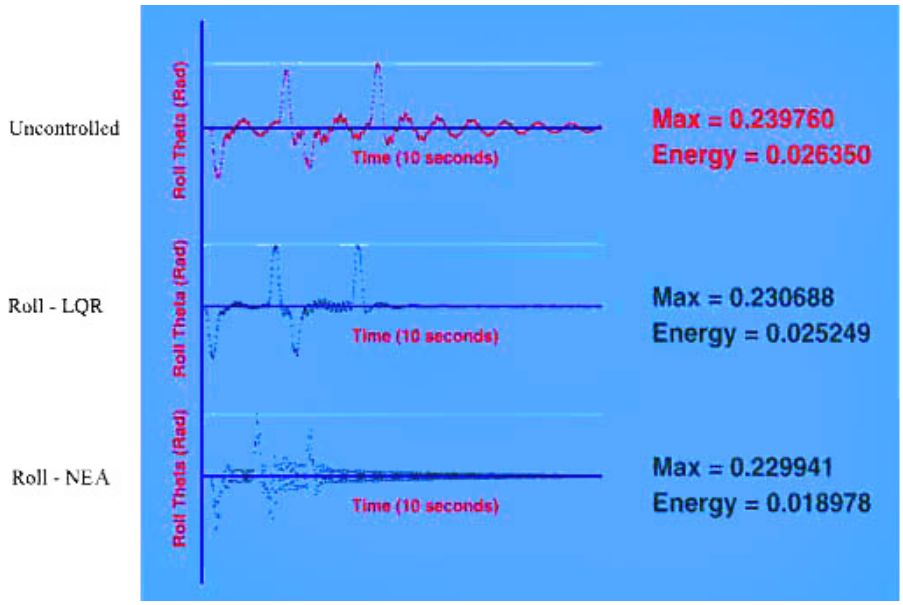

Figure 8: Roll Response

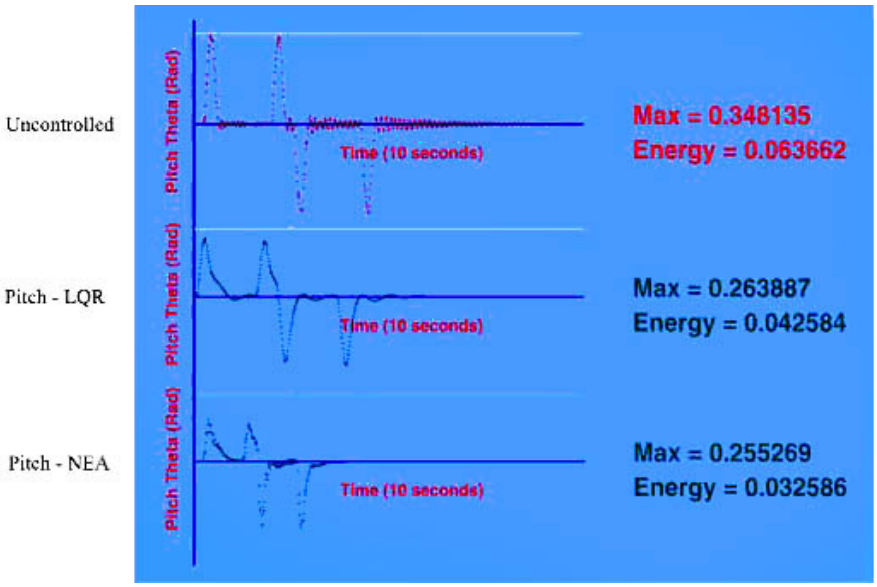

Figure 9: Pitch Response

5 Final Simulation Interface and Results

Having covered the model for the car and the functions to solve for the control gains, the final aspect of the simulation is what the user can experience: input and output of the virtual environment. A thorough interface provides the user with a comprehensive interaction and understanding with the virtual model.

5.1 Simulation Interface

As mentioned in the introduction, one of the major advantages of having a virtual environment is the ability to have virtually instantaneous changes to the model parameters. Figure 4 depicts the adjustable model parameters that can be accessed while in the virtual environment. The hardware that is used to interact with the environment can be found in Appendix A. The software rendered interface used to interact with the program includes a menu and a virtual pointer shown in Figure 10.

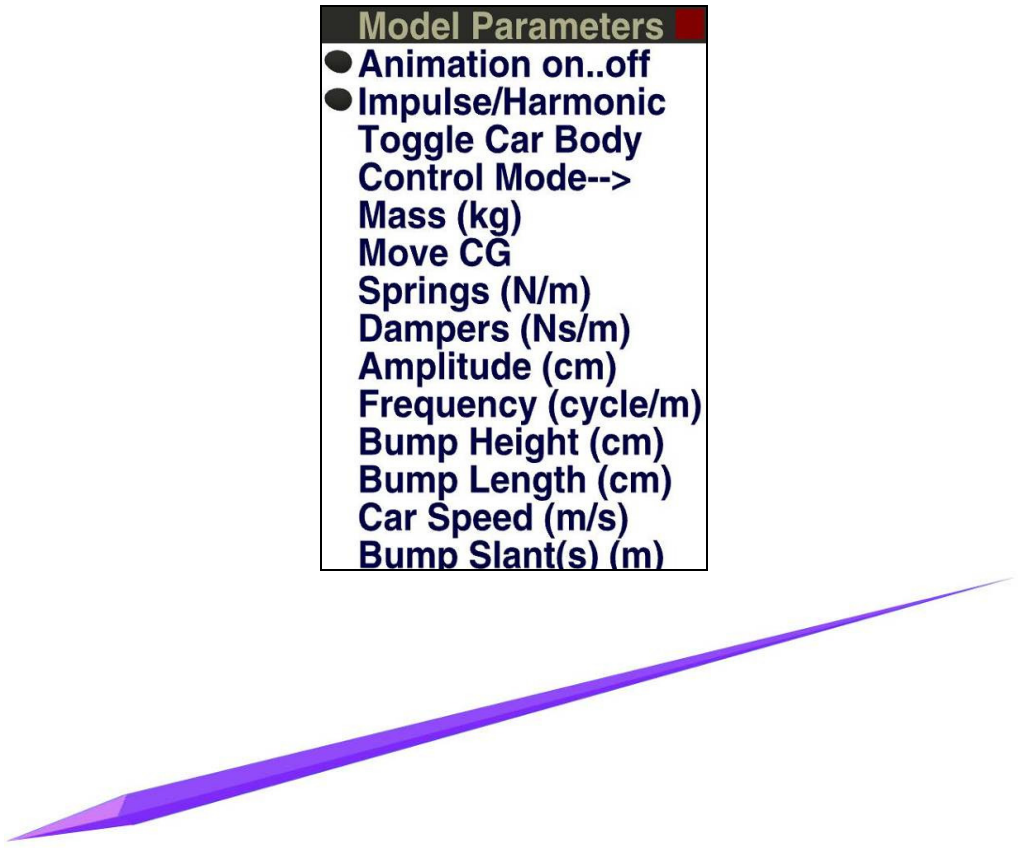

Figure 10: Program menu and virtual pointer

The menu itself is composed of three types of interfaces. The first two on the menu are Boolean buttons allowing for one of two situations and for this menu allows an on/off toggle for Animation and a toggle between harmonic and impulse (bump) inputs. Figure 11 gives an example of the Animation toggled on.

Figure 11: Menu with Animation On

The second interface is a radio button which allows a choice between multiple states. Figure 12 allows a toggle for the car body and type of control used in the model.

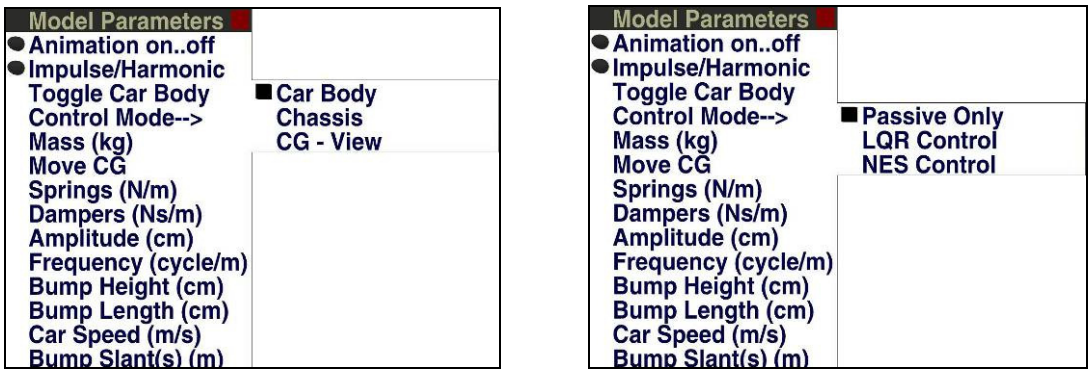

Figure 12: Menu with car body and control mode radio buttons

The final interface is a value adjuster. The program allows for adjustments in mass of the chassis, center of gravity, spring and damper coefficients, the amplitude and frequency of the harmonic input, the height and length of the speed bump, the car speed, and the phase shift of the input between the right and left wheels. The phase shift allows the user to adjust the σ_l from (2.17) and (2.18) affecting both the Impulse and Harmonic

inputs as depicted in the figures. If set to zero, the model with essentially virtually become a four degree of freedom model where the roll is negated. Each value adjustments have two levels of positive and negative adjustment that allow for fine and coarse adjustment. An example of the value adjuster is shown in Figure 13 with the adjustment of mass.

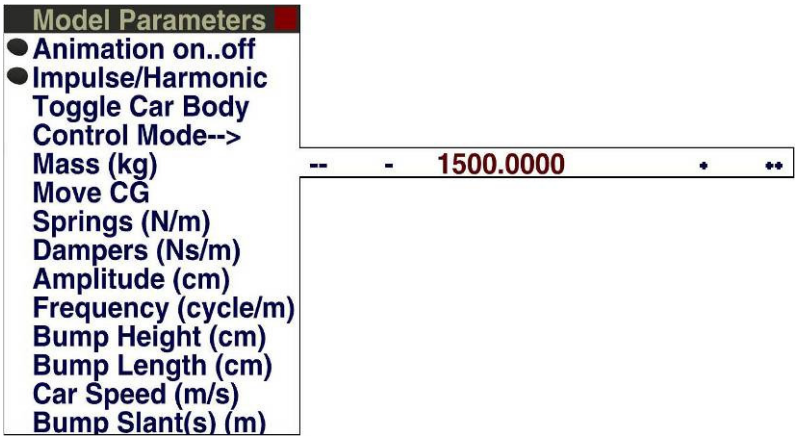

Figure 13: Menu with mass value adjuster

Aside from using the wand to make changes to the system parameters, the wand is also used to reset the start position of the car and navigate within the virtual environment. The program allows for 4 degrees of movement including x, y, z and rotation about the z (up and down) axis.

Since the geometrical representation of the road input is visually subject to the parameter changes, the geometry of the road and bump is created in the program itself. The car and components of the car, however, are for the most part merely rotated or translated. Therefore, these are created using CAD software and imported into the virtual environment.

5.2 Simulation Results

Many of the parameters show on the menu can be seen immediately after they are changed. Figure 14, Figure 15 and Figure 16 show the car model with the car body, chassis, and the center of gravity view respectively. The chassis view exposes the suspension system to the observer and the center of gravity view allows the user to see where the longitudinal center of gravity is which changes as the parameter changes.

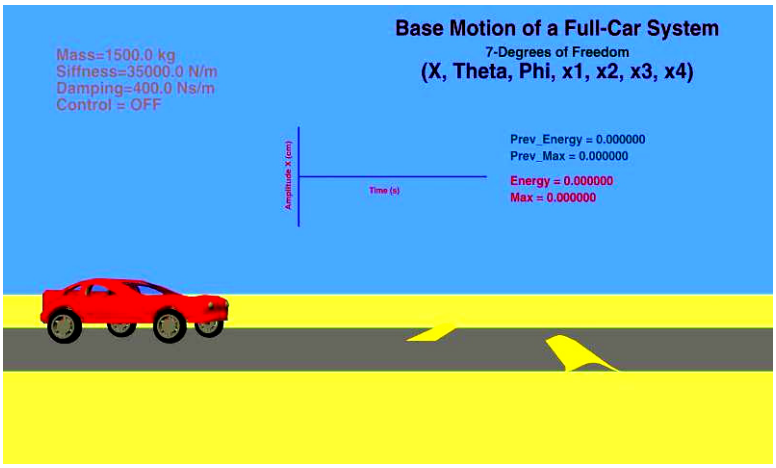

Figure 14: Model with car body

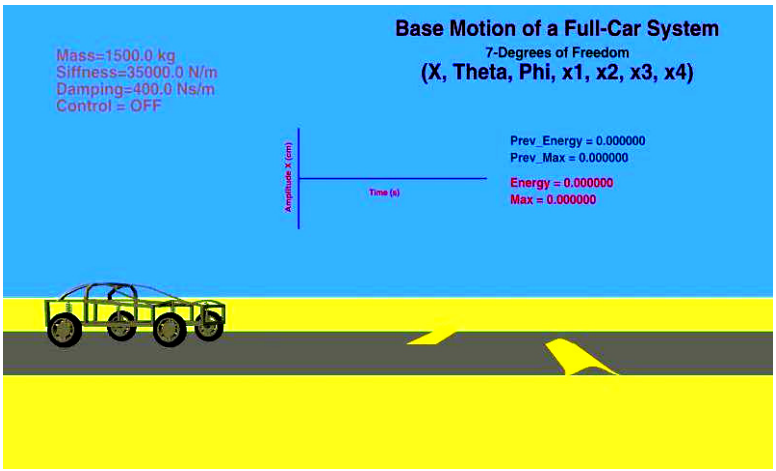

Figure 15: Model with chassis

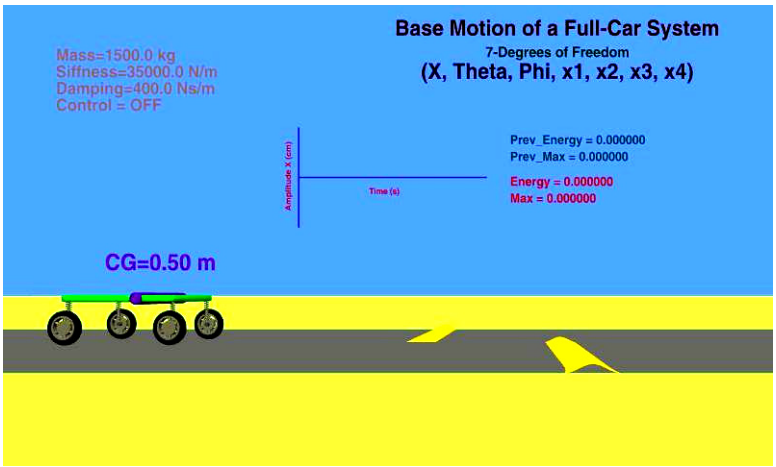

Figure 16: Model with center of gravity

Note on the upper left of the interface and in Figure 17, the current car model parameters are displayed. Also a graph (Figure 18) is also displaying the graphed vertical displacement of the driver as well as the numerical max displacement and energy of the displaced driver for both the current run and the previous run. This allows a comparison between different parameters.

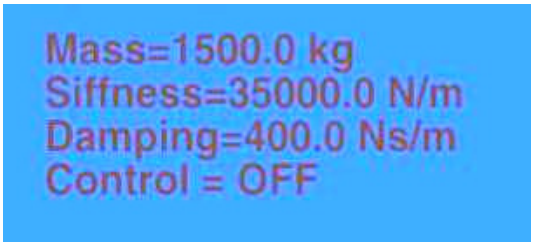

Figure 17: Model parameters with passive only control

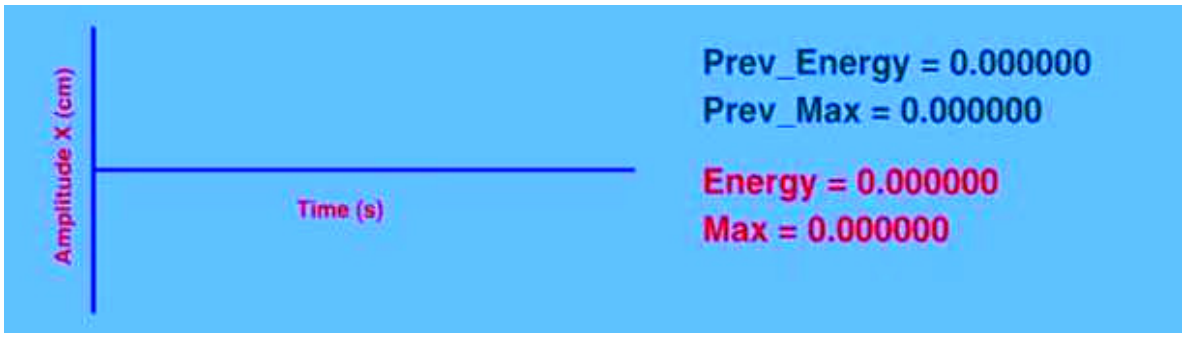

Figure 18: Graph of driver's vertical displacement

The figures below depict changes in bump height and length, change in left to right phase, and amplitude and frequency of the harmonic input.

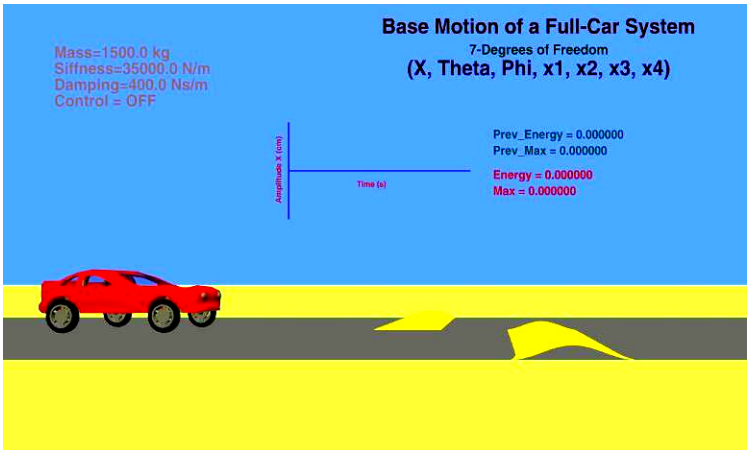

Figure 19: Road with long and high bump far apart

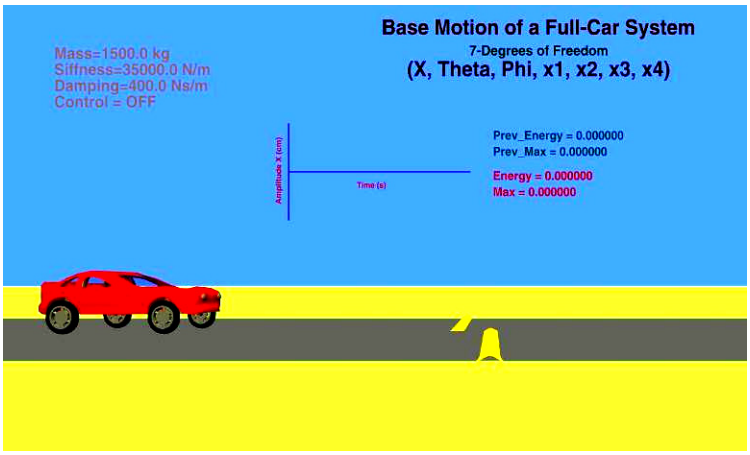

Figure 20: Road with short and low bump close together

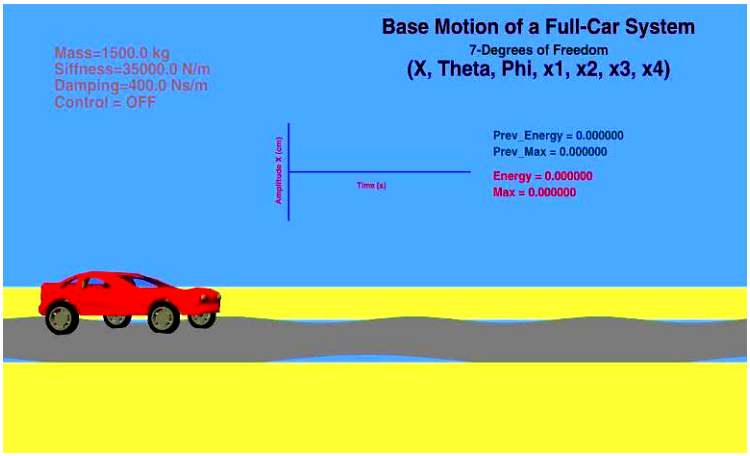

Figure 21: Low frequency, high amplitude harmonic input

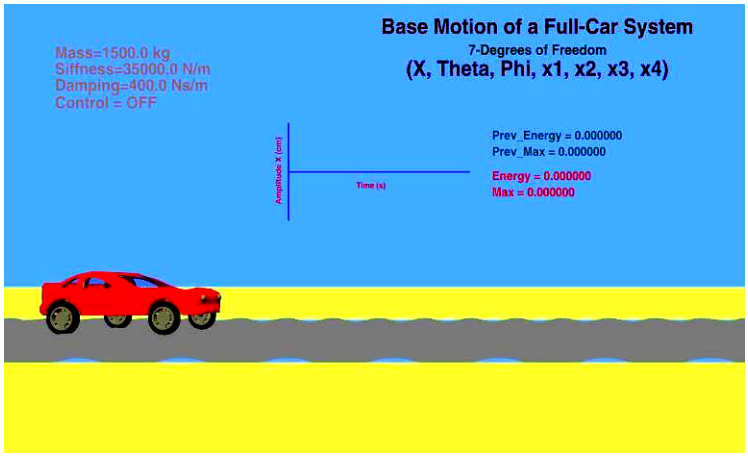

Figure 22: High frequency low amplitude harmonic input

The figures below demonstrate the graph in action. The two scenarios run the car at 2 m/s each showing one run with passive control and the second run with LQR and NEA control respectively.

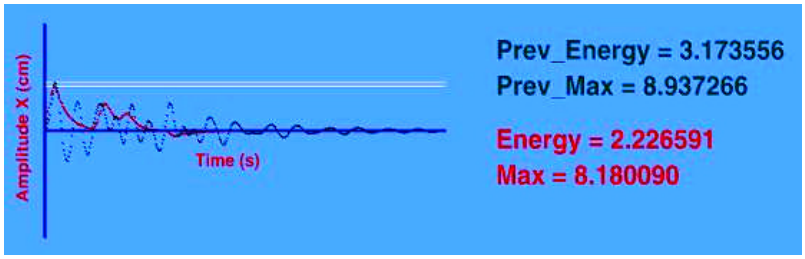

Figure 23: Car moving at 2 m/s without and with LQR control

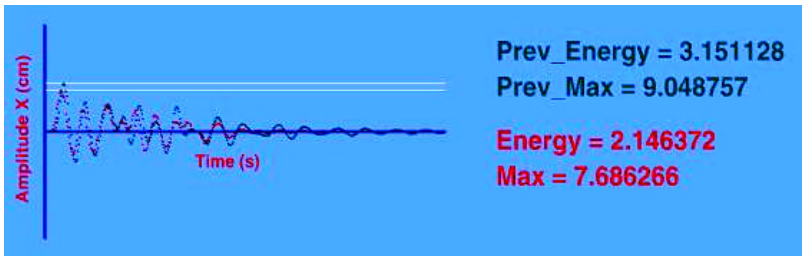

Figure 24: Car moving at 2 m/s without and with NEA control

6 Conclusions

A virtual model of a car suspension system with active control that can be demonstrated in real time was successfully created. LQR controller and the novel NEA controller which includes nonlinear damping were demonstrated and compared. This virtual reality model will allow a new medium for learning and understanding mechanical behavior of a car suspension system. The interface menu allows a user to change the parameters of the car in a moment and instantly see the changes in response that result. This instantaneous change lends a great advantage over physical models used in demonstration, which require time and effort to adjust parameters such as stiffer springs, or more massive weights. This thesis outlines the process of not only creating the system model of a full car, but also allowing it to be adjustable so to enable a user to explore the system in different configurations.

Appendix A: CAVE Lab

A.1: Layout and Introduction
- -VR Lab layout

A.2: Projection
- -Projectors
- -Mirrors
- -CAVE walls

A.3: Tracking
- -*Ascension: Flock of Birds*
- -Motion Tracker
- -Range Extension Controller

A.4: Audio
- -MIDI system
- -Speaker

A.5: Interface
- -Stereo Glasses
- -Wand
- -Glove
- -Headset

A.1 Layout and Introduction

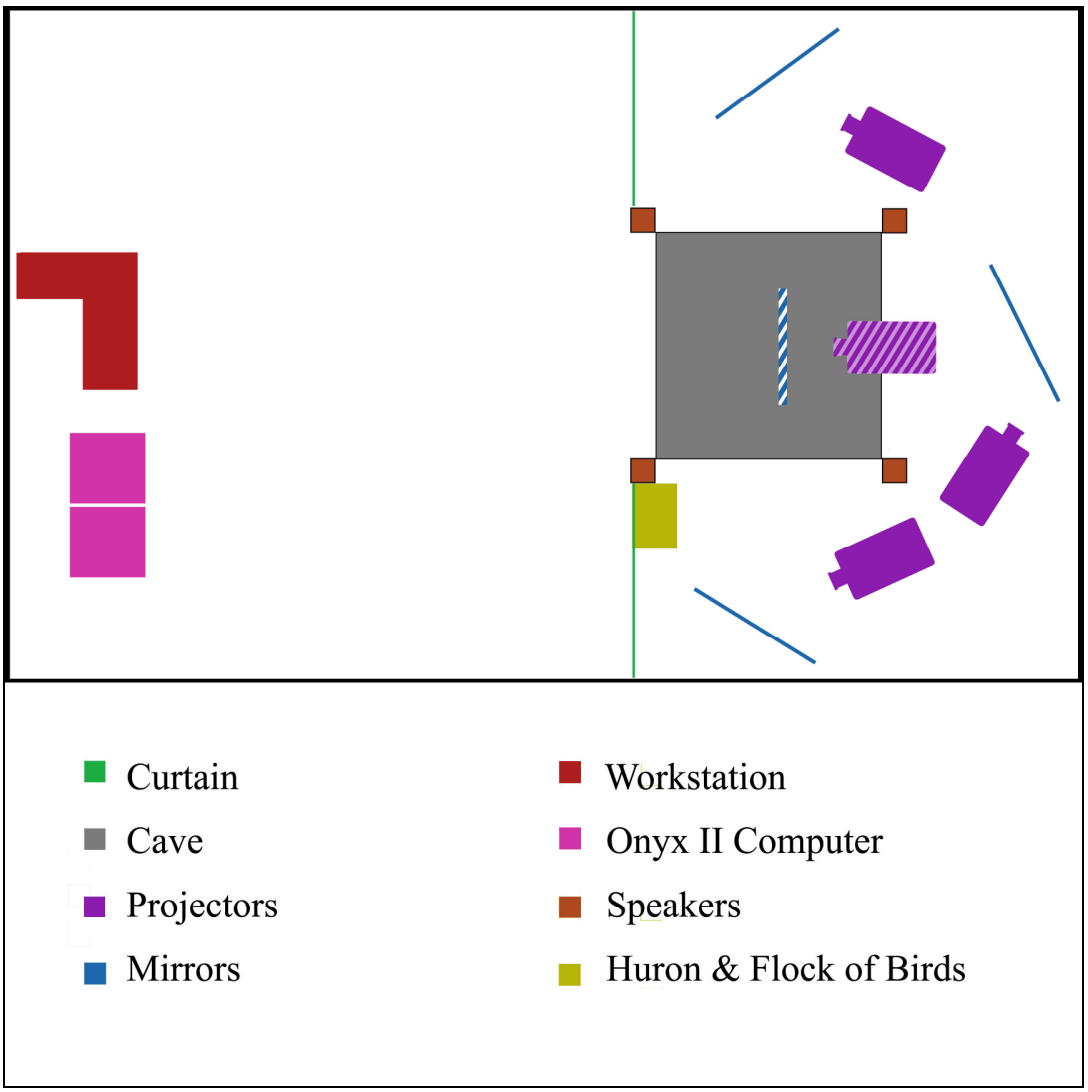

Figure 25: VR Lab Layout

The Figure above describes the layout of the CAVE components in virtual reality lab at the University of Maryland. Note that the cross hatched figures represent items mounted from overhead while the remaining items are near eyelevel or below. The following sections will overview the hardware components that are utilized in the CAVE experience. These were written as a tutorial for future users of the CAVE system.

A.2 Projection

There are four digital *Galaxy* projectors that display the images of for the CAVE system. The projectors are remote controllable and are capable of analog and digital

inputs. Three of these projectors are set on the floor to rear project on the walls of the CAVE while the fourth is mounted from above front projecting down to the floor.

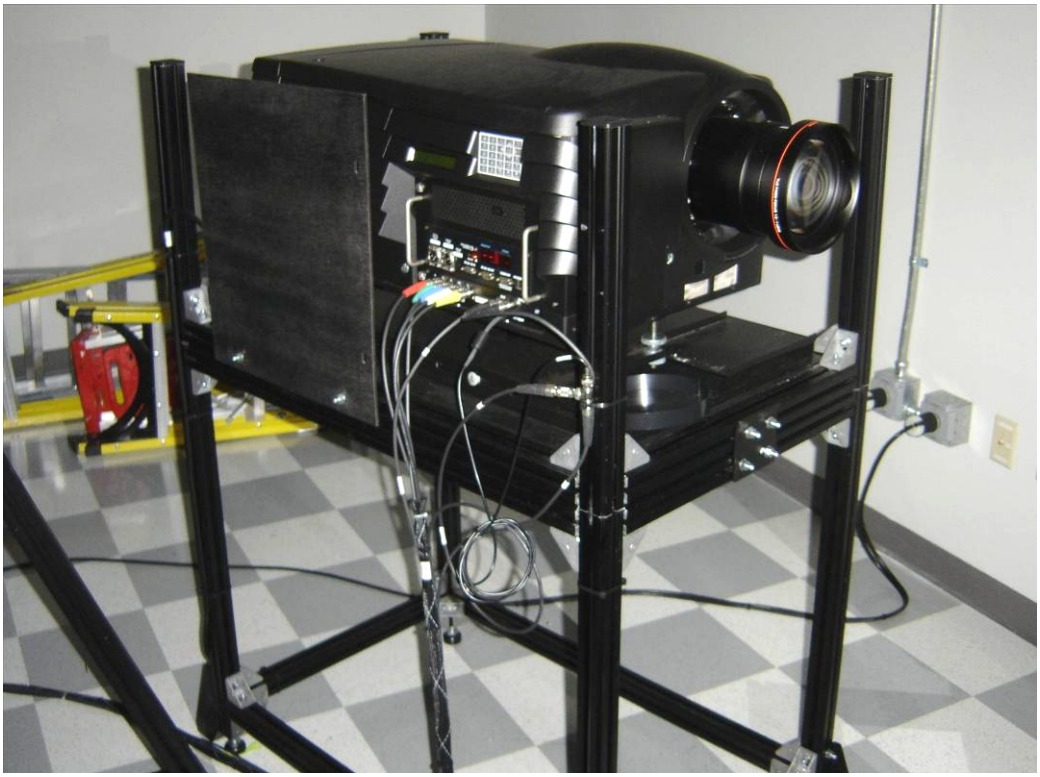

Figure 26: Standing Projector

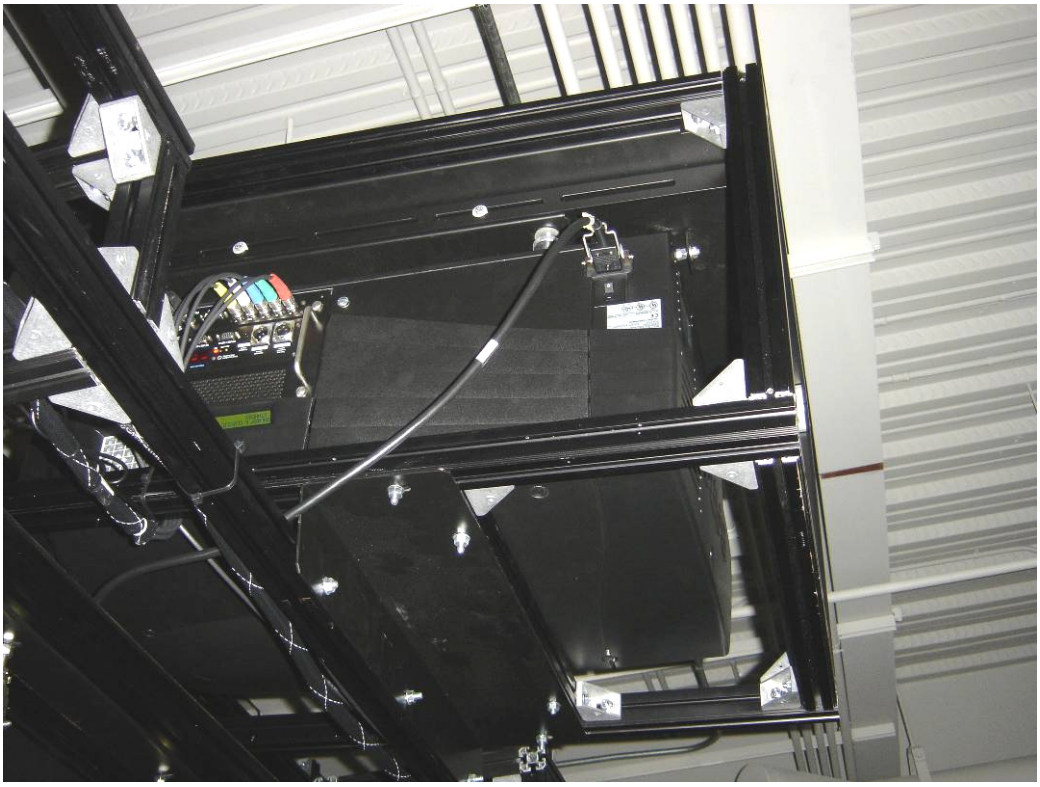

Figure 27: Hanging Projector (Back)

Figure 28: Hanging Projector (Front)

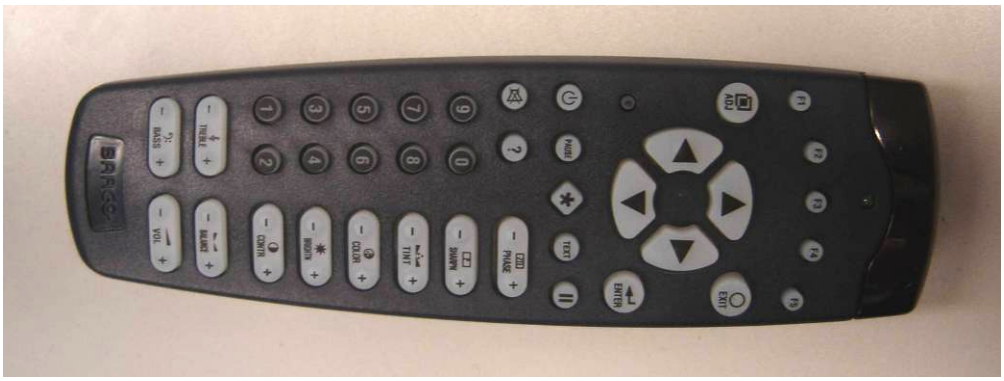

Figure 29: Projector Remote

With respect to the lens being in the front, the right side of the projector houses all the connection for the projector. The main power supply and switch is located closer to the rear of the projector (Figure 30) while signal connections and interface buttons are located closer to the front (Figure 31).

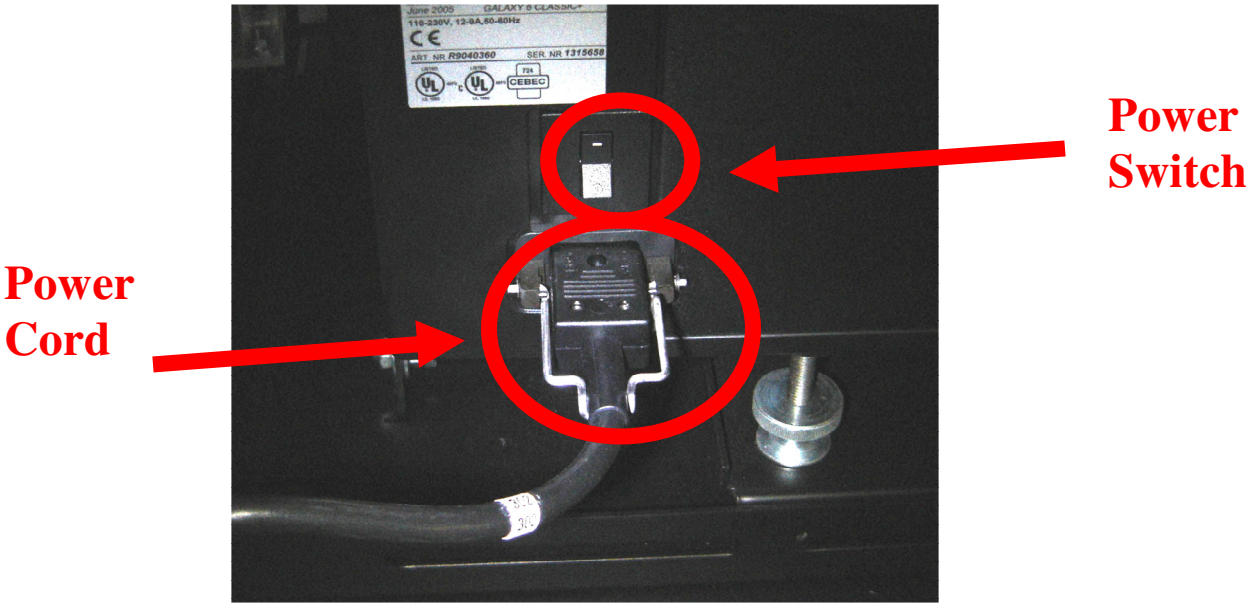

Figure 30: Projector Front

There are three type of connection on the back of each projector: the RGB inputs, the Horizontal and Vertical Sync for the video and the IR sync for the stereo glasses. The first two connect directly back to the Onyx2, while the IR sync signal connects to the other projectors and to the IR emitters described in section A.5. The projector shown in Figure 31 shows all three types of connection in the order described from left to right.

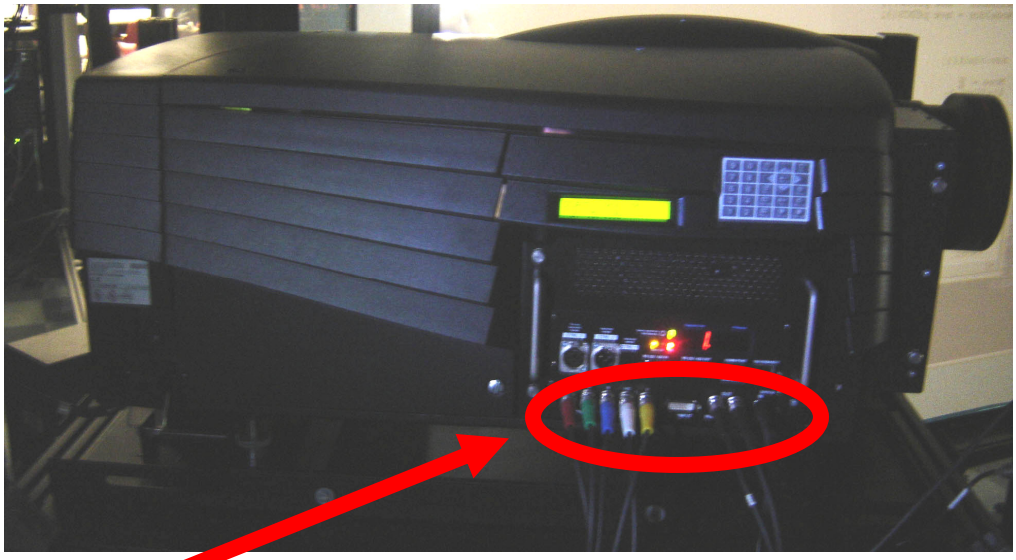

Figure 31: Projector Back

As shown in the layout in Figure 25, the projectors use mirrors to project their image onto the screen. This setup allows more distance for the image to expand to the desired size while saving floor space.

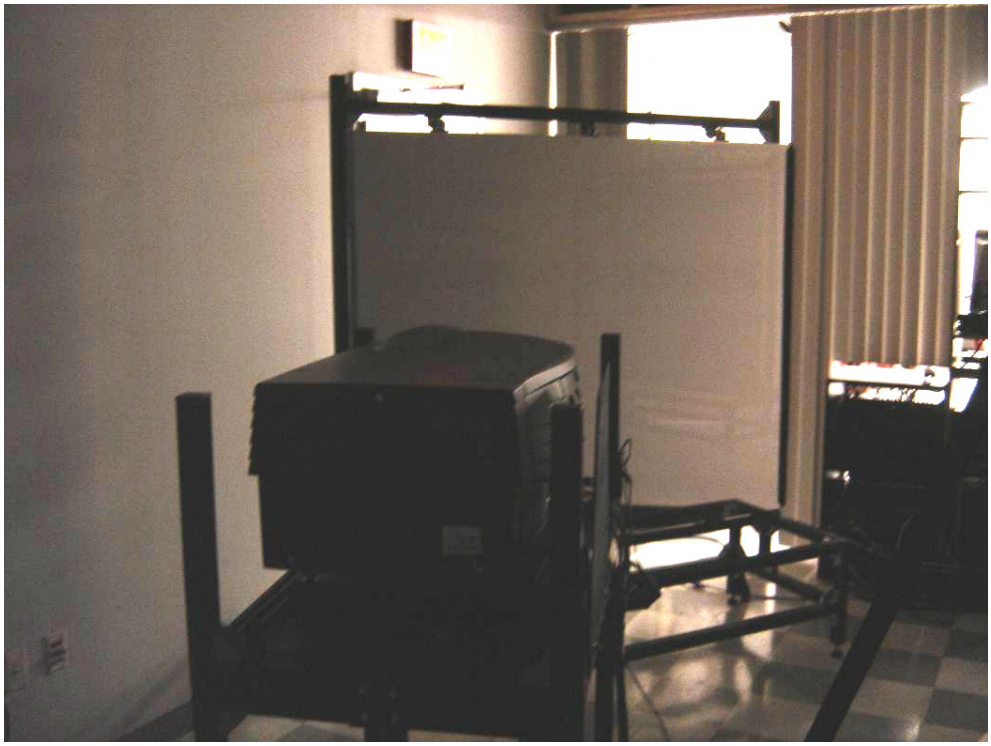

Figure 32: Sitting Mirror

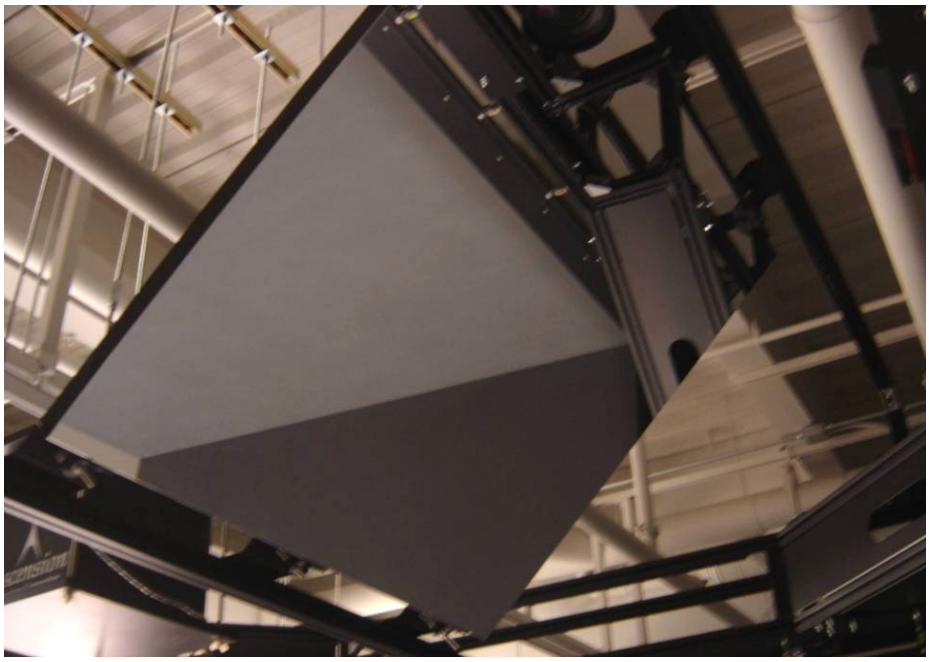

Figure 33: Overhead Mirror

The images from the projectors are ultimately drawn upon a hard screen. The CAVE walls are shown below from the inner side of the CAVE.

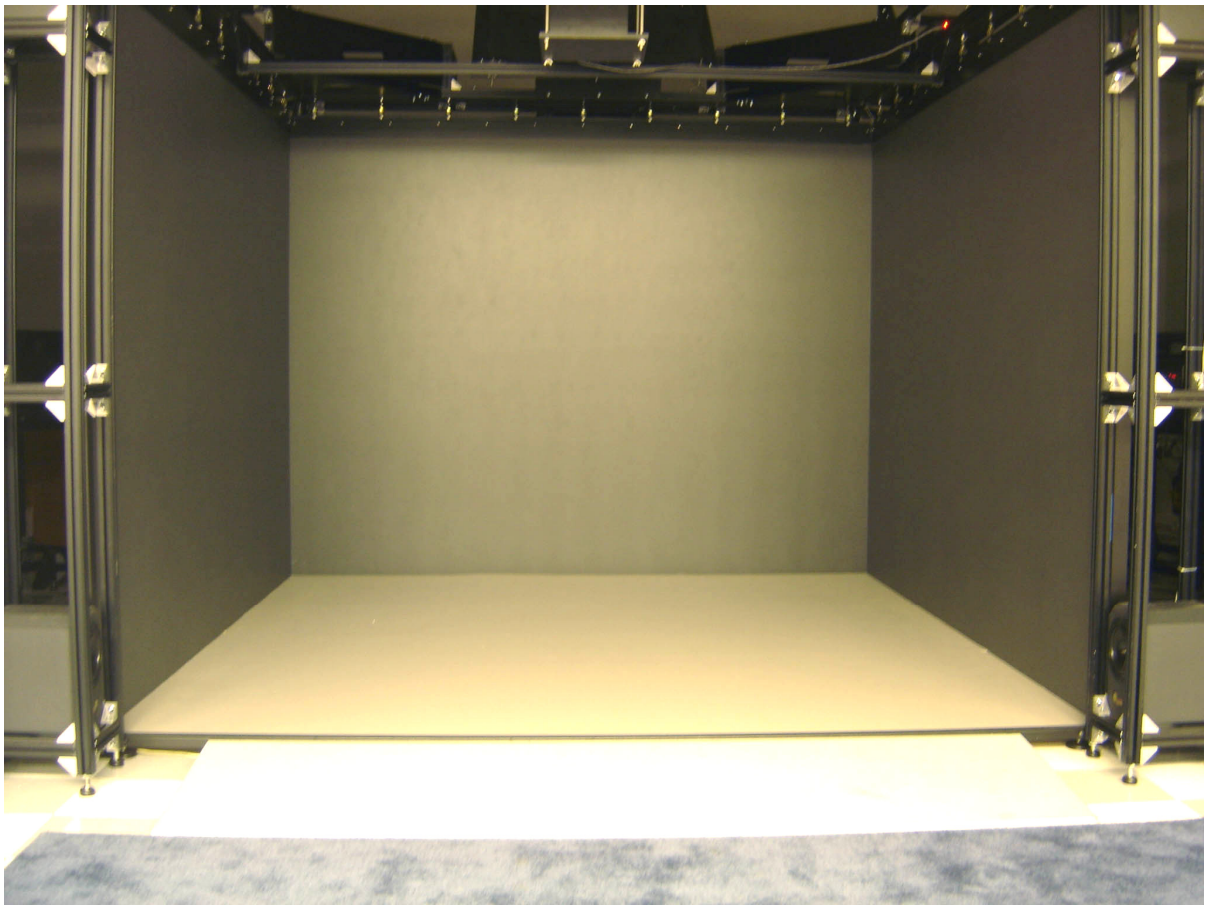

Figure 34: CAVE walls and floor

A.3 Tracking

A major part of the interface with the cave is the tracking involved that tells the system where the user is located so that actions taken by the user will be reflected in the change of the images projected. In the CAVE, this is accomplished through the *Ascension: Flock of Birds* tracker shown below. This system is able to track both three dimensional position as well as sensor orientation.

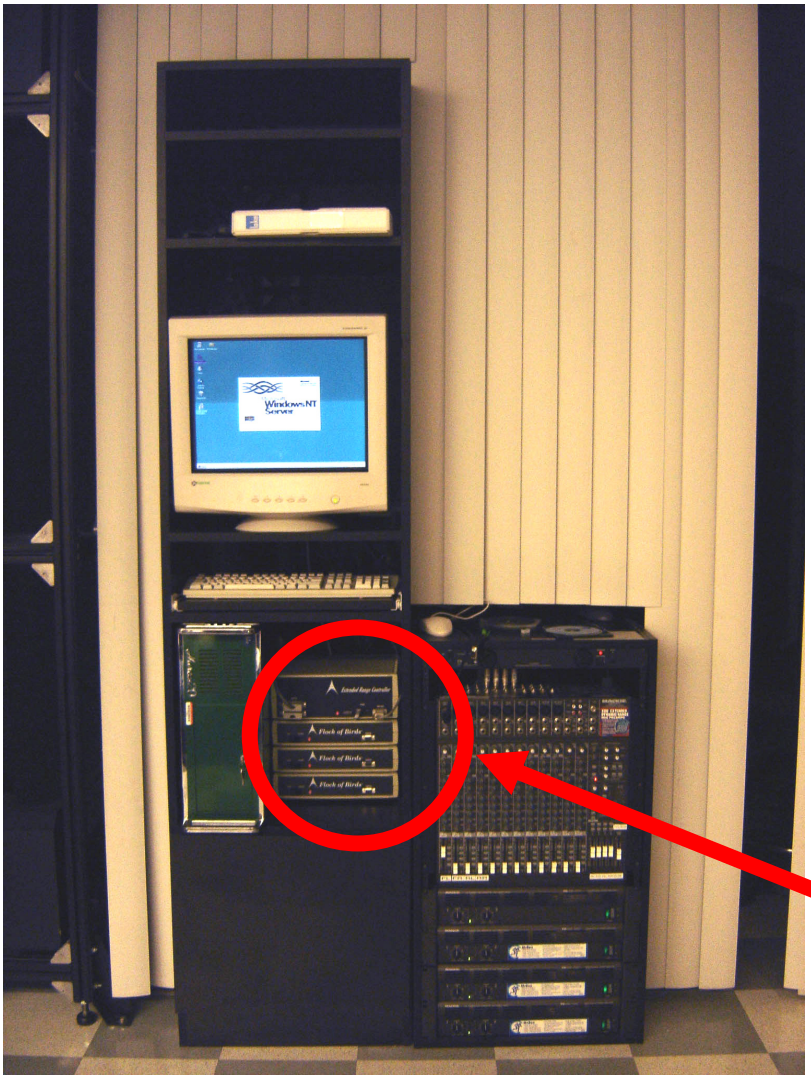

Figure 35: Three sensor controllers and the extended range controller

This CAVE system has three sensor controllers shown below, and three corresponding sensors, which will be explained section A.4. The back three controllers

are connected in series to each other by Fast Bird Bus (FBB). Each also is connected to a power cord and its corresponding sensor.

Figure 36: Flock of Birds: Sensor Controllers

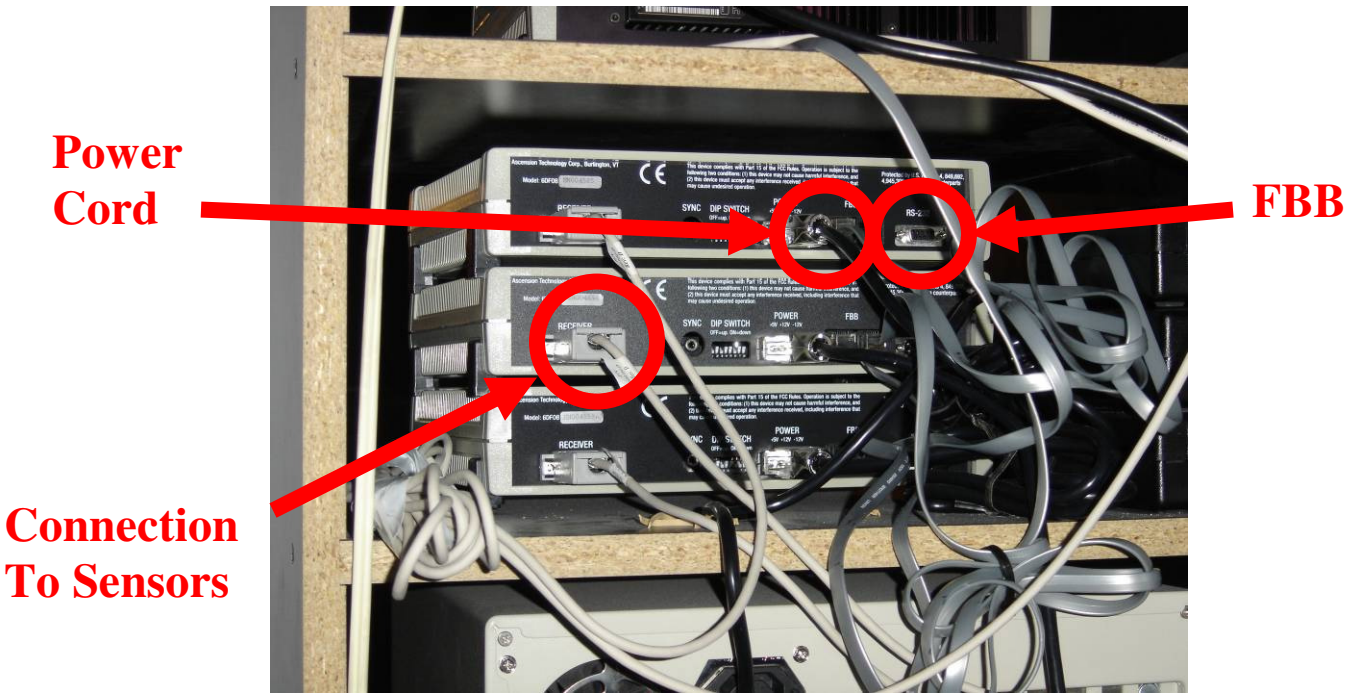

Figure 37: Back of FOB sensor controllers

Because the CAVE space is so large, the sensor controllers are hooked up to an extended range controller and extended range transmitter located above the regular sensor controllers.

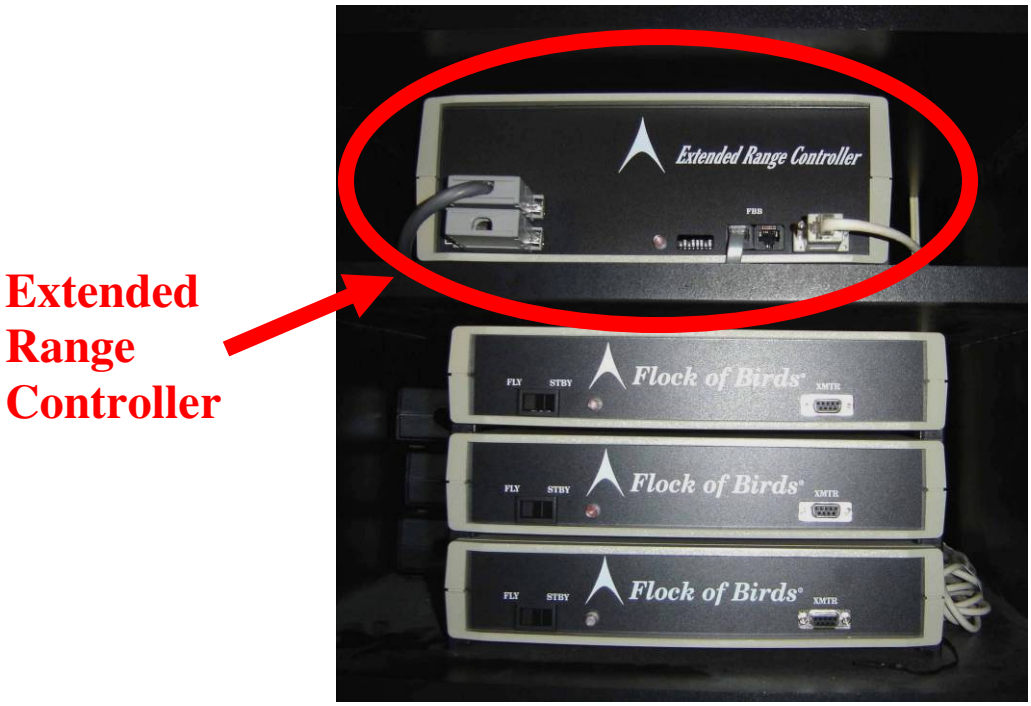

Figure 38: Extended Range Controller

The FBB from one of the regular sensor controllers (dubbed the master controller), is fed into the front of the extended range controller. The front is also connected to the extended range transmitter and also back to the Onyx2 via RS-232 serial cable. The back of the extended range controller is connected simply to a power cord.

RS-232 Back to Onyx2

Feed to Extended Range Transmitter

Figure 39: Front of Extended Range Controller

Figure 40: Back of Extended Range Controller

Power Cord

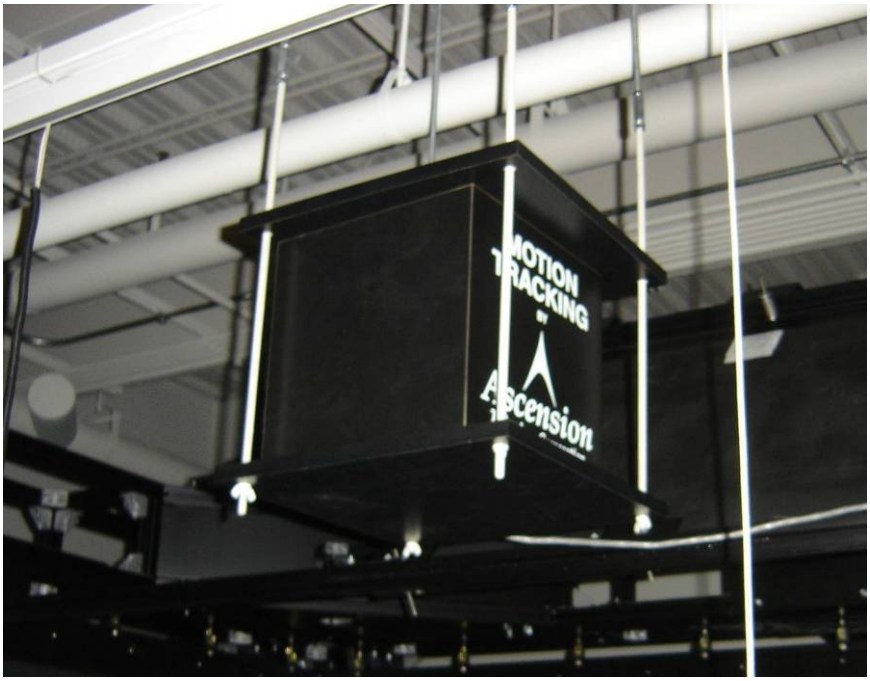

Figure 41: Extended Range Tracker

A.4 Audio

The audio control of the CAVE is handled by the HURON PC shown in Figure 42. The back of the HURON has the standard power, VGA, keyboard, and mouse connection as well as an RJ-45 connection to the Onyx2, an audio input connection, and an audio output connection.

Figure 42: Huron PC

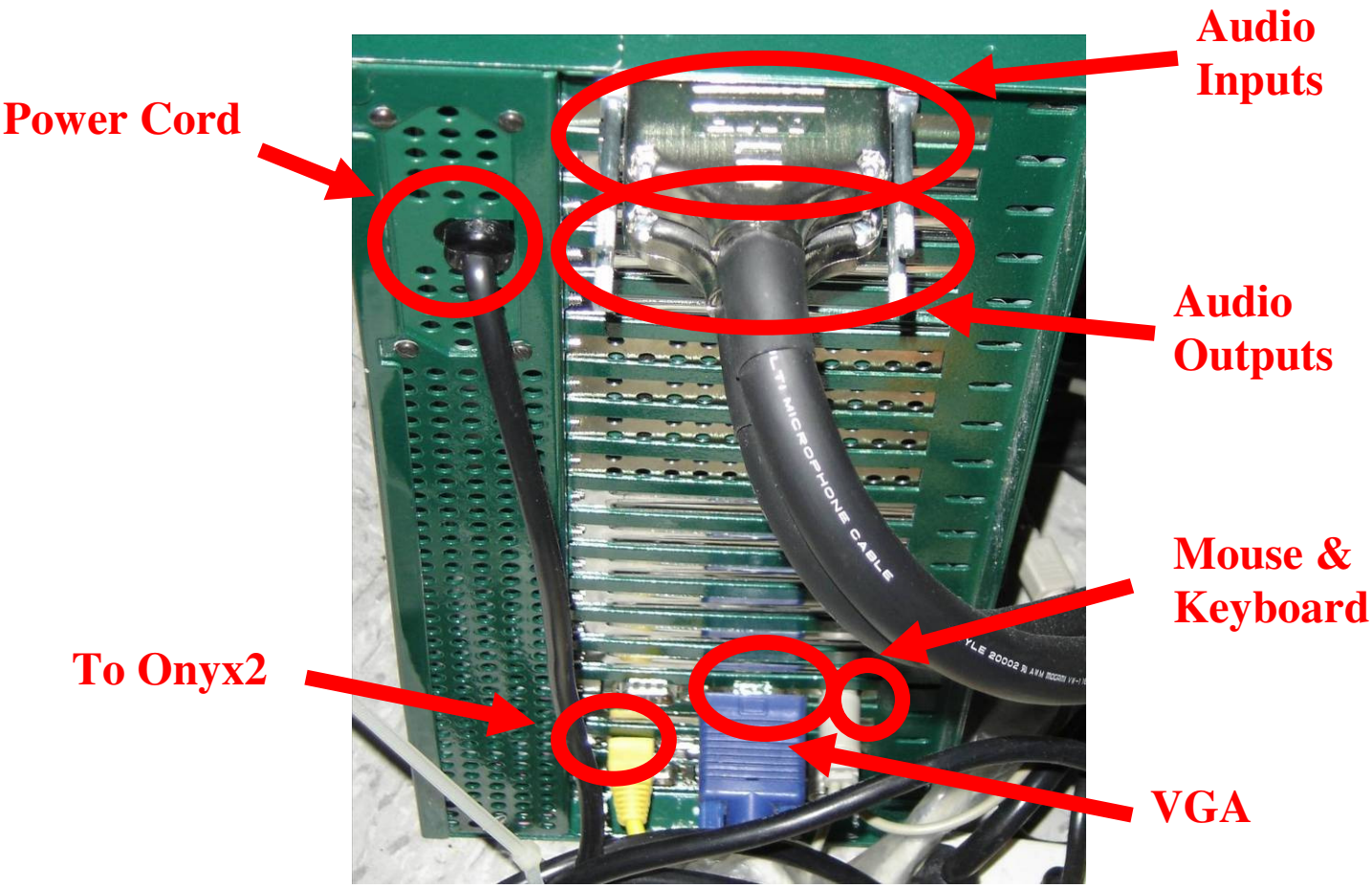

Figure 43: Back of Huron PC

The audio lines are then sent to a splitter to be separated into individual channels. In this setup there are eight audio output channels and one audio input.

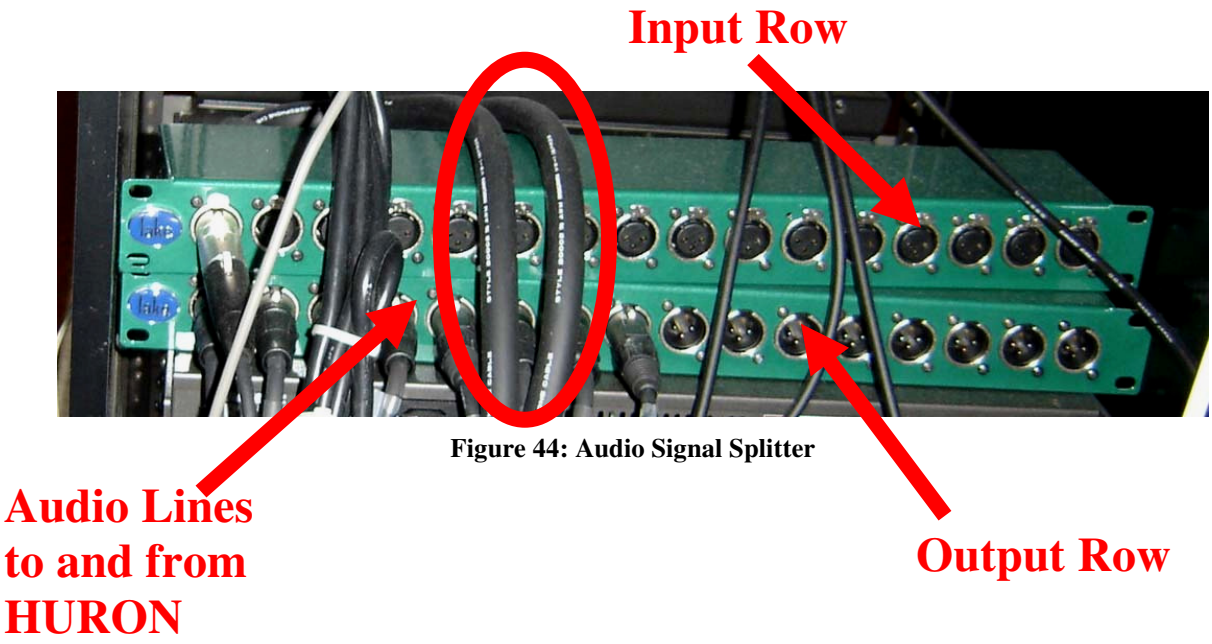

Figure 44: Audio Signal Splitter

From here, the single input is sent to the headset microphone described in section A.1.4, while the eight outputs are sent to 4 x 2-Channel High Current Mackie Amplifiers.

(Drawing taken from MACKIE M800 OWNERS MANUAL to avoid wire clutter)

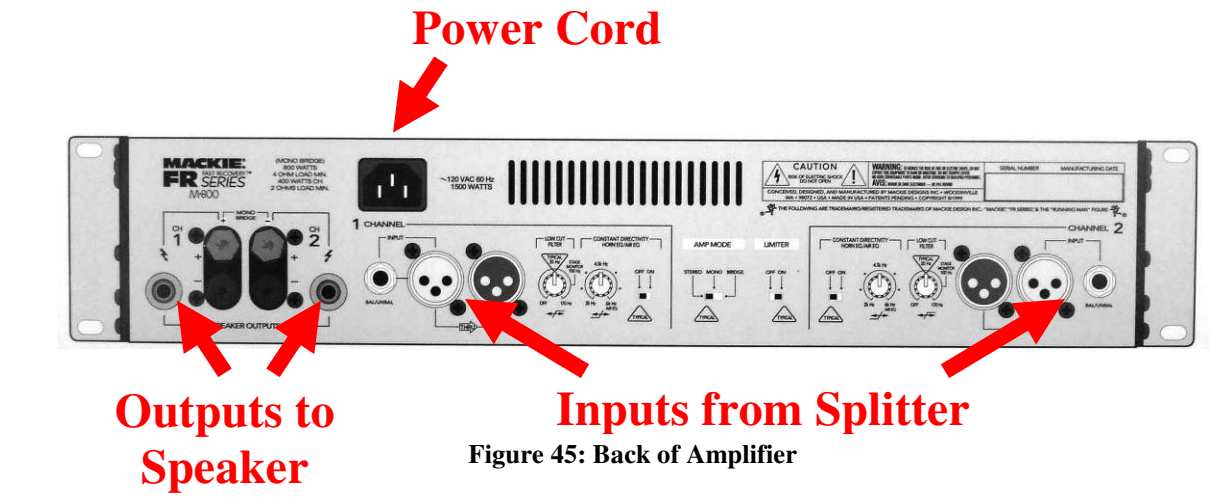

Figure 45: Back of Amplifier

The speakers are divided up into four sets of two. Each pair is divided by height at each corner of CAVE. One is set on the ground, while the other is mounted from above as shown.

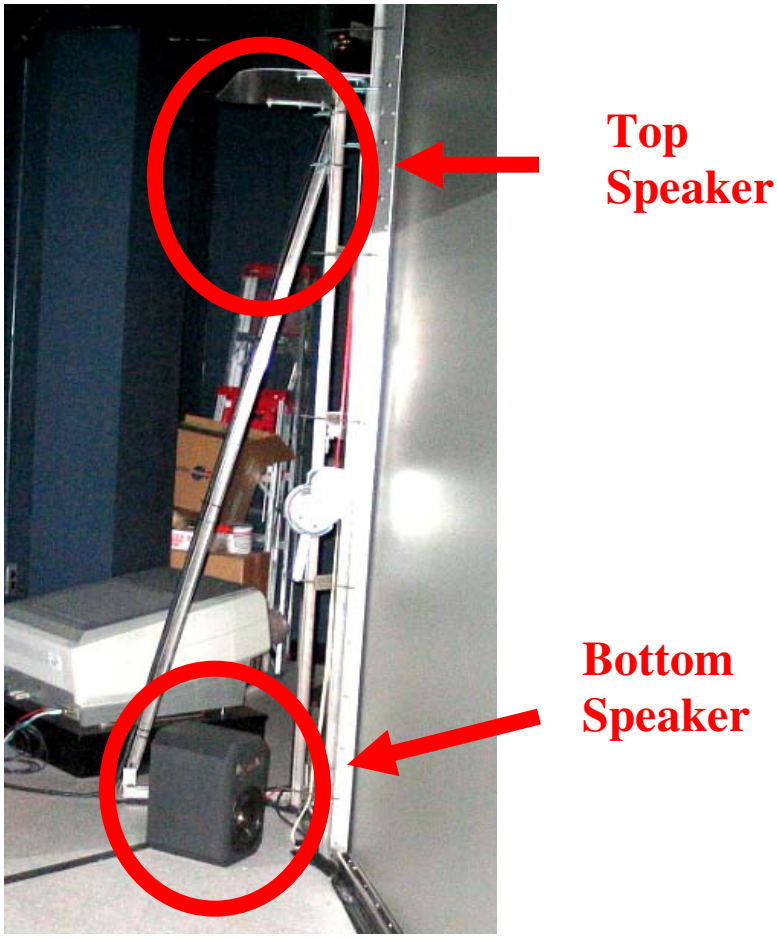

Figure 46: Corner Pair of Speakers

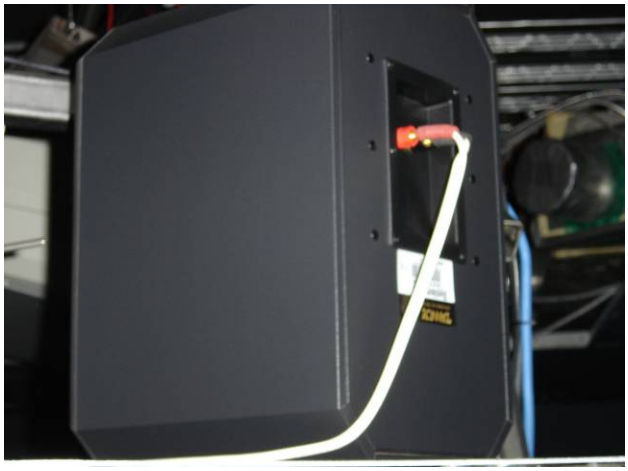

Figure 47: Connection at Back of Speaker

A.5 Hardware Interface

The last few items of the CAVE system are vital to interacting with the CAVE system once it is running. These items are located within the CAVE walls hanging from the ceiling structure. They can be seen below in Figure 48 as opposed to the CAVE walls without the interface items shown in Figure 34.

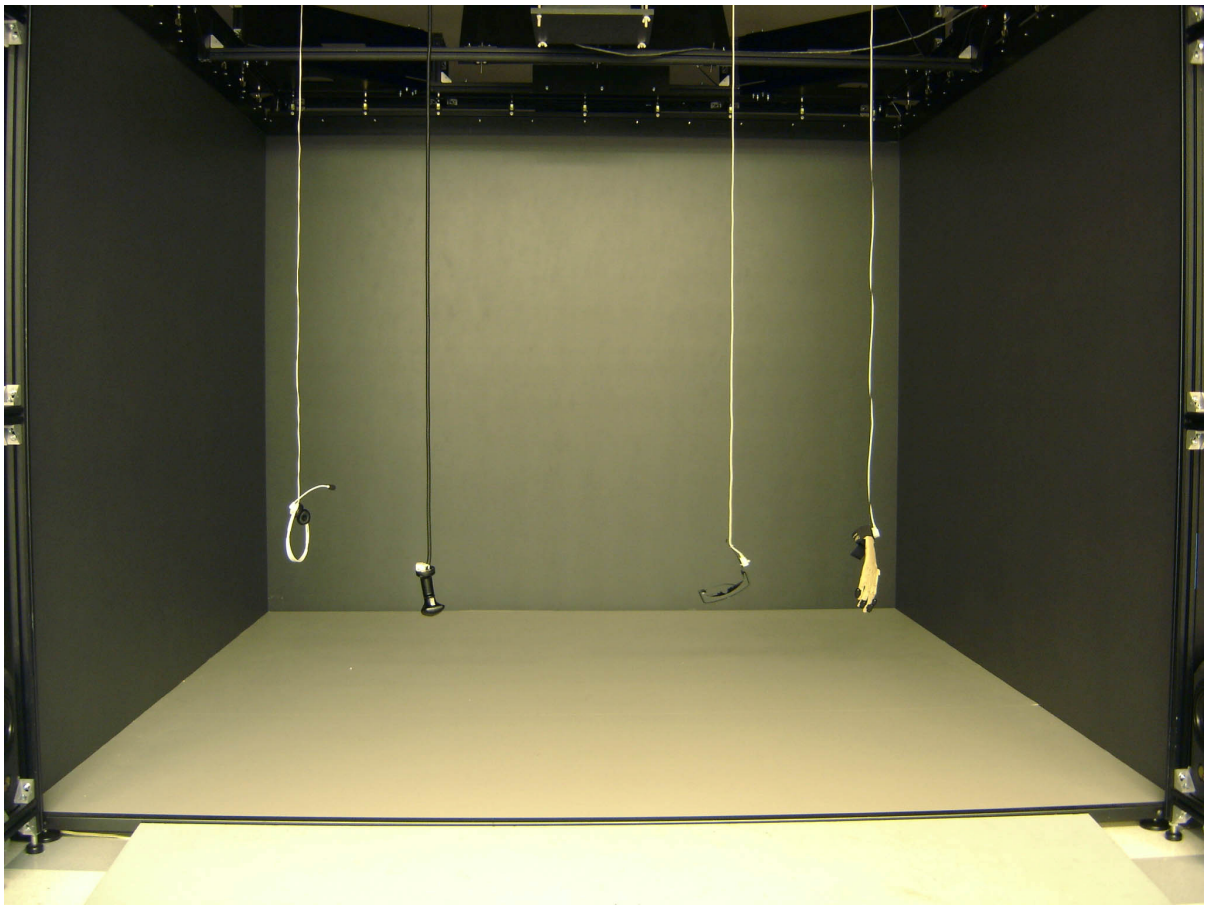

Figure 48: CAVE with interface items hanging

The foremost of these items are the stereo glasses (CrystalEYES Stereo Graphics). The stereo glasses are vital to give the user the virtual and interactive feel. When the Onyx displays a Virtual environment, it actually displays two similar images adjacent to each other. The stereo glasses are then used to combine these images into one 3D image.

Figure 49: Stereo Glasses

The adjacent images are drawn separately but at such a high rate of speed that the human eye can not make out the individual images. The images are also drawn askew from one another to emulate the different angles that the eyes see each image. The stereo glasses separate these images by turning on and off the polarized lenses in sync with the images so the right lens only lets the right eye see the right images, and the left lens does the same. To turn the polarization on and off, the glasses are powered by two CR2032 batteries located on the left side of the frame.

Figure 50: Batteries for Stereo Glasses

Because they are run on batteries, they must be turn off after use to extend the life of the glasses. This can be done by closing the right earpiece which releases the power trigger on the right ride of the frame.

Figure 51: Power Button

The stereo glasses are able to sync to the images by way of Infra-Red Blasters (IR Blasters). These IR blasters send an IR signal to the glasses to "tell" them when to turn each lens on and off.

Figure 52: IR Emitter(Front)

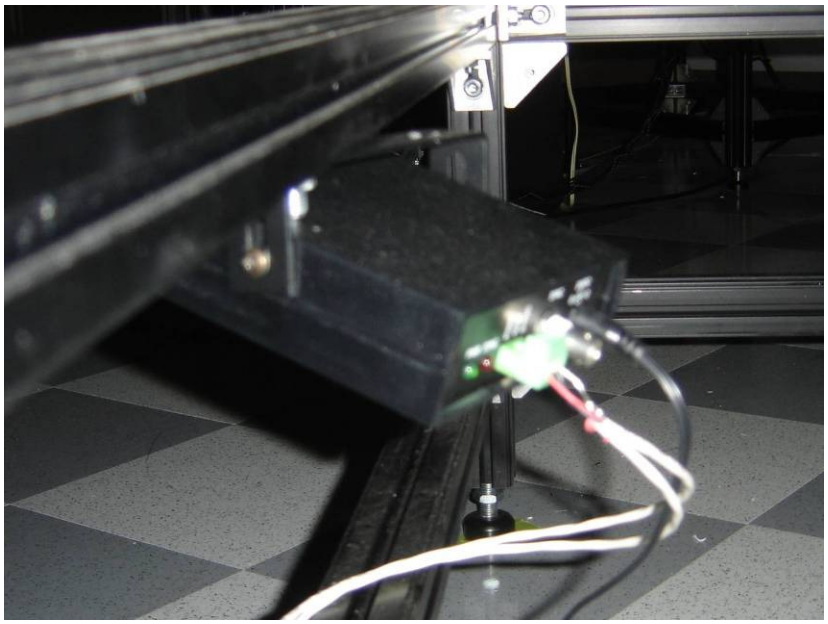

Figure 53: IR Emitter (Back)

These IR Blasters have a matrix of infra-red emitters on the front and are connected on the back by a RS-232 connection adapted to a RJ-45 cable and a power cord. Three of these are located behind the three walls of the CAVE and one is hung next

to the hanging projector. These are hooked in series with each other and terminate at one projector that outputs the synchronous signal.

The side of the stereo glasses is attached one of the sensors of the Flock of Birds tracking system. This not only allows the Onyx to know where the user's head is in the environment, but also to calculate the angles to display the image. As mentioned before, the images are askew from on another to present the different view angle of each eye; however, the system also needs a reference point for the eyes as well. This is provided by the said tracking sensor.

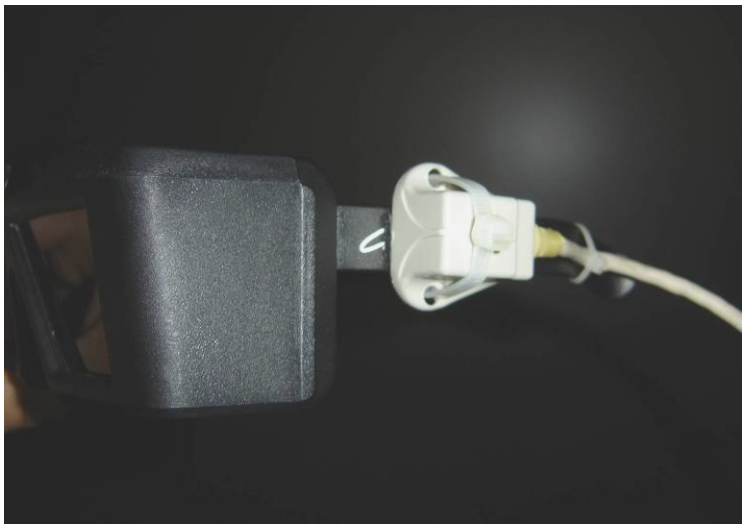

Figure 54: Flock of Birds Sensor on Stereo Glasses

There are additional unattached glasses that secondary users can wear in the CAVE; however, their images won't be as clear as the primary users because the images are not drawn from their view point. The result is a feeling of being cross eyed.

Figure 55: Unattached Stereo Glasses

The next important element for interacting with the CAVE is the wand. The wand is the basic element for virtual reality environment navigation and interaction. The joystick has three buttons and a pressure sensitive joystick. The first button is a trigger button for the index finger, while buttons two and three are the left and right of the joystick respectively.

Figure 56: Sideview of Wand Showing Trigger (Button1)

Figure 57: Back View of Wand Showing Joystick and Buttons 2&3

Like the stereo glasses, the wand also has an attached sensor. The sensor is vital when using the wand to point and choose objects as well as for navigation. The joystick allows forward and reverse movement as well rotation, but does not allow simple lateral movement. Lateral movement, however, can be accomplished by turning the orientation of the joystick so that the new forward/reverse will correspond with the former desired lateral movement. Also attached to the underside of the wand is the RS-232 signal cable that relays the joystick input back to the Onyx2.

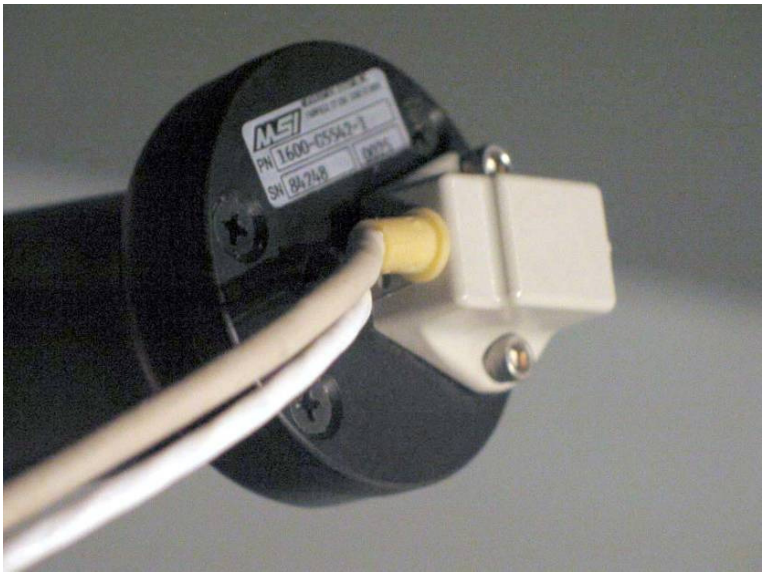

Figure 58: Underside of Wand Showing Sensor and Connection to Onyx2

An additional tool for the CAVE interface is the CyberGlove. The glove itself is equipped with joint sensors that can determine the position of the individual digits of the fingers. The glove also has miniature vibrators on the tips of each finger and on the palm of the hand. These are used in conjunction with the third Flock of Birds sensor for interaction with vibration analysis in the CAVE. In addition to the tracking sensor, the glove also has an RS-232 connection back to the Onyx2. The glove also has an on/off switch located next to the sensor. These can be seen in the figures below.

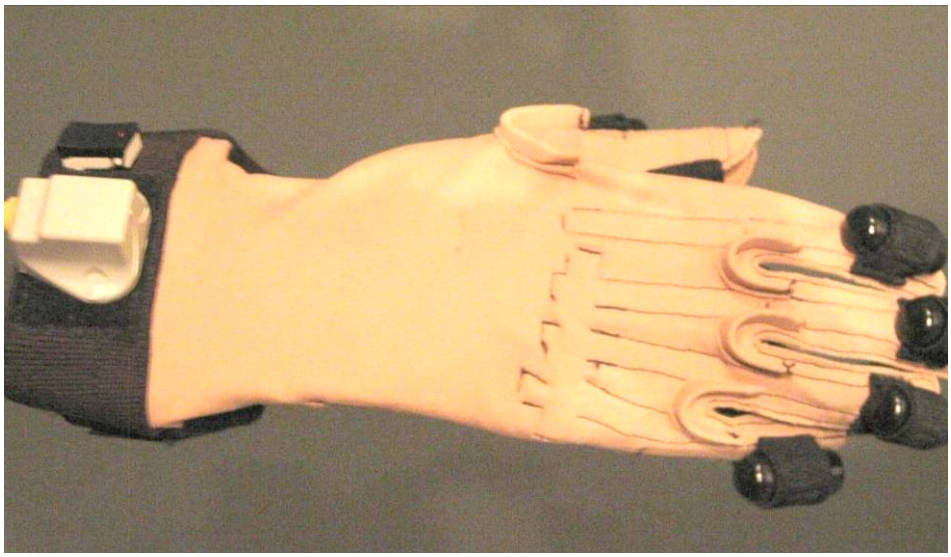

Figure 59: CyberGlove

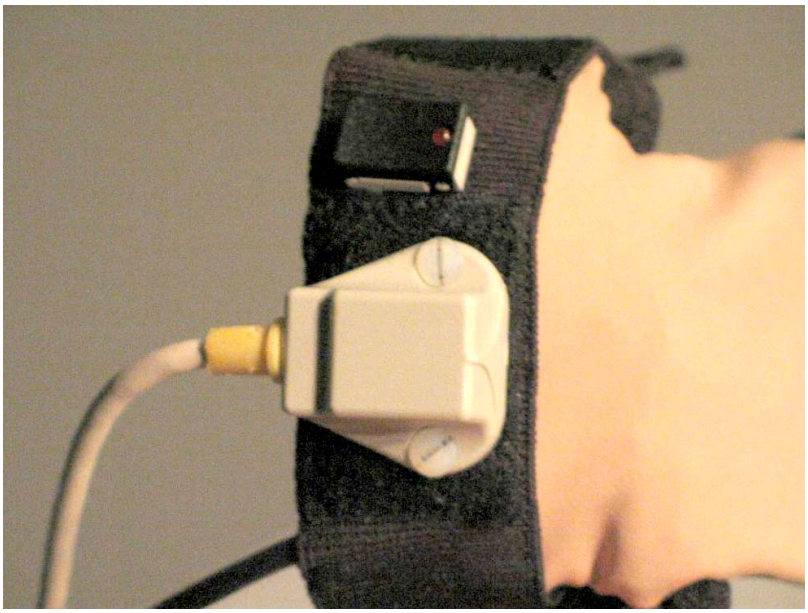

Figure 60: CyberGlove Showing Tracking Sensor and RS-232

Appendix B: Operation of the CAVE

B.1: Startup Sequence
- -Onyx
- -Tracking
- -Audio
- -Projection

B.2: Basic Onyx2 Use (IRIX OS)
- -File Browser
- -Shell Terminal
- -Simple programs
- -Older programs

B.3: Capturing CAVE Images and Video
- -Media Recorder

B.1 Startup Sequence

Normally the majority CAVE systems will be running at all times save items with a short wear out life such as batteries and bulbs. There may be times, however, where the system needs to be restarted from scratch, for example a total power failure. The following section will describe the startup sequence of the components of the CAVE.

Figure 61: Onyx2 Main Panel Location

The first step to starting up the CAVE is to power up the Onyx2. This can be done by pressing the "menu/cancel" button on the main panel of the Onyx2, using the four navigation buttons to locate the "action" menu, scrolling down to "power up," choosing the "action" menu, selecting "power up," and pressing the "execute" button.

Figure 62: Main Panel

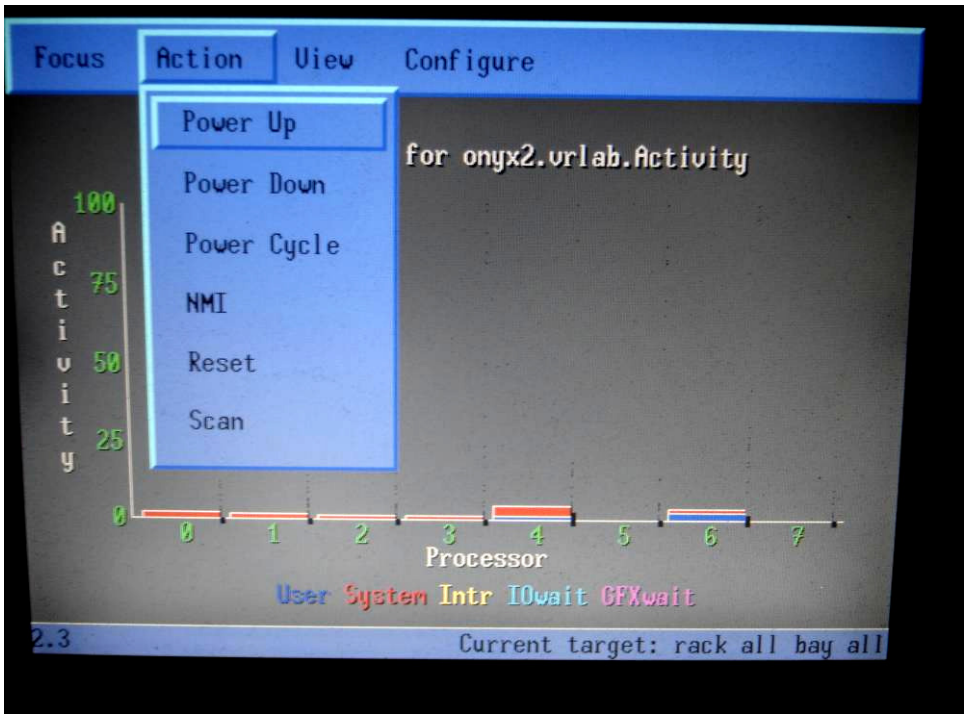

Figure 63: Location of "Power Up"

Once the Onyx2 is powered up the user must login from the workstation.

Figure 64: Workstation

Before proceeding to initiate the tracking, audio, and projectors, the power strips for these systems should be checked to make sure they are on. The first one can be found behind the Huron PC while the second and third are behind the monitors for the Huron and projector PCs.

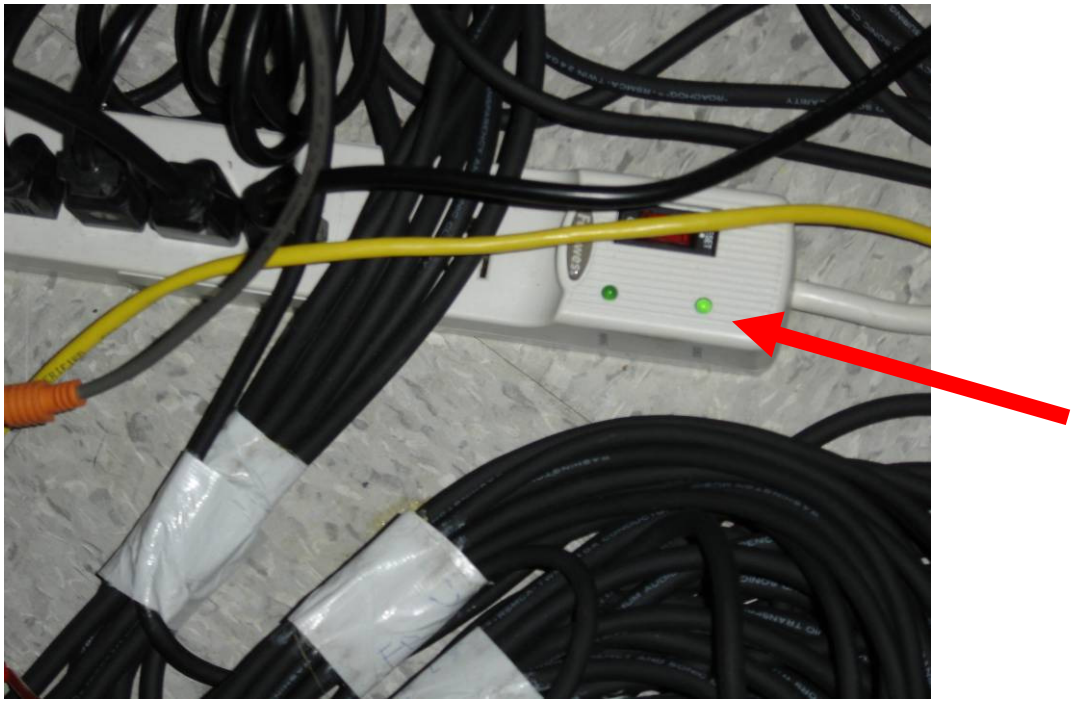

Figure 65: Power Strip Behind Huron PC

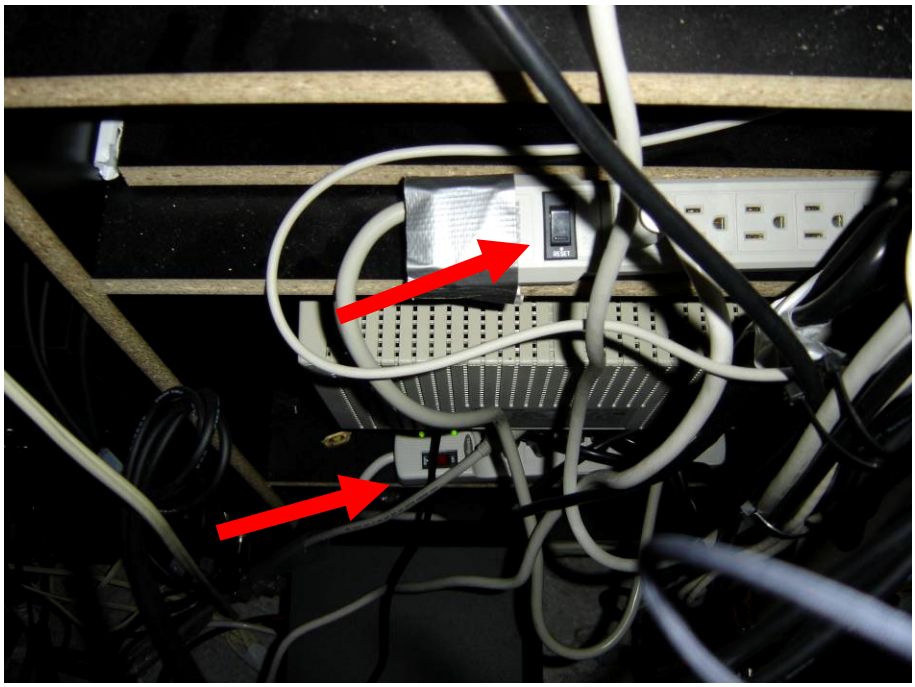

Figure 66: Power Strips Behind Monitors

The tracking system must be started in a certain sequence, so if the system is on, it must be shut off first. Shutting off the sensor controllers can be done from the front. The extended range controller power switch, however, is located in the back.

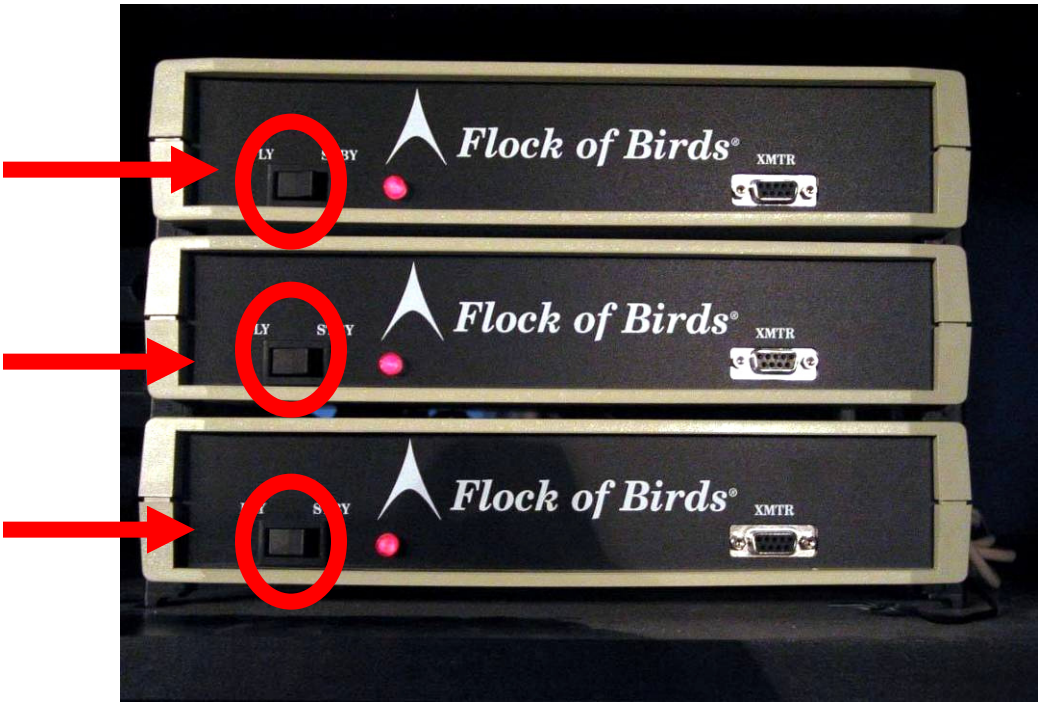

Figure 67: Controller Power Switch

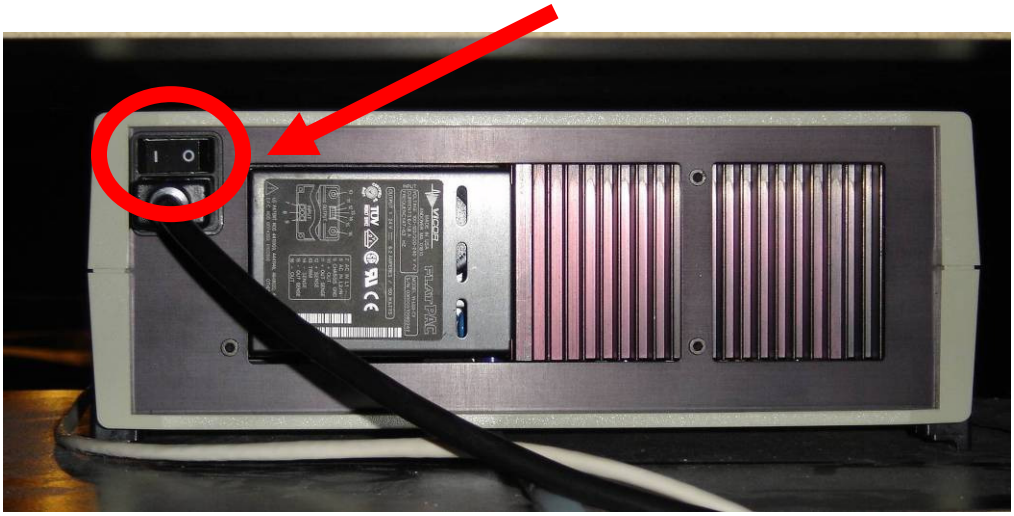

Figure 68: Extended Range Controller Power Switch

Once the controllers are off, the extended range can be turned back on followed by the three regular controllers. Finally, double clicking “Start Tracker” icon will start the system. For the system to be running correctly, all four of the controller lights must be on after “Start Tracker” is run.

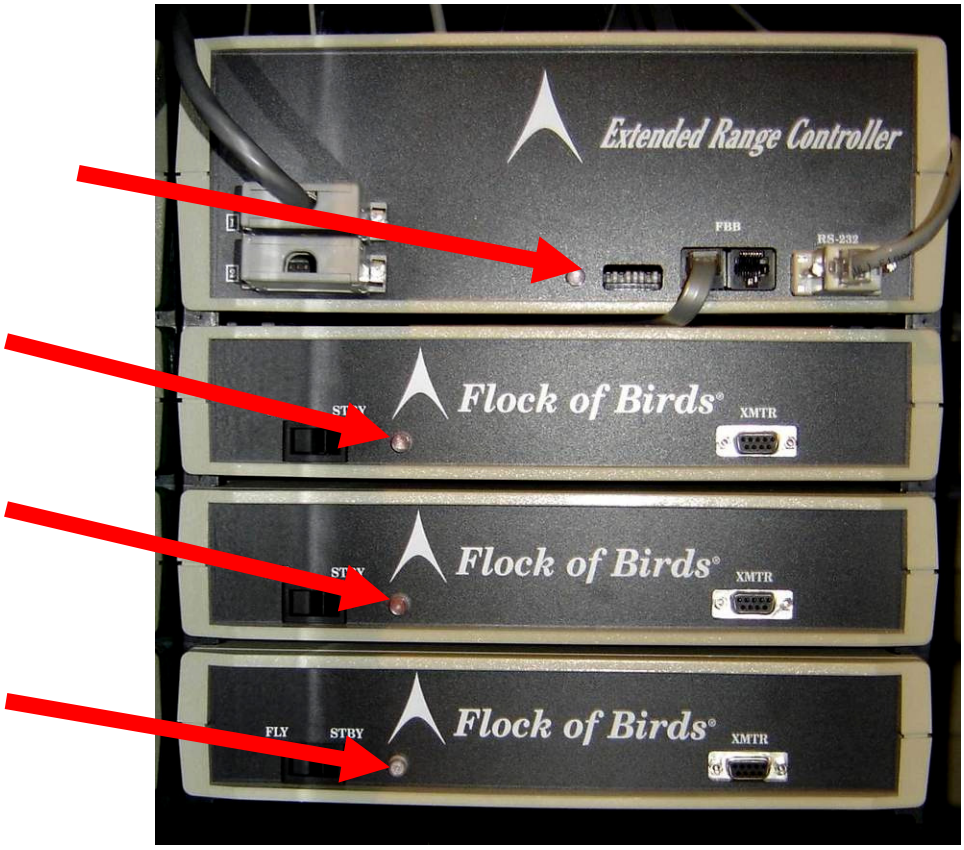

Figure 69: Activation Lights

To start the audio system, the sound amplifiers and mixers should first be turned on.

Figure 70: Switches for the Mixer and Amplifiers

Next, the DSP Switch on the Huron (Figure 71) must be turned on.

Figure 71: DSP Power Button on the Huron PC

The computer must then be logged into. Once logged in, the applications "Socket for NT" and "VRack for NT" must be initialized. These can be found under "Start"→ "Programs"→"Huron3.1." Once "VRack for NT" begins, "D:\Akl\shell_g.wsp" must be loaded into the program.

Once the Huron PC programs have been initialized, a corresponding audio program must be started on the Onyx2. This can be done by using the following commands in a shell (for information on opening a shell terminal, reference section 2.1).

```
cd /disk3/VSS/vss3.1/
./vss
```

This will start the VSS program and bring up the following window.

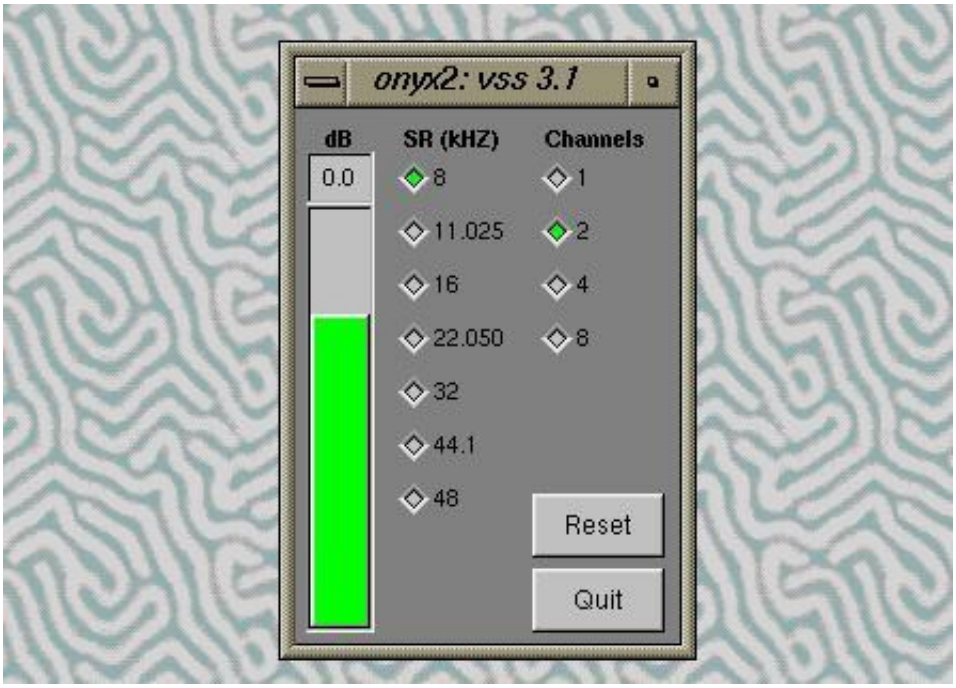

Figure 72: VSS Program

For this system to function correctly, the channel should be set to 2 and the SR should be set to 8kHz.

To run programs with the CyberGlove following command should be used:

```
cd /usr/local/vertex/App*/v*/i*/6*/
./master
```

The remote can be tuned to a specific projector or tunes so that all four projectors will repond to the remote. This option can be toggled by pressing the small projector selection button with a pen followed by the projector number where 1 corresponds to the front wall projector, 2 correspond to the right wall, 3 correspond to the left wall, 4 corresponds to the floor and 0 operates on all projectors. This will turn on projectors ability to accept menu commands.

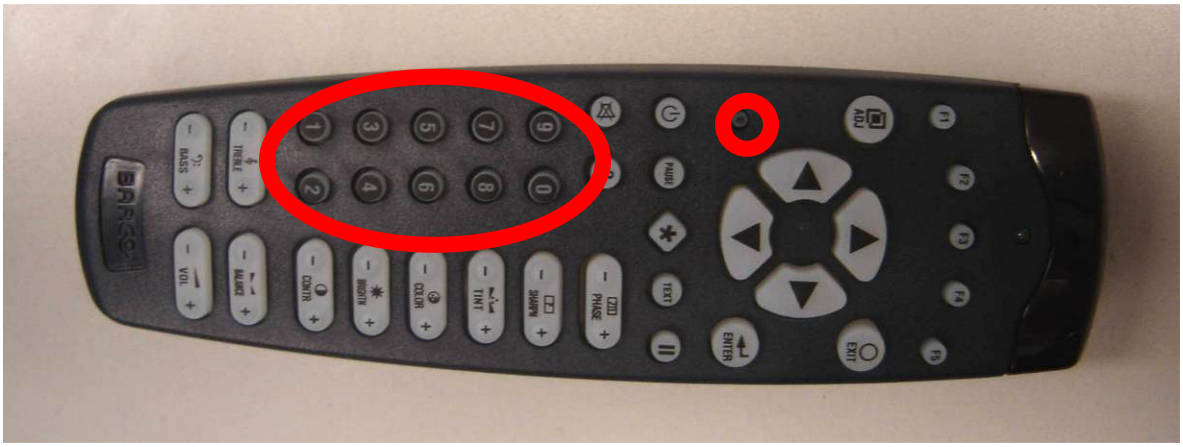

Figure 73: Projector Selection

In most cases the projectors will always be on, but left in a standby mode which saves all the settings but simply keeps the bulbs off as to not wear them out. The status of the projectors can be determined by checking the LED on the side of the projectors. The standby indicator light will show red if the projector is in standby and green if the projector is operational. For proper operation the yellow sync light will also show as well as the red IR indicator.

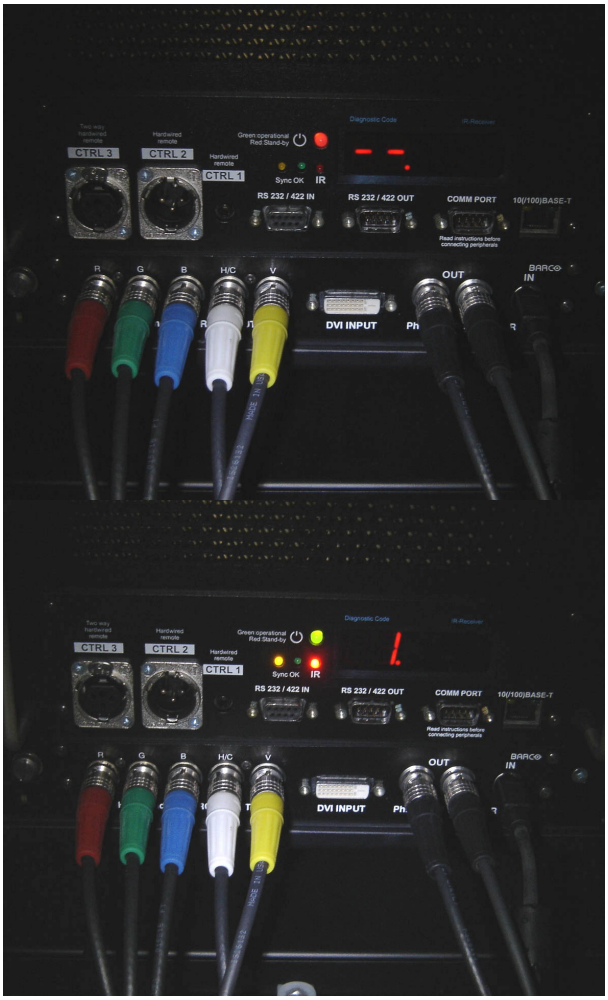

Figure 74: Mode LEDs

Switching in between standby mode can simply be done by pressing and holding the standby button on remote and pointing to each projector. This is not to be confused with the pause button adjacent with the standby button. The pause button simply puts a shutter in front of lens effectively blocking any image; however, the projector would still fully operational.

Figure 75: Standby Button

Figure 76: Toggling Image Shape Settings

The Onyx2 system and CAVE are now fully running and ready to run any CAVE program.

B.2 Basic Onyx2 Use (IRIX OS)

Two of the most useful program in IRIS are the file browser and the shell terminal. The file browser, as the name suggests lets the user browse the file system and manipulate the locations of files. To some degree, the browser also lets the user execute programs as well. To open the file browser, the user simple selects from the desktop toolbox "Desktop"→"Access Files"→"In My Home Directory". This will bring up the browser in the home directory indicated by a single "/". From here, the user can drag and

drop items, perform function by selecting a file(s) and right clicking for options, and execute files by double clicking on a file. Images of how to open the browser and the browser itself can be found below.

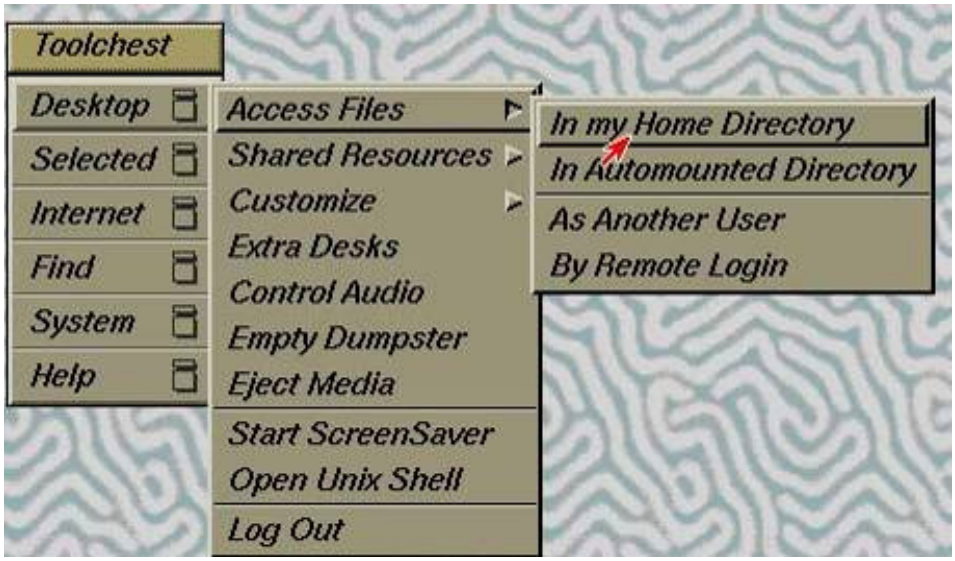

Figure 77: Opening Browser

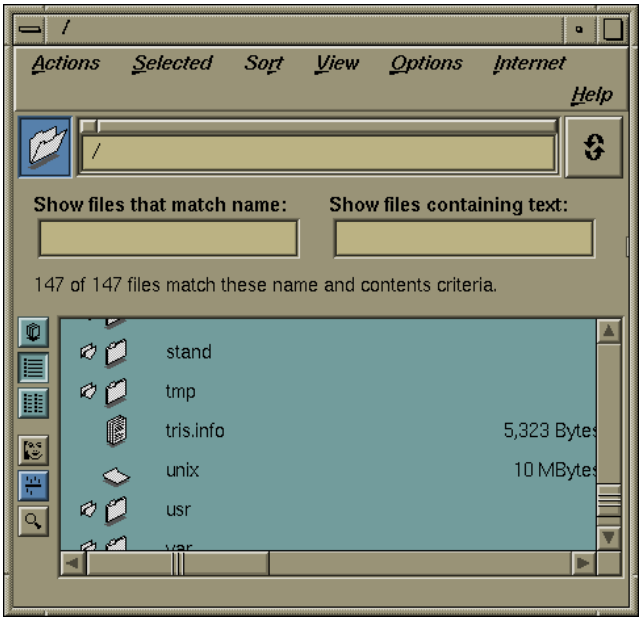

Figure 78: File Browser

The shell terminal allows a user to manipulate and execute programs. Most IRIX users tend to run programs out of a shell rather than through a Graphical User Interface since the shell allows more freedom to choose how the program executes. An example of

this can be seen in Section 2.2 when trying to capture an image. To open a shell, the user simply selects "Desktop"→"Open Unix Shell" from the tool chest. See below for figures.

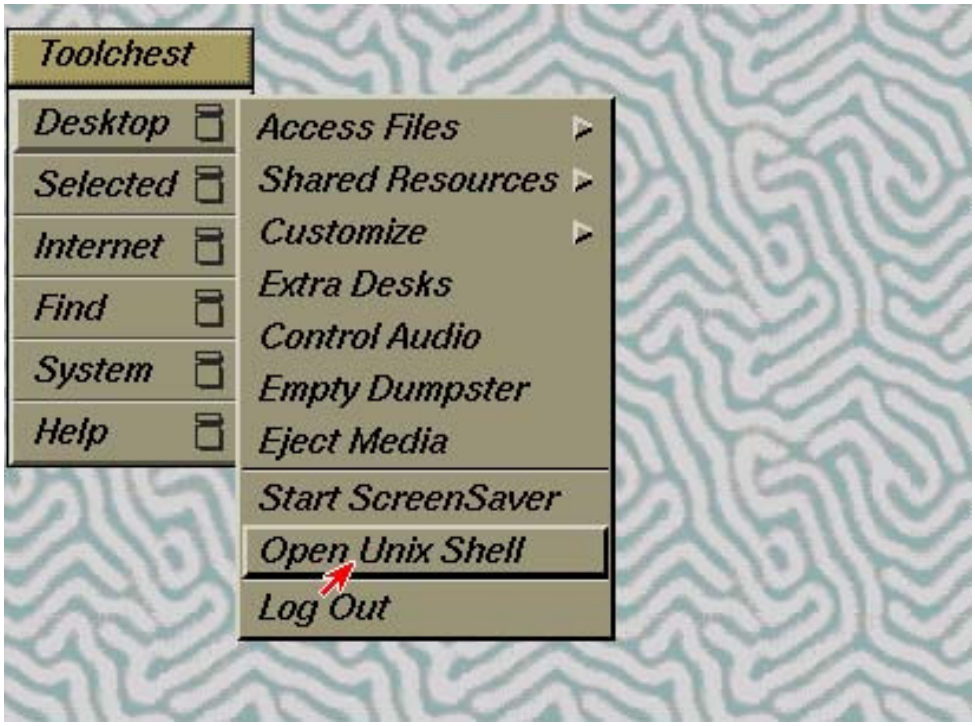

Figure 79: Opening Shell Terminal

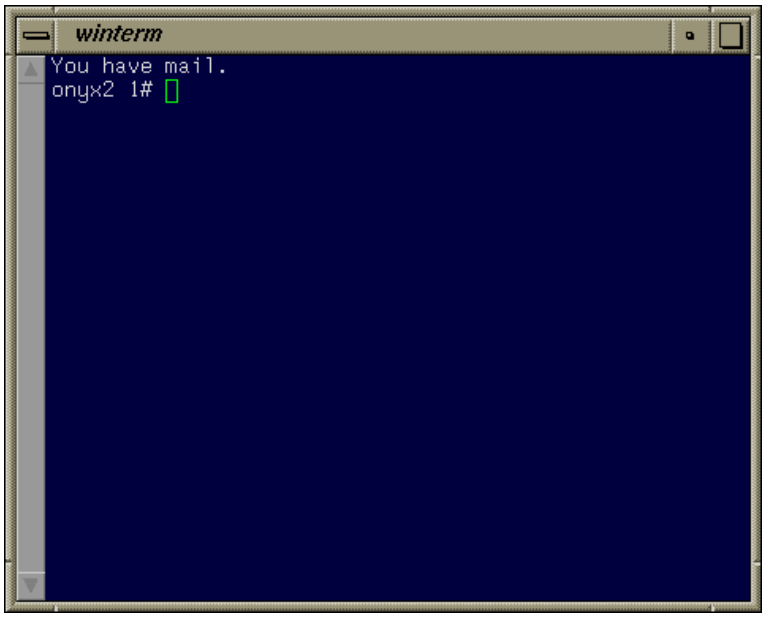

Figure 80: Shell Terminal

Running a basic program from a shell simply involves changing directories to the directory that the program is in and typing in the program preceded by "./". Examples follow:

```
cd /usr/local/jsmoker/Shell/64
./shell

cd /usr/local/waelakl/Helic/
./helic2
```

To run older programs that require the libraries of an older version of Matlab, the following command should be used from the terminal that program will be run from:

```
Setenv LD_LIBRARY64_PATH /usr/local/matlab/extern/lib/sgi64
```

This will allow the programs to use the libraries of Matlab 5.3 rather than 6.5.

B.3 Capturing CAVE Images and Video

When running certain programs, it may become advantageous to capture an image or a video. This can be done with a program called "mediarecorder." "Mediarecorder" can be found under the "MediaTools" tab of the "Icon Catalog."

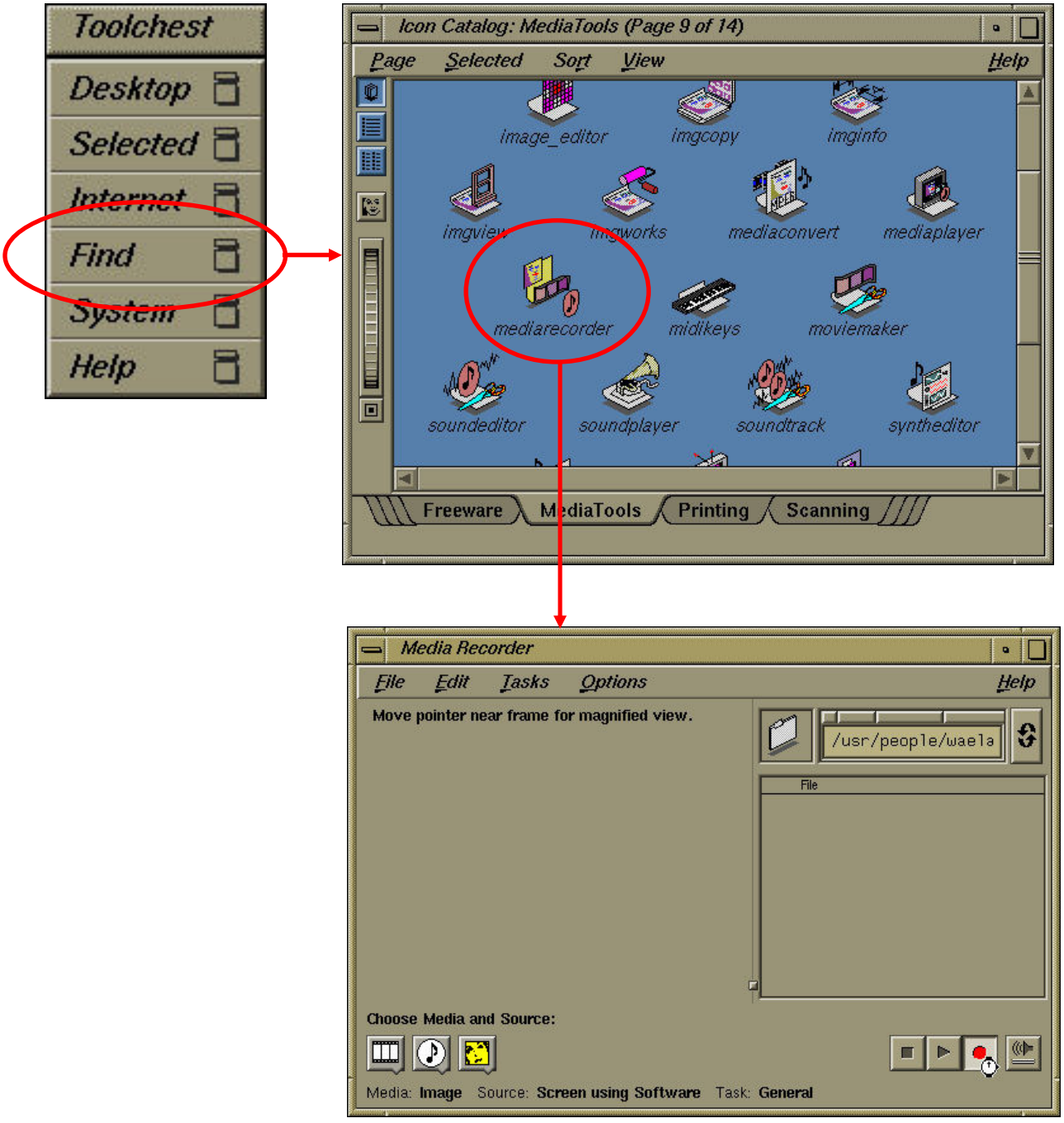

Figure 81: Mediarecorder startup

To capture a picture, a location must first be chosen for the captured file.

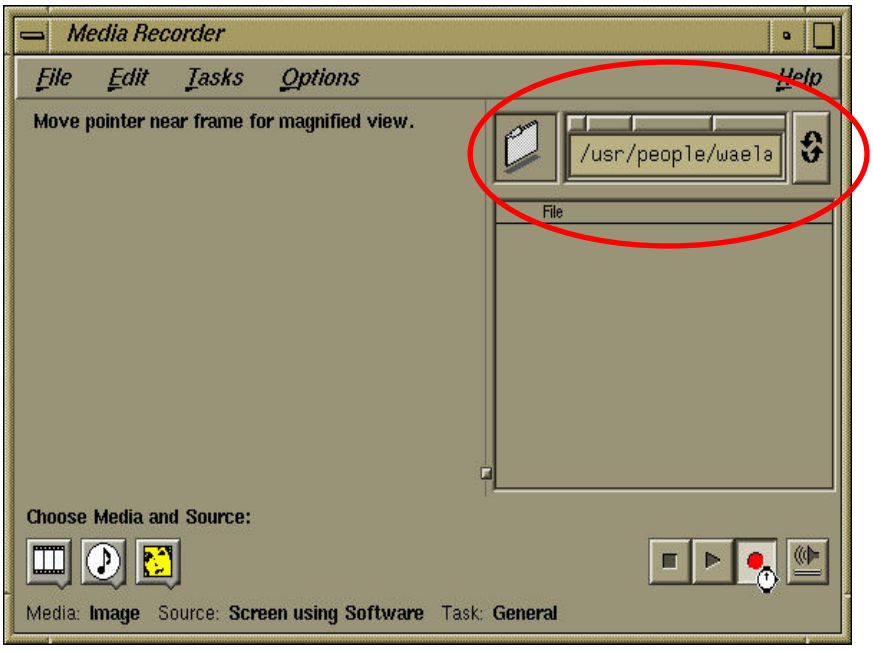

Figure 82: Set capture location

The “capture image” button can then be used to choose the type of capture, and capture area.

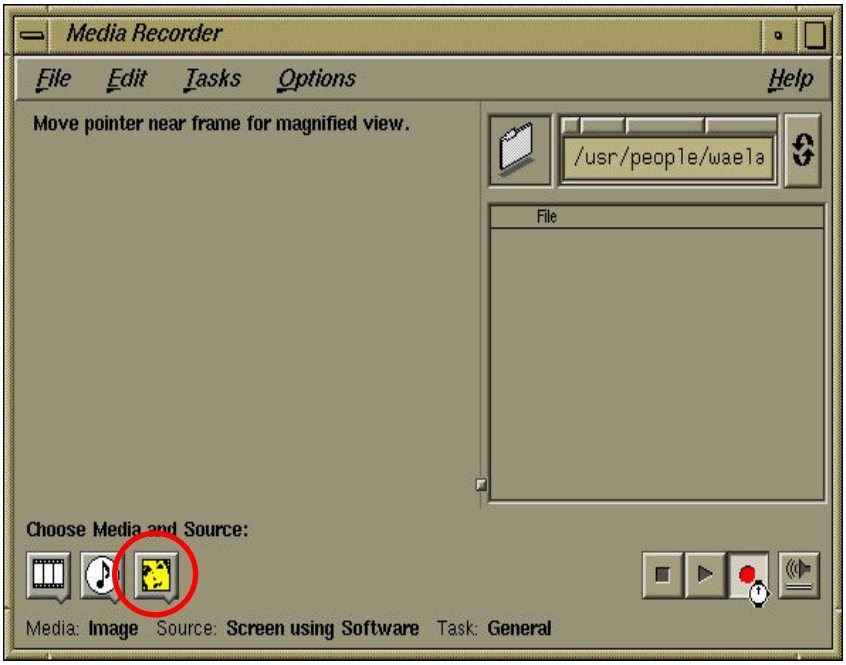

Figure 83: Set type of capture and capture area

The record button can then be depressed to capture the image.

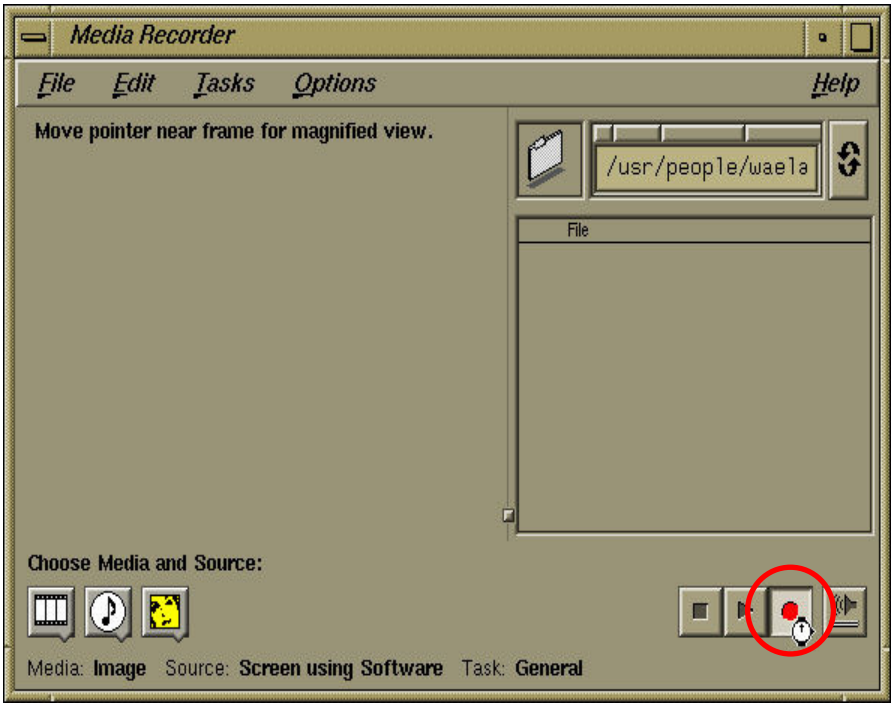

Figure 84: Capture button

The same procedure can be used to capture a video, but instead of using the capture image button, the capture video button must be used instead.

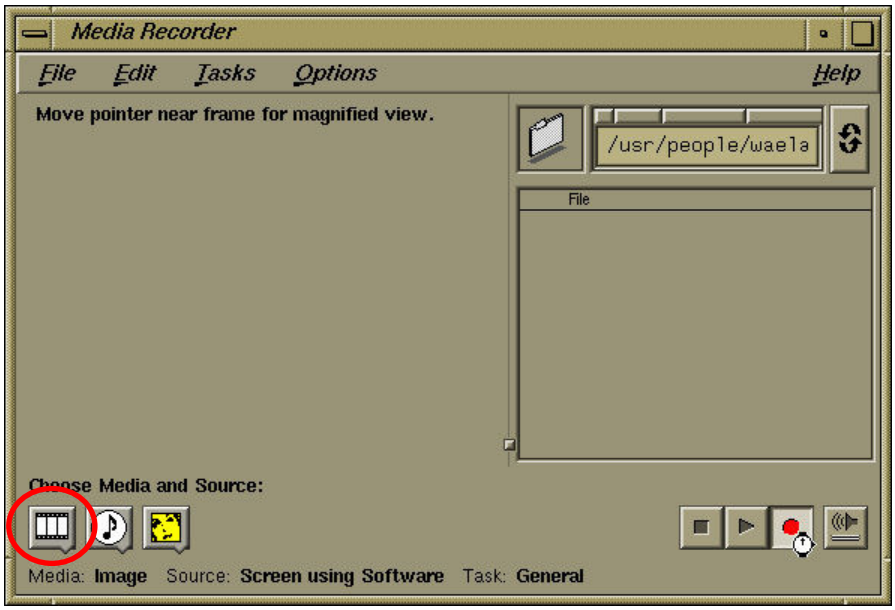

Figure 85: Video button

Also, unlike the capture image mode, the stop button must be used to stop the recording

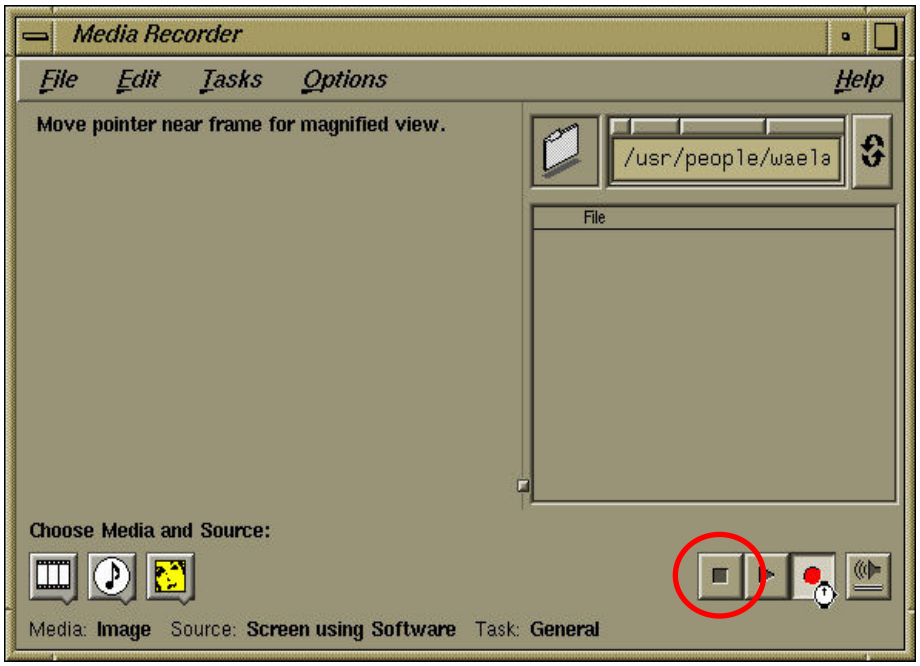

Figure 86: Stop Button

To capture an image or video from a CAVE program, the CAVE mode should be switched from stereo to mono for a single clear image. This can be done by editing the following file:

/usr/local/CAVE/etc/cave.config

Within the file, switch the comment (#) setting as follows:

DisplayMode stereo
#DisplayMode mono

to

#DisplayMode stereo

DisplayMode mono

Another problem that arises is capturing the image while the CAVE program is running. This can be done in two ways. The first way is to simply use the built in time delay within mediarecorder. Here the type of capture and the amount of time to delay the capture and then the delay timer is run while the user opens the program. The negative part of this method is rushing to get the image as wanted for the capture. An alternative is to open mediarecorder from a shell terminal with a delay command. An example of this follows:

(./helic2& ; sleep 5; mediarecorder)&

This will delay the opening of mediarecorder 5 seconds after the program helic2 is run. From that point mediarecorder can be used as if the user was capturing from the desktop. A sample image is shown below.

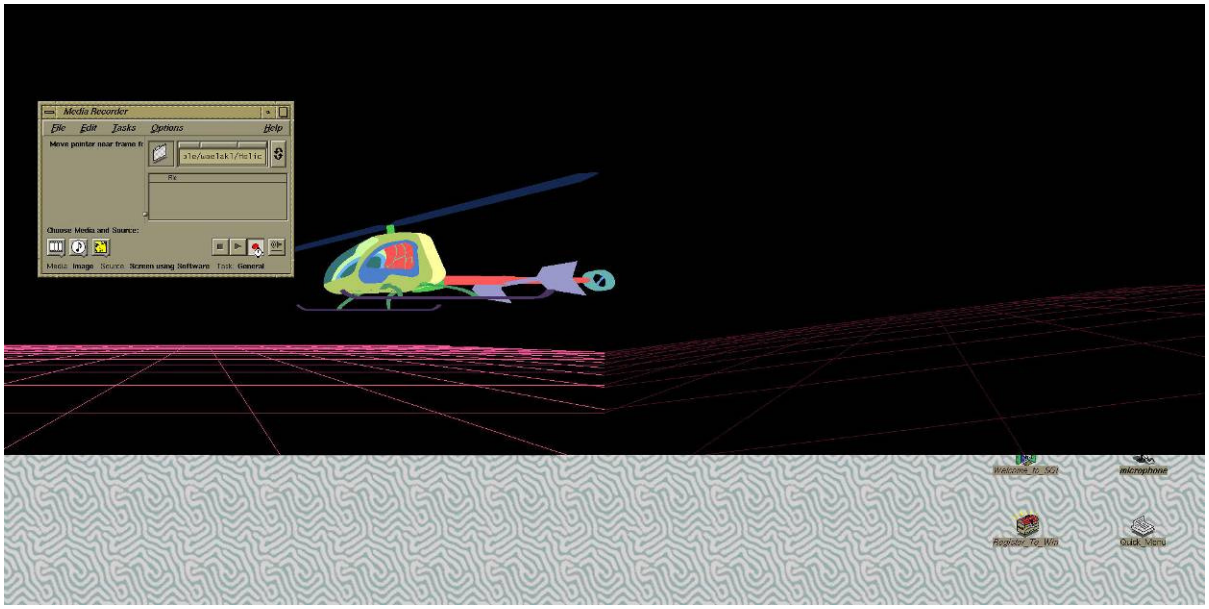

Figure 87: Sample captured image

Appendix C: Preliminary Models

A.1: Quarter Car Model
A.2: Quarter Car Interface
A.3: Half Car Model
A.4: Half Car Interface

C.1 Quarter Car Model

The vehicle model for the quarter car can be seen below in Figure 88. This is a solely one dimensional model with *x* as the output and *y* as the input. The resulting system equation is listed below the model[10,11,41].

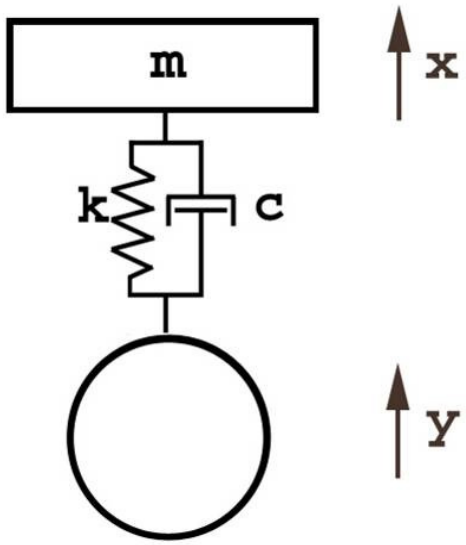

Figure 88: Quarter Car Model

$$m\ddot{x} + c\dot{x} + kx = c\dot{y} + ky \tag{C.1.1}$$

Because of the simplicity of this one dimensional system, the harmonic and impulse forms of the system response were taken in closed form and used in the virtual reality modeling. The harmonic response is shown below where *y* is a simple sine wave input resulting in a cosine output *x* with an amplitude and frequency shift[10].

$$y(t) = Y\sin(\omega t) \tag{C.1.2}$$

$$x_p(t) = Y\left[\frac{k^2 + (cw)^2}{(k - mw)^2 + (cw)^2}\right]^{\frac{1}{2}} \cos(\omega t - \phi_1 - \phi_2) \tag{C.1.3}$$

Where

$$\phi_1 = \tan^{-1}\left(\frac{c\omega}{k - m\omega^2}\right) \qquad \phi_2 = \tan^{-1}\left(\frac{k}{c\omega}\right) \tag{C.1.4}$$

The impulse response, also modeled in closed form, is shown below. Here the input of the wheel is just an instantaneous change for a brief duration, resulting in a decaying sine wave x[10].

$$x(t) = \frac{Fe^{-\xi\omega_n t}}{m\omega_d}\sin(\omega_d t) \tag{C.1.5}$$

Where

$$\omega_n = \sqrt{\frac{k}{m}} \qquad \zeta = \frac{c}{2m\omega_n} \qquad \omega_d = \omega_n \sqrt{1-\zeta^2} \tag{C.1.6}$$

C.2 Quarter Car Interface

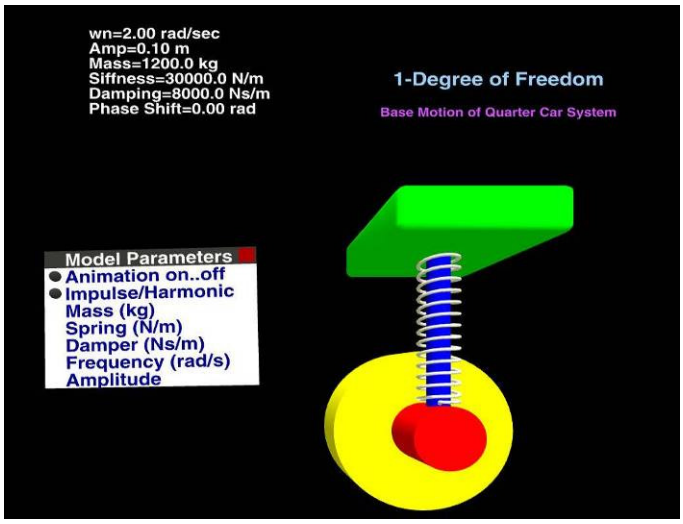

Figure 89: Quarter Car VR Model

The user interface for the virtual reality model allowed the user to change the value of the systems constants, toggle the animation and the type of response desired. As shown above, the constant values and the result phase shift is displayed overhead. Below shows an instance where a value is being changed by the user. The user can make changes in both small and large increments.

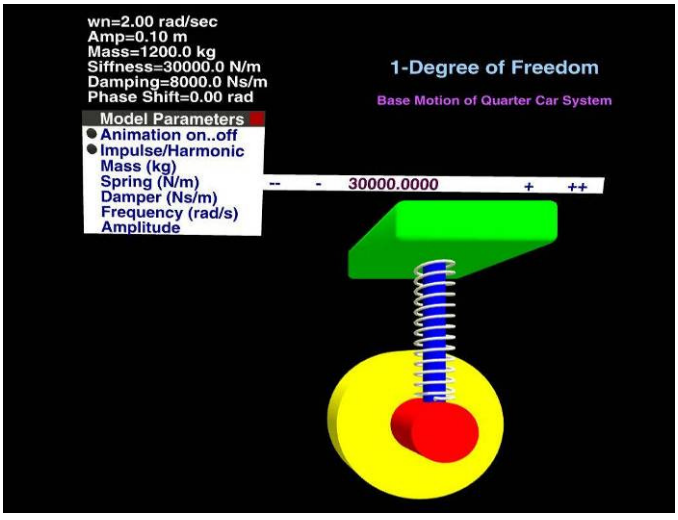

Figure 90: Quarter Car VR Model with value changing

C.3 Half Car Model

The model in Figure 91 is a two dimensional system. The inputs of the system are the displacement of the wheels: y_1 and y_2, and the outputs are the displacement of the mass center and the pitch angle of the mass. All spring constants are equal to each other as are the damper constants. The resulting system equations are shown below Figure 4[1,3, 15,17,42].

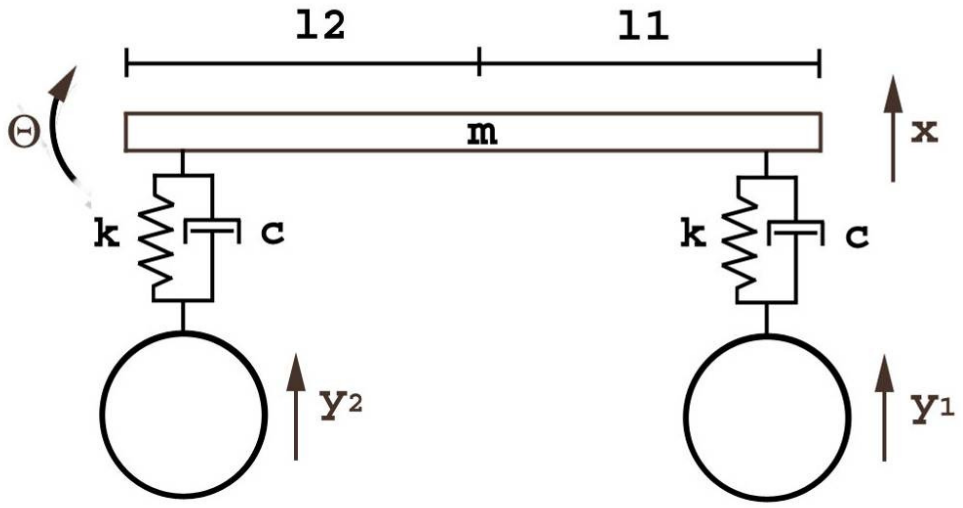

Figure 91: Half Car Model

$$m\ddot{x} + k(x - l_1\theta) + c(\dot{x} - l_1\dot{\theta}) + k(x + l_2\theta) + c(\dot{x} + l_2\dot{\theta}) = ky_1 + c\dot{y}_1 + ky_2 + c\dot{y}_2 \quad \text{(C.3.1)}$$

$$J_c\ddot{\theta} - k(x - l_1\theta)l_1 - c(\dot{x} - l_1\dot{\theta})l_1 + k(x + l_2\theta)l_2 + c(\dot{x} + l_2\dot{\theta})l_2 = -ky_1l_1 - c\dot{y}_1l_1 + k_2y_2l_2 + c\dot{y}_2l_2 \quad \text{(C.3.2)}$$

Unlike the previous model, finding the solution in closed for would be unreasonable. For this case, the solution will be found using a fourth order estimation method, i.e. Runge-Kutta-Fehlberg method[43]. To implement the method, the basic equations are first transformed into a system of equations as described below.

Isolating the Second Derivatives:

$$m\ddot{x} = -(k_1 + k_2)x - (b_1 + b_2)\dot{x} + (k_1 l_1 - k_2 l_2)\theta + (b_1 l_1 - b_2 l_2)\dot{\theta} + ky_1 + c\dot{y}_1 + ky_2 + c\dot{y}_2 \tag{C.3.3}$$

$$J_c\ddot{\theta} = (k_1 l_1 - k_2 l_2)x + (c_1 l_1 - c_2 l_2)\dot{x} - (k_1 l_1^2 + k_2 l_2^2)\theta - (c_1 l_1^2 + c_2 l_2^2)\dot{\theta} - ky_1 l_1 - c\dot{y}_1 l_1 + k_2 y_2 l_2 + c\dot{y}_2 l_2$$

Transform into a Series of Equations:

$$\mathrm{x}_1 = x \qquad \dot{x}_1 = x_2$$

$$x_2 = \dot{x} \qquad \dot{x}_2 = \frac{1}{m}\begin{bmatrix} -(k_1 + k_2)\mathrm{x}_1 - (b_1 + b_2)x_2 \\ +(k_1 l_1 - k_2 l_2)x_3 + (b_1 l_1 - b_2 l_2)x_4 + F_1 \end{bmatrix} \tag{C.3.5}$$

so

$$x_3 = \theta \qquad \dot{x}_3 = x_4$$

$$x_4 = \dot{\theta} \qquad x_4 = \frac{1}{J_c}\begin{bmatrix} (k_1 l_1 - k_2 l_2)\mathrm{x}_1 + (b_1 l_1 - b_2 l_2)x_2 \\ -(k_1 l_1^2 + k_2 l_2^2)x_3 - (c_1 l_1^2 + c_2 l_2^2)x_4 + F_2 \end{bmatrix}$$

Where the input forces are:

$$F_1 = ky_1 + c\dot{y}_1 + ky_2 + c\dot{y}_2 \tag{C.3.6}$$

$$F_2 = -ky_1 l_1 - c\dot{y}_1 l_1 + k_2 y_2 l_2 + c\dot{y}_2 l_2$$

and

$$y_1 = \frac{H}{2}\left(1 - \cos\left(\frac{2\pi V t}{L}\right)\right) \qquad y_2 = \frac{H}{2}\left(1 - \cos\left(\frac{2\pi V t}{L} + \sigma_d\right)\right) \tag{C.3.7}$$

$$\dot{y}_1 = \frac{H\pi V}{L}\sin\left(\frac{2\pi V t}{L}\right) \qquad \dot{y}_2 = \frac{H\pi V}{L}\sin\left(\frac{2\pi V t}{L} + \sigma_d\right)$$

Above, the σ_d makes the assumption that the front and rear wheels are going over the same bump at different times (a shift in phase).

C.4 Half Car Interface

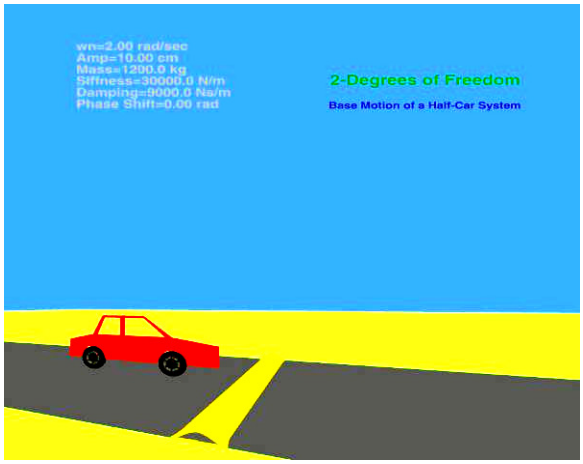

Figure 92: Initial 2-Dimensional Car Model'

The visual model of the car was first created as a two dimensional model as shown above to reflect solely the two degrees of freedom of the system as shown above. However, to improve the virtual reality feel of the interface, the car was remade in three dimensions as shown below. The system remained the same, however. The two front wheels were treats as a point of single input as were the two rear wheels. In this way, roll motion could still be ignored.

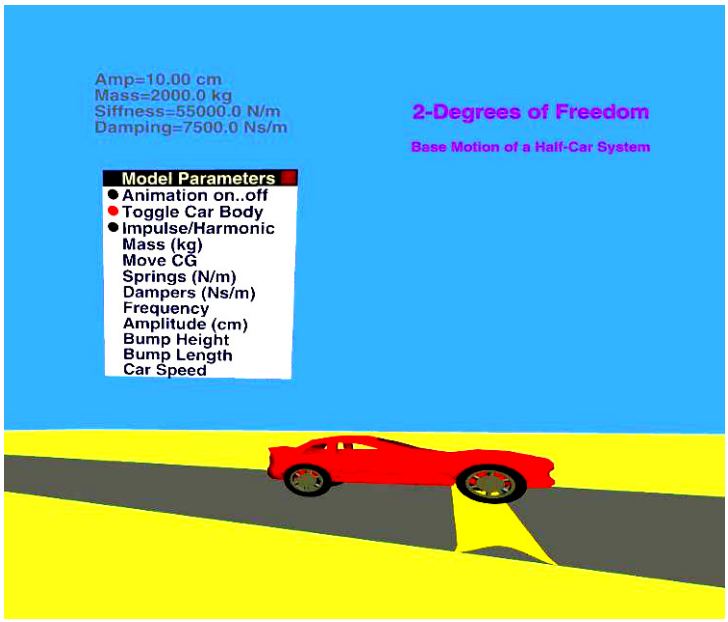

Figure 93: 3-D Car Model in 2-DoF with Menu

The figures below show the different modes in which the simulation can be run. Toggling the car body off allows the adjustable center of gravity to be seen, and also allows a clearer view of the car's response to input. In Figure 94, the bump width and height can be changed by the user with the menu shown. Figure 95 depicts the car in harmonic mode with a regular sine input. The frequency and amplitude of the input can also be changed with same menu as in the impulse mode. Finally, while "Frequency" and "Bump Length" will change the virtual length of a cycle, the "Car Speed" will change the rate at which the car is moving. Therefore, there are essentially two ways to adjust the input frequency in both the impulse and harmonic modes.

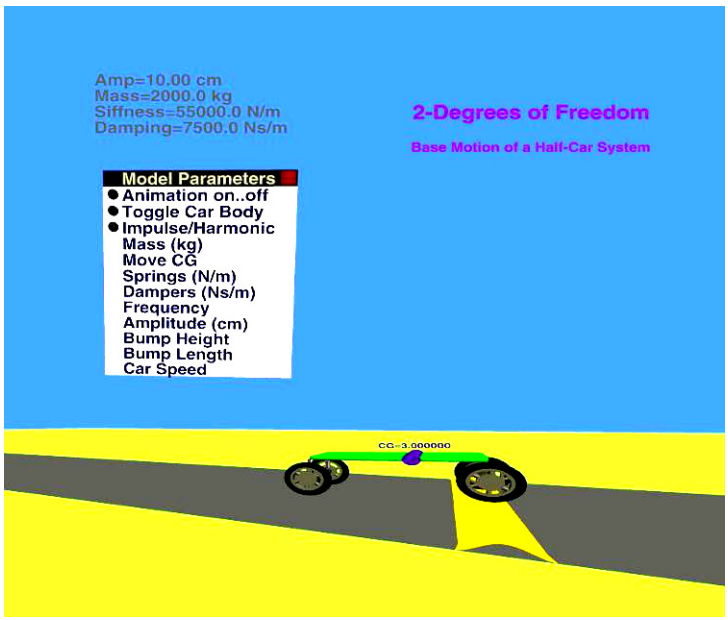

Figure 94: 2-DoF over Bump without Car Body

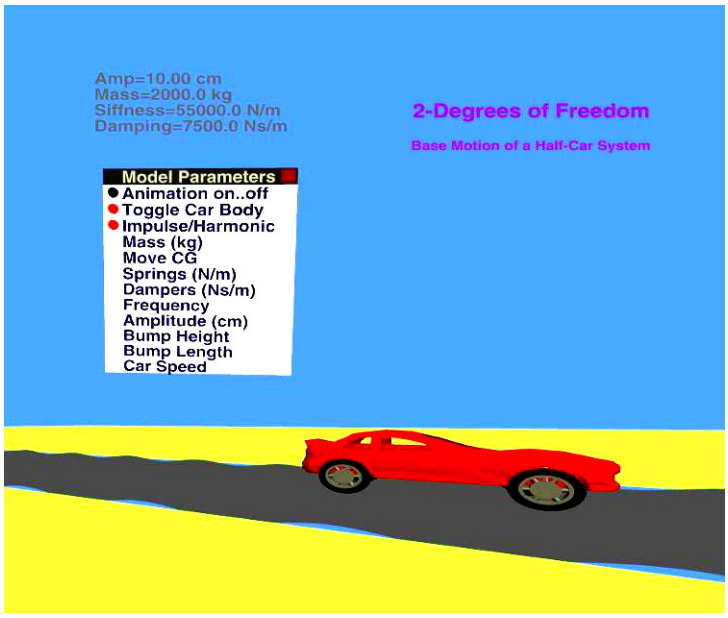

Figure 95:2-DoF over Harmonic Input with Car Body

Appendix D: Main Source Code

There are many files that needed to implement this program but most of the files are general files ported from program to program and are used as an aid for navigation and menu creation. The source code below is heart of the program described in this paper and has been broken into four basic sections: the main loop wherein the program continues to run in real-time, the assignment of variables & scene setup where the environment is setup, the Runge-Kutta-Fehlburg integration used to solve the system, and the kinematics & gain optimization which displays the system and updates it as needed.

D.1: Main Loop

```
//                          fullcar.c++

#include "mtesthead.h"
#define MY_PI 3.141592654
#define MY_E 2.71821828

/*********************** main Function ****************************/
/******************************************************************/

int main(int argc, char **argv)
{
          Initialization();
          pfInit();
          pfCAVEConfig(&argc, argv, NULL);
          shared = (shared_data_t *) pfMalloc(sizeof(shared_data_t),
          pfGetSharedArena());

          // set CAVENear and CAVEFar here, before the display processes are forked
          // NOTE: these are floats, not ints
          CAVENear = 0.1;
          CAVEFar = 65000.0;
          // set multiprocess mode
          pfMultiprocess(PFMP_FORK_COMPUTE);
          pfConfig();
          pfCAVEInitChannels();

          // Navigation Class
          ////////////////////
          nav = new simpleNavigator;
          ////////////////////////////

          if (antialias)
                    graphicsInit();

          shared->master_channel = pfCAVEMasterChan();
          shared->scene = createScene(argc, argv);

          // main loop
          while (! (CAVEgetbutton(CAVE_ESCKEY) || CAVEgetbutton(CAVE_QKEY) || exit_program))
          {
                    pfSync();
                    pfCAVEPreFrame();
                    menu->handle();
                    pfFrame();
                    pfCAVEPostFrame();
          }
          pfCAVEHalt();
          CleaningUp();
          pfExit();
          return 0;

}// main
/******************************************************************/
```

D.2: Assignment of Variables & Scene Setup

```
/************************** Initialization ************************/
void Initialization(void){}// Initialization
/*****************************************************************/

/******************** CleaningUp Function*************************/
void CleaningUp(void){}// CleaningUp
/*****************************************************************/

/********************creatScene Function ***************************/
pfScene *createScene(int argc, char **argv)
{
          font = pfdLoadFont_type1("Helvetica-Bold", PFDFONT_FILLED);
          pfGeoState *gstate = new pfGeoState;
          shared->scene = new pfScene;
          sky = new pfEarthSky;
          sky->setColor(PFES_CLEAR,0.3, 0.5, 0.9, 0.0);
          shared->master_channel->setESky(sky);

          headT_dcs = new pfDCS();
          head1_dcs = new pfDCS();

          grounddcs = new pfDCS();
          roaddcs = new pfDCS();

          HCardcs = new pfDCS();
          Chassisdcs = new pfDCS();

          bumpdcs = new pfDCS();
          Wall1dcs = new pfDCS();
          cg_string= new pfString;
          cg_text = new pfText;
          CGwrddcs = new pfDCS();
          CGshpdcs = new pfDCS();
          CGdcs = new pfDCS();

          waitdcs = new pfDCS();
          wait_string = new pfString;
          wait_text= new pfText;
          Wheel1dcs = new pfDCS();
          WHub1dcs = new pfDCS();
          WHL1dcs = new pfDCS();
          Spring1dcs = new pfDCS();

          Wheel2dcs = new pfDCS();
          WHub2dcs = new pfDCS();
          WHL2dcs = new pfDCS();
          Spring2dcs = new pfDCS();

          Wheel3dcs = new pfDCS();
          WHub3dcs = new pfDCS();
          WHL3dcs = new pfDCS();
          Spring3dcs = new pfDCS();

          Wheel4dcs = new pfDCS();
          WHub4dcs = new pfDCS();
          WHL4dcs = new pfDCS();
          Spring4dcs = new pfDCS();

          nav->init();

          Move = 3;
          k1=35000.0;
          k2=k1;
          k3=k1;
          k4=k1;

          kt=220000.0;

          b1=400.0;
          b2=b1;
          b3=b1;
          b4=b1;

          MASS=1500;
          J1=MASS*5/3;
          J2=MASS/3;
          m1=80;
          m2=80;
          m3=80;
          m4=80;

          freq1 = 1*3.2808;
          ampl = 0.075;
          psh1 = 0.0;
          psh2 = 0.0;
          Bheight = .15;
          Blength = 1;

          springdsf=-(1.0/6.0);

          WHubipos=12*0.12/3.2808+.01;//wheel radius+road height
```

```
WHubisc=0.10*.3/3.2808;

WHLipos=12*0.12/3.2808+.01;//wheel radius+road height
WHLisc=0.12/3.2808;

//wheel radius+road height+just short ofhub radius
Springipos=12*0.12/3.2808+.01+0.25/3.2808;
Springisc=0.35/3.2808;

cg = -0.5;
Spread=15*.3/3.28;
bspd = 2;
WHLDPTH = 8.5*.3/3.28;
WHLDPTH2 = 8.5*.3/3.28;

Wallipos=1.1;
Walliscz=0.2/3.2808;//z
Walliscx=Spread/4.5;//x
Walliscy=.164;//y

//wheel radius+road height+just short ofhub radius
HCaripos=12*0.12/3.2808+.01+0.25/3.2808;
HCarisc=1*.3/3.2808;
Gipos = -10;

gstate->setMode(PFSTATE_SHADEMODEL, PFSM_GOURAUD);
gstate->setMode(PFSTATE_ANTIALIAS, PFAA_ON);
gstate->setMode(PFSTATE_TRANSPARENCY, PFTR_ON);
gstate->setMode(PFSTATE_CULLFACE, PFCF_OFF);
shared->scene->setGState(gstate);

shared->nav_dcs = new pfDCS;
shared->scene->addChild(shared->nav_dcs);

pfMatrix matrix;
matrix.makeRot(0.0, 1.0, 0.0, 0.0);
matrix.preScale(1.0, 1.0, 1.0, matrix);

shared->nav_scs = new pfSCS(matrix);
shared->nav_dcs->addChild(shared->nav_scs);

shared->nav_switch = new pfSwitch;
shared->nav_scs->addChild(shared->nav_switch);

pfLightSource *sun = new pfLightSource;
sun->setColor(PFLT_DIFFUSE, 1.0, 1.0, 0.8);
sun->setColor(PFLT_AMBIENT, 0.15, 0.15, 0.05);
sun->setPos(.3945, -0.65, 0.65, 0.0);
sun->setMode(PFLS_SHADOW_ENABLE, PF_ON);
sun->setVal(PFLS_INTENSITY, .6f);
shared->nav_dcs->addChild(sun);

shared->scene->setTravMask(PFTRAV_ISECT, 0xffffffff,
PFTRAV_SELF | PFTRAV_DESCEND | PFTRAV_IS_CACHE, PF_OR);

pfFilePath(path);

CarOut = new grapher(shared->scene, gnodes, 0.0, 0.0 ,-.1);
CarOut->reset(gnodes);
CarOut->limit = 0;

CarOut2 = new grapher(shared->scene, gnodes, 0.0, 0.0 , .4);
CarOut2->textChoice = 2;//1;
CarOut2->reset(gnodes);
CarOut2->limit = 0;
CarOut2->graphColor(0,.3,.3);

// INIT MENU
////////////

pfMenuGeometry *geometry = new pfMenuGeometry;
menu = new pfMenu(shared->scene, geometry, "Model Parameters");

pfMenuItem *new_item;

new_item = new pfMenuToggleItem(geometry, mn_an_st_toggle_cbf, "Animation on..off", 0);
menu->addItem(new_item);

new_item = new pfMenuToggleItem(geometry, mn_mode_toggle_cbf, "Impulse/Harmonic", 0);
menu->addItem(new_item);

static const char *button_p_text[] = {
"Car Body", "Chassis", "CG - View" };

new_item = new pfMenuRadioItem(geometry, mn_carbody_value_cbf, "Toggle Car Body",button_p_text, 3);
menu->addItem(new_item);

static const char *button_p_text2[] = {
"Passive Only", "LQR Control", "NEA Control" };

new_item = new pfMenuRadioItem(geometry, mn_Control_value_cbf, "Control Mode-->",button_p_text2, 3);

menu->addItem(new_item);

new_item = new pfMenuValueItem(geometry, mn_WHub_value_cbf, "Mass (kg)", MASS);
```

```
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_cg_value_cbf, "Move CG", cg);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_stiff1_value_cbf, "Springs (N/m)", k1);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_damp_value_cbf, "Dampers (Ns/m)",b1);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_ampl_value_cbf, "Amplitude (cm)", ampl*100);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_freq1_value_cbf, "Frequency (cycle/m)", freq1/3.2808);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_Bheight_value_cbf, "Bump Height (cm)", Bheight*100);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_Blength_value_cbf, "Bump Length (cm)", Blength*100);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_speed_value_cbf, "Car Speed (m/s)", Move);
        menu->addItem(new_item);

        new_item = new pfMenuValueItem(geometry, mn_slant_value_cbf, "Bump Slant(s) (m)", bspd);
        menu->addItem(new_item);

        DataUpdate();

        if (dump_scene)
        {
        FILE *out_file = fopen("shared->scene.out", "w");
        pfMemory::print(shared->scene, PFTRAV_SELF | PFTRAV_DESCEND,
          PFPRINT_VB_DEBUG, out_file);
        fclose(out_file);
        }

        // For Springs and Masses Loading
        ////////////////////////////////

        WHubcolor= (pfVec4 *) pfMalloc(sizeof(pfVec4),pfGetSharedArena());
        CGcolor= (pfVec4 *) pfMalloc(sizeof(pfVec4),pfGetSharedArena());

        WHubcolor->set(0.5,0.5,0.5,1);
        CGcolor->set(0.5,0,1,1);

/*****************************Wheels***********************************/
        WHubgeode = new pfGeode();
        WHubgeode = createCylinder(6,WHubcolor);
        WHLgeode = new pfGeode();
        WHLgeode =(pfGeode *) pfdLoadFile("wheel.iv");

        WHub1dcs->addChild(WHubgeode);
        WHL1dcs->addChild(WHLgeode);
        Wheel1dcs->addChild(WHub1dcs);
        Wheel1dcs->addChild(WHL1dcs);
        head1_dcs->addChild(Wheel1dcs);

        WHub2dcs->addChild(WHubgeode);
        WHL2dcs->addChild(WHLgeode);
        Wheel2dcs->addChild(WHub2dcs);
        Wheel2dcs->addChild(WHL2dcs);
        head1_dcs->addChild(Wheel2dcs);

        WHub3dcs->addChild(WHubgeode);
        WHL3dcs->addChild(WHLgeode);
        Wheel3dcs->addChild(WHub3dcs);
        Wheel3dcs->addChild(WHL3dcs);
        head1_dcs->addChild(Wheel3dcs);

        WHub4dcs->addChild(WHubgeode);
        WHL4dcs->addChild(WHLgeode);
        Wheel4dcs->addChild(WHub4dcs);
        Wheel4dcs->addChild(WHL4dcs);
        head1_dcs->addChild(Wheel4dcs);

/************************Shocks and Struts*********************************/

        Springgeode = new pfGeode();
        Springgeode = (pfGeode *) pfdLoadFile("spring2.iv");
        Spring1dcs->addChild(Springgeode);
        head1_dcs->addChild(Spring1dcs);

        Spring2dcs->addChild(Springgeode);
        head1_dcs->addChild(Spring2dcs);

        Spring3dcs->addChild(Springgeode);
        head1_dcs->addChild(Spring3dcs);

        Spring4dcs->addChild(Springgeode);
        head1_dcs->addChild(Spring4dcs);
/***********************Center of Gravity Body**************************/
```

```
          Wallgeode = (pfGeode *) pfdLoadFile("wall_1.iv");
          Wall1dcs->addChild(Wallgeode);
          head1_dcs->addChild(Wall1dcs);
          Wall1dcs->setTravMask(PFTRAV_ISECT, 0xffffffff,
                       PFTRAV_SELF|PFTRAV_DESCEND|PFTRAV_IS_CACHE, PF_OR);
          Wall1dcs->setTravFuncs(PFTRAV_APP,BeamMotion,NULL);

/********************Center of Gravity Marker************************/

          CGgeode = createCylinder(2,CGcolor);
          cg_string->setFont(font);
          cg_string->setMode(PFSTR_JUSTIFY, PFSTR_CENTER);
          cg_string->setColor(0.5, 0.0, 1, 1.0);
          sprintf(cgChar,"CG=%3.2f m", -cg);
          cg_string->setString(cgChar);
          cg_text->addString(cg_string);
          CGwrddcs->addChild(cg_text);
          CGshpdcs->addChild(CGgeode);
          CGdcs->addChild(CGwrddcs);
          CGdcs->addChild(CGshpdcs);
          head1_dcs->addChild(CGdcs);

/********************Control Wait Request************************/

          wait_string->setFont(font);
          wait_string->setMode(PFSTR_JUSTIFY, PFSTR_CENTER);
          wait_string->setColor(0.7, 0.3, 0.3, 1.0);
          sprintf(waitChar,"Please Wait for Calculations");
          wait_string->setString(waitChar);
          wait_text->addString(wait_string);
          waitdcs->addChild(wait_text);
          shared->nav_dcs->addChild(waitdcs);

/*****************************GROUND*********************************/
          groundgeode = (pfGeode *) pfdLoadFile("ground.iv");
          grounddcs->addChild(groundgeode);
          //grounddcs->setTrans(-10/3.28,0,0);//z,x,y
          head1_dcs->addChild(grounddcs);

/*****************************ROAD***********************************/

          roadgeode = (pfGeode *) pfdLoadFile("road.iv");
          roaddcs->addChild(roadgeode);
          //roaddcs->setTrans(-10/3.28,0,0);//z,x,y
          head1_dcs->addChild(roaddcs);

/*****************************CARBODY*********************************/
          HCargeode = (pfGeode *) pfdLoadFile("car2.iv");
          HCardcs->addChild(HCargeode);
          head1_dcs->addChild(HCardcs);
          Chassisgeode = (pfGeode *) pfdLoadFile("chassis.iv");
          Chassisdcs->addChild(Chassisgeode);
          head1_dcs->addChild(Chassisdcs);
/*****************************SPEEDBUMP*********************************/

          bumpgeode = new pfGeode();
          bumpgeode = bump(Bheight,Blength);
          bumpdcs->addChild(bumpgeode);
          head1_dcs->addChild(bumpdcs);
          bumpdcs->setTravFuncs(PFTRAV_APP,BumpUpdate,NULL);

/********************Initial Scales and Transitions************************/

          CGwrddcs->setScale(2/3.2808);
          CGwrddcs->setTrans(2.5/3.2808,-cg,0);
          CGwrddcs->setRot(0 , 90, 90);
          CGshpdcs->setTrans(0,-cg,0);
          CGshpdcs->setScale(0.25/3.2808,0.25/3.2808,1.4/3.2808);
          CGdcs->setTrans(Wallipos+0.3/3.2808,Gipos,0);
          CGdcs->setScale(0);
          Wall1dcs->setTrans(Wallipos,Spread+Gipos+0.1,-WHLDPTH*1.05);
          Wall1dcs->setScale(0);

          waitdcs->setTrans(0,15,4);
          waitdcs->setScale(0);

          HCardcs->setRot(90 , 0, 90);
          HCardcs->setTrans(HCaripos,Gipos,0);
          HCardcs->setScale(HCarisc);

          Chassisdcs->setRot(90 , 0, 90);
          Chassisdcs->setTrans(HCaripos,Gipos,0);
          Chassisdcs->setScale(HCarisc);

          WHub1dcs->setScale(WHubisc);
          WHub1dcs->setRot(0,-90,0);
          WHub1dcs->setTrans(0,0.7*.3/3.28,0);
          WHL1dcs->setScale(WHLisc);
          Wheel1dcs->setTrans(WHubipos,Spread+Gipos,-WHLDPTH);
          Wheel1dcs->setRot(0,90,0);
          Spring1dcs->setScale(Springisc);
          Spring1dcs->setTrans(Springipos,Spread+Gipos,1.2*.3/3.28-WHLDPTH);
          Spring1dcs->setRot(90,0,90);

          WHub2dcs->setScale(WHubisc);
```

```
          WHub2dcs->setRot(0,-90,0);
          WHub2dcs->setTrans(0,0.7*.3/3.28,0);
          WHL2dcs->setScale(WHLisc);
          Wheel2dcs->setTrans(WHubipos,-Spread+Gipos,-WHLDPTH-1*.3/3.28);
          Wheel2dcs->setRot(0,90,0);
          Spring2dcs->setScale(Springisc);
          Spring2dcs->setTrans(Springipos,-Spread+Gipos,0.2*.3/3.28-WHLDPTH);
          Spring2dcs->setRot(90,0,90);

          WHub3dcs->setScale(WHubisc);
          WHub3dcs->setRot(0,-90,0);
          WHub3dcs->setTrans(0,0.7*.3/3.28,0);
          WHL3dcs->setScale(WHLisc);
          Wheel3dcs->setTrans(WHubipos,Spread+Gipos,WHLDPTH2);
          Wheel3dcs->setRot(0,-90,0);
          Spring3dcs->setScale(Springisc);
          Spring3dcs->setTrans(Springipos,Spread+Gipos,WHLDPTH2-1.2*.3/3.28);
          Spring3dcs->setRot(90,0,90);


          WHub4dcs->setScale(WHubisc);
          WHub4dcs->setRot(0,-90,0);
          WHub4dcs->setTrans(0,0.7*.3/3.28,0);
          WHL4dcs->setScale(WHLisc);
          Wheel4dcs->setTrans(WHubipos,-Spread+Gipos,WHLDPTH2+1*.3/3.28);
          Wheel4dcs->setRot(0,-90,0);
          Spring4dcs->setScale(Springisc);
          Spring4dcs->setTrans(Springipos,-Spread+Gipos,WHLDPTH2-0.2*.3/3.28);
          Spring4dcs->setRot(90,0,90);

          Node1dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5);
          Node2dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5+1);
          Node3dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5+2);

          Node4dcs->setTrans((Spread+Gipos+position)*0.4,30+0.3*(-WHLDPTH),0.3*(WHLipos+y01)-5);
          Node5dcs->setTrans((-Spread+Gipos+position)*0.4,30+0.3*(-WHLDPTH),0.3*(WHLipos+y02)-5);
          Node6dcs->setTrans((Spread+Gipos+position)*0.4,30+0.3*(WHLDPTH),0.3*(WHLipos+y03)-5);
          Node7dcs->setTrans((-Spread+Gipos+position)*0.4,30+0.3*(WHLDPTH),0.3*(WHLipos+y04)-5);

          CarOut->scale = 0.1;
          CarOut2->scale = 0.1;

          head1_dcs->setRot(270,0,-90);
          head1_dcs->setScale(3.2808, 3.2808, 3.2808);
          head1_dcs->setTrans(-0,40,0); //x,y,z

          ScreenText();

          shared->scene->addChild(freq1_scs);
          shared->scene->addChild(WHub_scs);
          shared->scene->addChild(stiff1_scs);
          shared->scene->addChild(damp_scs);
          shared->scene->addChild(ampl_scs);
          shared->scene->addChild(psh_scs);

          /***************** Initialize Matricies*********************/
          initMatrices();
          /************************************************************/

          headT_dcs->addChild(head1_dcs);
          nav->addChild(headT_dcs);
          shared->scene->addChild(nav);

          /************************************************************/

   shared->master_channel->setScene(shared->scene);

   return shared->scene;
}// createScene
/*********************************************************************/



/********************** graphicsInit Function *******************/
void graphicsInit(void)
{
  int my_fb_config[] =
  {
    PFFB_LEVEL, 0,
    PFFB_RGBA,
    PFFB_DOUBLEBUFFER,
    PFFB_STEREO,
    PFFB_SAMPLE_BUFFER, 1,
    PFFB_DEPTH_SIZE, 15,
    PFFB_STENCIL_SIZE, 1,
    PFFB_RED_SIZE, 10,
    PFFB_GREEN_SIZE, 10,
    PFFB_BLUE_SIZE, 10,
    PFFB_SAMPLES, 8,
    None
  };

  int num_pipes = pfGetMultipipe();

  for (int i = 0; i < num_pipes; i++)
```

```
  {
    pfPipe *p = pfGetPipe(i);
    pfPipeWindow *pw = p->getPWin(0);
    XVisualInfo *valid_config = pw->chooseFBConfig(my_fb_config);
    pw->setFBConfigId(valid_config->visualid);
  }
}//graphicsInit
/*****************************************************************/

/******************* Menu Functions *************************/
/************************************************************/
/************************************************************/

static void mn_an_st_toggle_cbf(pfMenuItem *, int toggle)
{
          // toggle item callback

          if(AnimIndex)
          {
                    AnimIndex=toggle;
          }
          else
          {
                    if(((ControlMode == 1) && ParameterChange1) || ((ControlMode == 2) && ParameterChange2) ||
((ControlMode == 3) && ParameterChange3))
                              wait2go = 1;
                    else
                              AnimIndex=toggle;

                    precalc();
                    Calculations();
          }
          sprintf(WHubChar,"Mass=%1.1f kg",MASS);
          WHub_string->setString(WHubChar);

          sprintf(stiff1Char,"Siffness=%1.1f N/m",k1);
          stiff1_string->setString(stiff1Char);

          sprintf(dampChar,"Damping=%1.1f Ns/m",b1);
          damp_string->setString(dampChar);




}
static void mn_carbody_value_cbf(pfMenuItem *, int button)
{
          // toggle item callback


          switch(button)
          {
                    case 1:
                              //carbody = 1;
                              Chassisdcs->setScale(HCarisc);
                              HCardcs->setScale(HCarisc);
                              Wall1dcs->setScale(0);
                              CGdcs->setScale(0);
                              break;

                    case 2:
                              //carbody = 2;
                              Chassisdcs->setScale(HCarisc);
                              HCardcs->setScale(0);
                              Wall1dcs->setScale(0);
                              CGdcs->setScale(0);
                              break;

                    case 3:
                              //carbody = 3;
                              Chassisdcs->setScale(0);
                              HCardcs->setScale(0);
                              Wall1dcs->setScale(Walliscz, Walliscx, Walliscy);
                              CGdcs->setScale(1);
                              break;
          }
}

static void mn_mode_toggle_cbf(pfMenuItem *, int toggle)
{
          // toggle item callback
          mode=toggle;
          Calculations();

          if(mode)
          {
                    sprintf(freq1Char,"Amp=%1.2f cm",ampl/10);
                    ampl_string->setString(amplChar);

                    sprintf(WHubChar,"Mass=%1.1f kg",MASS);
                    WHub_string->setString(WHubChar);

                    sprintf(stiff1Char,"Siffness=%1.1f N/m",k1);
                    stiff1_string->setString(stiff1Char);
```

```
                    sprintf(dampChar,"Damping=%1.1f Ns/m",b1);
                    damp_string->setString(dampChar);

          }
          bumpchange = 1;

}

static void mn_Control_value_cbf(pfMenuItem *, int button)
{
          // radio item callback
          switch(button)
          {
                    case 1:
                              ControlMode = 1;
                              sprintf(freq1Char,"Control = OFF");
                       freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.7, 0.5, 0.5, 1.0);
                              break;

                    case 2:
                              ControlMode = 2;
                              sprintf(freq1Char,"Control = LQR");
                       freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.7, 0.3, 0.3, 1.0);
                              break;

                    case 3:
                              ControlMode = 3;
                              sprintf(freq1Char,"Control = NEA");
                       freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.3, 0.7, 0.3, 1.0);
                              break;
          }

}
//car speed
void mn_WHub_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback

          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 1000;
                              break;
                    case MN_NRML_DEC:
                              value -= 100;
                              break;
                    case MN_NRML_INC:
                              value += 100;
                              break;
                    case MN_FAST_INC:
                              value += 1000;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);

          MASS=value;
          J1 = 5*MASS/3;
          J2 = MASS/3;
          ParameterChange1 = 1;
          ParameterChange2 = 1;
          ParameterChange3 = 1;

          precalc();
          Calculations();

          sprintf(WHubChar,"Mass=%1.1f kg",MASS);
          WHub_string->setString(WHubChar);

}

void mn_cg_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback

          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
          case MN_FAST_DEC:
                    value -= 0.5;
                    break;
          case MN_NRML_DEC:
                    value -= 0.1;
```

```
                break;
        case MN_NRML_INC:
                value += 0.1;
                break;
        case MN_FAST_INC:
                value += 0.5;
                break;
        }

        // set new value
        ((pfMenuValueItem *) item_ptr)->setValue(value);

        cg=value;
        CGshpdcs->setTrans(0,-cg,0);
        CGwrddcs->setTrans(2.5,-cg,0);
        ParameterChange1 = 1;
        ParameterChange2 = 1;
        ParameterChange3 = 1;
        precalc();
        Calculations();

        sprintf(cgChar,"CG=%3.2f m", -cg);
        cg_string->setString(cgChar);
}


void mn_stiff1_value_cbf(pfMenuItem *item_ptr, int action)
{
        // value item callback
        // get current value
        float value = ((pfMenuValueItem *) item_ptr)->getValue();

        // modify value based on user action
        switch (action)
        {
                case MN_FAST_DEC:
                        value -= 5000.0;
                        break;
                case MN_NRML_DEC:
                        value -= 500.0;
                        break;
                case MN_NRML_INC:
                        value += 500.0;
                        break;
                case MN_FAST_INC:
                        value += 5000.0;
                        break;
        }

        // set new value
        ((pfMenuValueItem *) item_ptr)->setValue(value);
        k1=value;
        k2=value;
        k3=value;
        k4=value;
        ParameterChange1 = 1;
        ParameterChange2 = 1;
        ParameterChange3 = 1;
        precalc();
        Calculations();

        sprintf(stiff1Char,"Stiffness1=%1.1f N/m2",k1);
        stiff1_string->setString(stiff1Char);
}

static void mn_damp_value_cbf(pfMenuItem *item_ptr, int action)
{
        // value item callback
        // get current value
        float value = ((pfMenuValueItem *) item_ptr)->getValue();

        // modify value based on user action
        switch (action)
        {
                case MN_FAST_DEC:
                        value -= 100.0;
                        break;
                case MN_NRML_DEC:
                        value -= 50.0;
                        break;
                case MN_NRML_INC:
                        value += 50.0;
                        break;
                case MN_FAST_INC:
                        value += 100.0;
                        break;
        }

        // set new value
        ((pfMenuValueItem *) item_ptr)->setValue(value);
        b1=value;
        b2=value;
        b3=value;
        b4=value;
        ParameterChange1 = 1;
```

```
          ParameterChange2 = 1;
          ParameterChange3 = 1;
          precalc();
          Calculations();


          sprintf(dampChar,"Damping=%1.1f N/m2",b1);
          damp_string->setString(dampChar);

}

static void mn_freq1_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 0.5;
                              break;
                    case MN_NRML_DEC:
                              value -= 0.05;
                              break;
                    case MN_NRML_INC:
                              value += 0.05;
                              break;
                    case MN_FAST_INC:
                              value += 0.5;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          bumpchange = 1;
          freq1=value*3.2808;//convert from feet to meters
          Calculations();
}

static void mn_ampl_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 5;
                              break;
                    case MN_NRML_DEC:
                              value -= 1;
                              break;
                    case MN_NRML_INC:
                              value += 1;
                              break;
                    case MN_FAST_INC:
                              value += 5;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          bumpchange = 1;
          ampl=value/100;
          Calculations();
}


static void mn_Bheight_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 10.0;
                              break;
                    case MN_NRML_DEC:
                              value -= 1.0;
                              break;
                    case MN_NRML_INC:
                              value += 1.0;
                              break;
                    case MN_FAST_INC:
                              value += 10.0;
                              break;
          }
```

```
          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          bumpchange = 1;
          Bheight=value/100;
          Calculations();
}
static void mn_Blength_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 10.0;
                              break;
                    case MN_NRML_DEC:
                              value -= 2.0;
                              break;
                    case MN_NRML_INC:
                              value += 2.0;
                              break;
                    case MN_FAST_INC:
                              value += 10.0;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          bumpchange = 1;
          Blength=value/100;
          Calculations();
}


static void mn_speed_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 1.0;
                              break;
                    case MN_NRML_DEC:
                              value -= 0.5;
                              break;
                    case MN_NRML_INC:
                              value += 0.5;
                              break;
                    case MN_FAST_INC:
                              value += 1.0;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          Move=value;
          Calculations();
}
static void mn_slant_value_cbf(pfMenuItem *item_ptr, int action)
{
          // value item callback
          // get current value
          float value = ((pfMenuValueItem *) item_ptr)->getValue();

          // modify value based on user action
          switch (action)
          {
                    case MN_FAST_DEC:
                              value -= 1.0;
                              break;
                    case MN_NRML_DEC:
                              value -= 0.1;
                              break;
                    case MN_NRML_INC:
                              value += 0.1;
                              break;
                    case MN_FAST_INC:
                              value += 1.0;
                              break;
          }

          // set new value
          ((pfMenuValueItem *) item_ptr)->setValue(value);
          bspd=value;
          bumpchange = 1;
          Calculations();
}
/*************************************************************/
```

```
/********************** DataUpdate Function ******************/
void DataUpdate(void)
{
          shared->spl_switch = menu->buildSplashText(shared->scene,
    "Base Motion of a Full-Car System", "7-Degrees of Freedom",
   "(X, Theta, Phi, x1, x2, x3, x4)","");

}// DataUpdate
/*****************************************************************/

/********************** GroundDraw Function ********************/
void GroundDraw(pfDCS *dcs)
{
          int i,j;
          int N=50;
          int GridStep=5;

   pfVec3 *corners;
   pfVec4 *colors;
          int *lengths;

          pfGeoState *gstate;
          pfGeoSet *groundgset;
          pfGeode *groundgeode;

          gstate = new pfGeoState;
          groundgset = new pfGeoSet;
          groundgeode = new pfGeode;

          colors = (pfVec4 *) pfMalloc(sizeof(pfVec4), pfGetSharedArena());
          corners = (pfVec3 *) pfMalloc(2*(2*21)*sizeof(pfVec3), pfGetSharedArena());

          groundgset->setPrimType(PFGS_LINES);
          groundgset->setNumPrims(42);

          for(i=0;i<=20;i++)
          {
                    lengths[i]=2;
          }


          groundgset->setPrimLengths(lengths);
          groundgset->setAttr(PFGS_COORD3, PFGS_PER_VERTEX, corners, NULL);
          groundgset->setAttr(PFGS_COLOR4, PFGS_OVERALL, colors, NULL);

          colors[0].set(0.3f, 0.4f, 0.0f, 1.0f);

          for(i=0, j=0; i<=41; i=i+2, j=j+GridStep)
          {
                    corners[i].set((float)(j-N),(float)(-N), -5.0f);
                    corners[i+1].set((float)(j-N),(float)(N), -5.0f);
          }

          for(i=42, j=0; i<=83; i=i+2, j=j+GridStep)
          {
                    corners[i].set((float)(-N), (float)(j-N), -5.0f);
                    corners[i+1].set((float)(N), (float)(j-N), -5.0f);
          }

          groundgset->setGState(gstate);

          groundgeode->addGSet(groundgset);
          dcs->addChild(groundgeode);

}//GroundDraw
/*****************************************************************/

pfGeode * createCylinder(float size,pfVec4 *color)
{
 pfGeoSet *gset;
 pfMatrix matrix;
 pfGeode *geode = new pfGeode;
 gset = pfdNewCylinder(256,pfGetSharedArena());
 gset->setAttr(PFGS_COLOR4,PFGS_OVERALL,color,NULL);
 matrix.makeScale(size, size, size);
 pfdXformGSet(gset,matrix);
 geode->addGSet(gset);
 return geode;
}

pfGeode * createSphere(float size,pfVec4 *color)
{
 pfGeoSet *gset;
 pfMatrix matrix;
 pfGeode *geode = new pfGeode;
 gset = pfdNewSphere(256,pfGetSharedArena());
 gset->setAttr(PFGS_COLOR4,PFGS_OVERALL,color,NULL);
 matrix.makeScale(size, size, size);
 pfdXformGSet(gset,matrix);
 geode->addGSet(gset);
 return geode;
}

pfGeode * bump(float Bheight, float Blength)
```

```
{
          pfGeoState *gstate;
          pfGeoSet *gset;
          pfGeode *geode;
          int ind = 0;
          int pind = 0;
          int i,j,c;
          float StPos[]={0,-Blength/2,-5};
          float points[50004][3];
          int polygon[60000][4];
          pfVec4 *bcolor = (pfVec4 *)pfMalloc( 20002 * sizeof(pfVec4),pfGetSharedArena());
          if(!mode)
          {
                    roaddcs->setScale(1);
                    ///////make points//////
                                        for(j=0;j<6;j++)
                                        {

                                                  for (i=0;i<101;i++)
                                                  {
                                                            if(j==2 || j==3)
                                                                      points[ind][0] = StPos[0];
                                                            else
                                                                      points[ind][0] = StPos[0]+(Bheight/2)*(1-
cos(2*MY_PI*i/101));

                                                            if(j < 3 )
                                                                      points[ind][1] = StPos[1]+i*Blength/101+bspd;
                                                            else
                                                                      points[ind][1] = StPos[1]+i*Blength/101-bspd;

                                                            switch(j)
                                                            {
                                                                      case 0:
                                                                                points[ind][2] = StPos[2];
                                                                                break;
                                                                      case 5:
                                                                                points[ind][2] = StPos[2]+10;
                                                                                break;
                                                                      default:
                                                                                points[ind][2] = StPos[2]+5;

                                                            }

                                                            bcolor[ind].set(0.85,0.85,0,1);
                                                            ind++;
                                                  }
                                        }
                    ///////connectivity///////
                    for (i=0; i<500; i++)
                    {
                              polygon[pind][0] = pind;
                              polygon[pind][1] = pind+1;
                              polygon[pind][2] = pind+101;
                              polygon[pind][3] = pind+100;
                              pind++;
                    }
          }
          else
          {
                    roaddcs->setScale(0);
                    Blength = 2000;
                    SGipos = Blength/2+Gipos;
                    StPos[0]=0;
                    StPos[1]=-Blength/2;
                    ///////make points//////
                                        for(j=0;j<2;j++)
                                        {
                                                  for (i=0;i < 10001;i++)
                                                  {
                                                            points[ind][0] = StPos[0]+ampl*sin(freq1*i*Blength/10001);
                                                            if(j < 1 )
                                                                      points[ind][1] = StPos[1]+i*Blength/10001+bspd;
                                                            else
                                                                      points[ind][1] = StPos[1]+i*Blength/10001-bspd;
                                                            points[ind][2] = StPos[2]+j*10;
                                                            bcolor[ind].set(0.4,0.4,0.4,1);
                                                            ind++;
                                                  }
                                        }
                    ///////connectivity///////
                    for (i=0; i<10000; i++)
                    {
                              polygon[pind][0] = pind;
                              polygon[pind][1] = pind+1;
                              polygon[pind][2] = pind+10002;
                              polygon[pind][3] = pind+10001;
                              pind++;
                    }
          }

          ////////define arrays for coordinates and an index array into coordinates array///////
          pfVec3 *coords = (pfVec3 *)pfMalloc( ind * sizeof( pfVec3 ), pfGetSharedArena()   );
          ushort *index  = (ushort *)pfMalloc( 4*pind * sizeof( ushort ), pfGetSharedArena()  );
```

```
        /* store the points in the coordinate array */
    for( i = 0; i < ind; i++ )
    {
        coords[i].set( points[i][0], points[i][1], points[i][2] );
    }

        /* store the connectivity of polygon into index */
    c = 0;
        for( i = 0; i < pind; i++ )
        {
                for( j = 0; j < 4; j++ )
                {
                        index[c] = polygon[i][j];
                        c++;
                }
        }
        /* Create a Geometry Set and set attributes */
        gset = new pfGeoSet();
        gset->setAttr(PFGS_COORD3, PFGS_PER_VERTEX, coords, index);
        gset->setAttr(PFGS_COLOR4,PFGS_OVERALL,bcolor,index);

        /* Set first primitive type as quad  */
        gset->setPrimType( PFGS_QUADS);
        gset->setNumPrims( pind );

        /* Set the rendering mode as wireframe */
        gstate = new pfGeoState;

        /* Create a geometry node and attch the geometry set to it */
        gset->setGState( gstate );

        geode = new pfGeode;
        geode->addGSet( gset );
        return geode;
}
```

D.3: RKF45

```
void calcu(float *y)
{
//calculates gain for rkf45 step
          int j = 0;
          double xin1, xin2, xin3, xin4;

          switch(ControlMode)
          {
                    case 1:
                              u1=0;
                              u2=0;
                              u3=0;
                              u4=0;
                              break;

                    case 2:
                              u1=0;
                              u2=0;
                              u3=0;
                              u4=0;

               for(j = 0; j <14; j++)
                                        u1 = u1 - Kgain[0][j]*y[j+1];

               for(j = 0; j <14; j++)
                                        u2 = u2 - Kgain[1][j]*y[j+1];

               for(j = 0; j <14; j++)
                                        u3 = u3 - Kgain[2][j]*y[j+1];

               for(j = 0; j <14; j++)
                                        u4 = u4 - Kgain[3][j]*y[j+1];
                              break;
                    case 3:
                              xin1 = kt*y01/m1;
                              xin2 = kt*y02/m2;
                              xin3 = kt*y03/m3;
                              xin4 = kt*y04/m4;

                              u1=-(param[1]*pow(y[1]-l1*y[3]-l3*y[5]-y[7],3)+param[2]*(y[2]+l1*y[4]-l3*y[6])
                                        +param[3]*pow(y[7]-xin1,3)+param[4]*y[8]);

                              u3=-(param[1]*pow(y[1]-l1*y[3]+l3*y[5]-y[11],3)+param[2]*(y[2]-l1*y[4]-l3*y[6])
                                        +param[3]*pow(y[11]-xin2,3)+param[4]*y[12]);

                              u2=-(param[5]*pow(y[1]+l2*y[3]+l4*y[5]-y[9],3)+param[6]*(y[2]+l2*y[4]+l4*y[6])
                                        +param[7]*pow(y[9]-xin3,3)+param[8]*y[10]);

                              u4=-(param[5]*pow(y[1]-l2*y[3]+l4*y[5]-y[13],3)+param[6]*(y[2]-l2*y[4]+l4*y[6])
                                        +param[7]*pow(y[13]-xin4,3)+param[8]*y[14]);
                              break;
                    default:
                              u1=0;
                              u2=0;
                              u3=0;
                              u4=0;
                              printf("In default.  How did I get here?\n");

          }

          in1 = (u1+u2+u3+u4);
          in2 = (-l1*(u1+u3)+l2*(u2+u4));
          in3 = (-l3*(u1+u2)+l4*(u3+u4));
          in4 = kt*y01-u1;
          in5 = kt*y02-u2;
          in6 = kt*y03-u3;
          in7 = kt*y04-u4;

}
//User defined system of diff. equations
void f(float t, float *y, float *yp)
{

          defcount++;

          yp[1] = y[2];
   yp[2] = (a[0]*y[1] + a[1]*y[2] + a[2]*y[3] + a[3]*y[4] + a[4]*y[5] + a[5]*y[6] + a[6]*y[7] + a[7]*y[8] + a[8]*y[9] +
a[9]*y[10] + a[10]*y[11] + a[11]*y[12] + a[12]*y[13] + a[13]*y[14] + in1)/MASS;

   yp[3] = y[4];
   yp[4] = (a[14]*y[1] + a[15]*y[2] + a[16]*y[3] + a[17]*y[4] + a[18]*y[5] + a[19]*y[6] + a[20]*y[7] + a[21]*y[8] +
a[22]*y[9] + a[23]*y[10] + a[24]*y[11] + a[25]*y[12] + a[26]*y[13] + a[27]*y[14] + in2)/J1;

          yp[5] = y[6];
          yp[6] = (a[28]*y[1] + a[29]*y[2] + a[30]*y[3] + a[31]*y[4] + a[32]*y[5] + a[33]*y[6] + a[34]*y[7] +
a[35]*y[8] + a[36]*y[9] + a[37]*y[10] + a[38]*y[11] + a[39]*y[12] + a[40]*y[13] + a[41]*y[14] + in3)/J2;

          yp[7] = y[8];
   yp[8] = (a[42]*y[1] + a[43]*y[2] + a[44]*y[3] + a[45]*y[4] + a[46]*y[5] + a[47]*y[6] + a[48]*y[7] + a[49]*y[8] +
in4)/m1;
```

```
          yp[9] = y[10];
   yp[10] = (a[50]*y[1] + a[51]*y[2] + a[52]*y[3] + a[53]*y[4] + a[54]*y[5] + a[55]*y[6] + a[56]*y[9] + a[57]*y[10] +
in5)/m2;

          yp[11] = y[12];
   yp[12] = (a[58]*y[1] + a[59]*y[2] + a[60]*y[3] + a[61]*y[4] + a[62]*y[5] + a[63]*y[6] + a[64]*y[11] + a[65]*y[12] +
in6)/m3;

          yp[13] = y[14];
   yp[14] = (a[66]*y[1] + a[67]*y[2] + a[68]*y[3] + a[69]*y[4] + a[70]*y[5] + a[71]*y[6] + a[72]*y[13] + a[73]*y[14] +
in7)/m4;

}


void fehl(int neqn, float *y, float t, float *h, float *yp,
          float *f1, float *f2, float *f3, float *f4, float *f5, float *s) {


  float ch;
  int i;

  ch = *h / 4.0;

  for (i=1; i<=neqn; i++)
    f5[i] = y[i] + ch * yp[i];

  f(t + ch, f5, f1);

  ch = 3.0 * (*h) / 32.0;

  for (i=1; i<=neqn; i++)
    f5[i] = y[i] + ch * (yp[i] + 3.0 * f1[i]);

  f(t + 3.0 * (*h) / 8.0, f5, f2);

  ch = *h / 2197.0;

  for (i=1; i<=neqn; i++)
    f5[i] = y[i] + ch * (1932.0 * yp[i] + (7296.0 * f2[i] - 7200.0 * f1[i]));

  f(t + 12.0 * (*h) / 13.0, f5, f3);

  ch = *h / 4104.0;

  for (i=1; i<=neqn; i++)
    f5[i] = y[i] + ch * ((8341.0 * yp[i] - 845.0 * f3[i]) + (29440.0 * f2[i]
             - 32832.0 * f1[i]));

  f(t + *h, f5, f4);

  ch = *h / 20520.0;

  for (i=1; i<=neqn; i++)
    f1[i] = y[i] + ch * ((-6080.0 * yp[i] + (9295.0 * f3[i] - 5643.0 * f4[i]))
             + (41040.0 * f1[i] - 28352.0 * f2[i]));

  f(t + *h / 2.0, f1, f5);

// Ready to compute the approximate solution at T+H

  ch = *h / 7618050.0;

  for (i=1; i<=neqn; i++)
    s[i] = y[i] + ch * ((902880.0 * yp[i] + (3855735.0 * f3[i] - 1371249.0 * f4[i]))
            + (3953664.0 * f2[i] + 277020.0 * f5[i]));

} //fehl()



void rkf45 (int neqn, float *y, float t, float tout, float relerr, float abserr,
            int *iflag)
{

            rkfs(neqn, y, t, tout, relerr, abserr, iflag, w1, &w2, w3, w4, w5,
                       w6, w7, &w8, &w9, &iw1, &iw2, &iw3, &iw4, &iw5);

}

//Emulation of Fortran Intrisic Functions
int ISign(int a, int b) {
  if (b<0) return -abs(a);
  else return abs(a);
}

float Max(float a, float b) {
  if (a>=b) return a;
  else return b;
}

float Min(float a, float b) {
  if (a<=b) return a;
  else return b;
}
```

```
float Sign(float a, float b) {
  if (b<0) return (-fabs(a));
  else return fabs(a);
}

void rkfs(int neqn, float *y, float t, float tout, float relerr, float abserr,
          int *iflag, float *yp, float *h, float *f1, float *f2, float *f3,
                      float *f4, float *f5, float *savre, float *savae, int *nfe, int *kop,
                      int *init, int *jflag, int *kflag)  {
//Labels: e25,e40,e45,e50,e60,e65,e80,e100,e200,e260
float remin = 1e-12;
int    maxnfe = 3000;

float a,ae,dt,ee,eeoet,eps,esttol,et,hmin,rer,s,scale;
float tol,toln,ypk;
bool   hfaild,output;
int    i,k,mflag;

// Check the input parameters

  eps = EPSILON;

  if (neqn < 1) {
    *iflag = 8;
    return;
  }

  if (relerr < 0.0) {
    *iflag = 8;
    return;
  }

  if (abserr < 0.0) {
    *iflag = 8;
    return;
  }

  mflag = abs(*iflag);

  if (abs(*iflag) < 1 || abs(*iflag) > 8) {
    *iflag = 8;
    return;
  }

// Is this the first call?

  if (mflag == 1) goto e50;

// Check continuation possibilities

  if (t==tout && *kflag != 3) {
    *iflag = 8;
    return;
  }

  if (mflag != 2) goto e25;

// iflag = +2 or -2

  if (*kflag == 3) goto e45;
  if (*init == 0)  goto e45;
  if (*kflag == 4) goto e40;

  if (*kflag == 5 && abserr == 0.0) {
    printf("\n RKFS: Fatal error.\n");
          return;
  }

  if (*kflag == 6 && relerr <= *savre && abserr <= *savae) {
    printf("\n RKFS: Fatal error.\n");
          return;
  }

  goto e50;

// iflag = 3,4,5,6,7 or 8

e25:if (*iflag == 3) goto e45;
   if (*iflag == 4)  goto e40;
   if (*iflag == 5 && abserr > 0.0) goto e45;

// Integration cannot be continued since user did not respond to
// the instructions pertaining to iflag=5,6,7 or 8.

   printf("\n RKFS: Fatal error.\n");
   return;

// Reset function evaluation counter

e40: *nfe = 0;
  if (mflag == 2) goto e50;

// Reset flag value from previous call
```

```
e45:*iflag = *jflag;

  if (*kflag == 3) mflag = abs(*iflag);

// Save input iflag and set continuation flag for subsequent input checking.

e50: *jflag = *iflag;
   *kflag = 0;

// Save relerr and abserr for checking input on subsequent calls.

  *savre = relerr;
  *savae = abserr;

/* Restrict relative error tolerance to be at least as large as
   2*eps+remin to avoid limiting precision difficulties arising
   from impossible accuracy requests. */

  rer = 2.0 * EPSILON + remin;

// The relative error tolerance is too small.

  if (relerr < rer) {
    relerr = rer;
    *iflag = 3;
    *kflag = 3;
    return;
  }

  dt = tout - t;

  if (mflag == 1) goto e60;
  if (*init == 0) goto e65;
  goto e80;

/* Initialization:
     set initialization completion indicator, init
     set indicator for too many output points, kop
     evaluate initial derivatives
     set counter for function evaluations, nfe
     evaluate initial derivatives
     set counter for function evaluations, nfe
     estimate starting stepsize. */

e60: *init = 0;
  *kop = 0;
  a = t;
  f(a, y, yp);
  *nfe = 1;

  if (t == tout) {
    *iflag = 2;
    return;
  }

e65: *init = 1;
  *h = fabs(dt);
  toln = 0.0;
  for (k = 1; k<=neqn; k++) {
    tol = relerr * fabs(y[k]) + abserr;
    if (tol > 0.0) {
      toln = tol;
      ypk = fabs(yp[k]);
      if (ypk * pow(*h,5.0) > tol)
        *h = pow((tol/ypk),0.2);
    }
  }

  if (toln <= 0.0)  *h = 0.0;

  *h = Max(*h, 26.0 * eps * Max(fabs(t), fabs(dt)));
  *jflag =  ISign(2, *iflag);

// Set stepsize for integration in the direction from T to TOUT

e80: *h = Sign(*h,dt);

// Test to see if RKF45 is being severely impacted by too many output points.

  if (fabs(*h) >= 2.0 * fabs(dt))  *kop = *kop + 1;

// Unnecessary frequency of output.

  if (*kop == 100) {
    *kop = 0;
    *iflag = 7;
    return;
  }

// If too close to output point, extrapolate and return.

  if (fabs(dt) <= 26.0 * eps * fabs(t)) {
    for (i=1; i<=neqn; i++)
      y[i] = y[i] + dt * yp[i];
    a = tout;
```

```
    f(a, y, yp);
    *nfe = *nfe + 1;
    t = tout;
    *iflag = 2;
    return;
  }

// Initialize output point indicator

  output = FALSE;

// To avoid premature underflow in the error tolerance function,
// scale the error tolerances.

  scale = 2.0 / relerr;
  ae = scale * abserr;

// Step by step integration

e100: hfaild = FALSE;

// Set smallest allowable stepsize

  hmin = 26.0 * eps * fabs(t);

/* Adjust stepsize if necessary to hit the output point.
   Look ahead two steps to avoid drastic changes in the stepsize and
   thus lessen the impact of output points on the code. */

  dt = tout - t;
  if (fabs(dt) >= 2.0 * fabs(*h)) goto e200;

// The next successful step will complete the integration to the output point.

  if (fabs(dt) <= fabs(*h)) {
    output = TRUE;
    *h = dt;
    goto e200;
  }

  *h = 0.5 * dt;    // reduce step


// Too many function calls.

e200:if (*nfe > maxnfe) {
    *iflag = 4;
    *kflag = 4;
    return;
}

// Advance an approximate solution over one step of length H

  fehl(neqn, y, t, h, yp, f1, f2, f3, f4, f5, f1);
  *nfe = *nfe + 5;

/* Compute and test allowable tolerances versus local error estimates
   and remove scaling of tolerances.  Note that relative error is
   measured with respect to the average of the magnitudes of the
   solution at the beginning and end of the step. */

  eeoet = 0.0;

  for (k = 1; k<=neqn; k++) {

    et = fabs(y[k]) + fabs(f1[k]) + ae;

    if (et <= 0.0) {
      *iflag = 5;
      return;
    }

    ee = fabs((-2090.0 * yp[k] + (21970.0E+00 * f3[k] - 15048.0 * f4[k]))
          + (22528.0 * f2[k] - 27360.0 * f5[k]));

    eeoet = Max(eeoet, ee/et);

  }

  esttol = fabs(*h) * eeoet * scale / 752400.0;

  if (esttol <= 1.0) goto e260;

// Unsuccessful step.  Reduce the stepsize, try again.
// The decrease is limited to a factor of 1/10.

  hfaild = TRUE;
  output = FALSE;

  if (esttol < 59049.0)
    s = 0.9 / pow(esttol, 0.2);
  else
    s = 0.1;

  *h = s * (*h);
```

```
  if (fabs(*h) < hmin) {
    *iflag = 6;
    *kflag = 6;
    return;
  }
  else
    goto e200;

// Successful step.  Store solution at T+H and evaluate derivatives there.

e260: t = t + *h;
  for (i=1; i<=neqn; i++)  y[i] = f1[i];
  a = t;
  f(a, y, yp);
  *nfe = *nfe + 1;

// Choose next stepsize.  The increase is limited to a factor of 5.
// If step failure has just occurred, next stepsize is not allowed to increase.

  if (esttol > 0.0001889568)
    s = 0.9 / pow(esttol, 0.2);
  else
    s = 5.0;

  if (hfaild)  s = Min(s, 1.0);

  *h = Sign(Max(s * fabs(*h), hmin), *h);

// End of core integrator

// Should we take another step ?

  if (output) {
    t = tout;
    *iflag = 2;
  }

  if (*iflag > 0) goto e100;

// Integration successfully completed.

// ne-step mode
  *iflag = - 2;

}
```

D.4: Kinematics & Gain Optimization

```
void precalc(void)
{
   a[0] = -(k1+k2+k3+k4);
   a[1] = -(b1+b2+b3+b4);
   a[2] = k1*l1-k2*l2+k3*l1-k4*l2;
   a[3] = b1*l1-b2*l2+b3*l1-b4*l2;
   a[4] = k1*l3+k2*l3-k3*l4-k4*l4;
   a[5] = b1*l3+b2*l3-b3*l4-b4*l4;
   a[6] = k1;
   a[7] = b1;
   a[8] = k2;
   a[9] = b2;
   a[10] = k3;
   a[11] = b3;
   a[12] = k4;
   a[13] = b4;

   a[14] = ((k1+k3)*l1-(k2+k4)*l2);
   a[15] = ((b1+b3)*l1-(b2+b4)*l2);
   a[16] = -((k1+k3)*pow(l1,2)+(k2+k4)*pow(l2,2));
   a[17] = -((b1+b3)*pow(l1,2)+(b2+b4)*pow(l2,2));
   a[18] = -(k1*l1*l3-k2*l2*l3-k3*l1*l4+k4*l2*l4);
   a[19] = -(b1*l1*l3-b2*l2*l3-b3*l1*l4+b4*l2*l4);
   a[20] = -k1*l1;
   a[21] = -b1*l1;
   a[22] = k2*l2;
   a[23] = b2*l2;
   a[24] = -k3*l1;
   a[25] = -b3*l1;
   a[26] = k4*l2;
   a[27] = b4*l2;

   a[28] = ((k1+k2)*l3-(k3+k4)*l4);
   a[29] = ((b1+b2)*l3-(b3+b4)*l4);
   a[30] = -(k1*l1*l3-k2*l2*l3-k3*l1*l4+k4*l2*l4);
   a[31] = -(b1*l1*l3-b2*l2*l3-b3*l1*l4+b4*l2*l4);
   a[32] = -((k1+k2)*pow(l3,2)+(k3+k4)*pow(l4,2));
   a[33] = -((b1+b2)*pow(l3,2)+(b3+b4)*pow(l4,2));
   a[34] = -k1*l3;
   a[35] = -b1*l3;
   a[36] = -k2*l3;
   a[37] = -b2*l3;
   a[38] = k3*l4;
   a[39] = b3*l4;
   a[40] = k4*l4;
   a[41] = b4*l4;

   a[42] = k1;
   a[43] = b1;
   a[44] = -k1*l1;
   a[45] = -b1*l1;
   a[46] = -k1*l3;
   a[47] = -b1*l3;
   a[48] = -(kt+k1);
   a[49] = -b1;

   a[50] = k2;
   a[51] = b2;
   a[52] = k2*l2;
   a[53] = b2*l2;
   a[54] = -k2*l3;
   a[55] = -b2*l3;
   a[56] = -(kt+k2);
   a[57] = -b2;

   a[58] = k3;
   a[59] = b3;
   a[60] = -k3*l1;
   a[61] = -b3*l1;
   a[62] = k3*l4;
   a[63] = b3*l4;
   a[64] = -(kt+k3);
   a[65] = -b3;

   a[66] = k4;
   a[67] = b4;
   a[68] = k4*l2;
   a[69] = b4*l2;
   a[70] = k4*l4;
   a[71] = b4*l4;
   a[72] = -(kt+k4);
   a[73] = -b4;


          printf("Filling Matricies: In Precalc");

          zero(A,14,14);

          fillA();
          printf("%f\n", dts);
```

```
          for(int i = 0; i < 14; i++)
                    for(int j = 0; j < 14 ;j++)
                    {
                              Ad[i][j] = A[i][j]*.004;//dts;
                              if(i == j)
                                        Ad[i][j] = Ad[i][j] +1;
                    }

          zero(B,14,4);

          fillB();

          for(i = 0; i < 14; i++)
                    for(int j = 0; j < 4 ;j++)
                    {
                                Bd[i][j] = B[i][j]*.004;//dts;
                    }

          //Set Weighting Matrices
          zero(Q,14,14);
          eye(Q,14);

          Q[1][1] = 500000000.0;
          Q[3][3] = 2000000000.0;
          Q[5][5] = 1000000000.0;

          Q[0][0] = 100000000.0;
          Q[2][2] = 100000000.0;
          Q[4][4] = 100000000.0;


          zero(R,4,4);
          eye(R,4);
          for(i = 0;i < 4; i++)
                    R[i][i] = R[i][i]*1;

          zero(v1,1,14);
          zero(v2,1,14);

          for(i = 0; i<14; i++)
          {
                    v1[0][i] = A[1][i]+A[3][i]*(l1+cg-0.5) + A[5][i]*0.5;
                    v2[0][i] = A[1][i]+A[3][i]*(l1+cg-0.5) - A[5][i]*0.5;
          }

          transpose(v1,TM3,14,1);
          MM(TM3,v1,TM1,14,1,14);

          transpose(v2,TM3,14,1);
          MM(TM3,v2,TM2,14,1,14);

          scalarM(5.0,TM1,TM1,14, 14);// Multiply by weighting
          scalarM(5.0,TM2,TM2,14, 14);

          printMat(TM1, 14, 14);

          add(Q,TM1,Q,14,14);
          add(Q,TM2,Q,14,14);


          //Initialize Ricatti S
          zero(S,14,14);
          eye(S,14);

          //Initiate Empty Gain Matrices
          zero(FAKEgain,4,14);

          if(ParameterChange2)
          {
                    if(ControlMode == 2)
                    {
                       zero(Kgain,4,14);
                              sprintf(waitChar,"Calculating LQR Matrix");
                              waitdcs->setScale(1.4);
                              wait_string->setString(waitChar);
                    }
          }
          if(ParameterChange3)
          {
                    if(ControlMode == 3)
                    {
                       zero(NEAgain,2,4);
                              sprintf(waitChar,"Calculating NEA Gains: %i %% done", NEArep);
                              waitdcs->setScale(1.4);
                              wait_string->setString(waitChar);
                    }
          }
          MakeCalc = 1;//actual calculations are in beam
                                                  //traverser so that calculation notification shows
}
void Calculations(void)
{
          float i;
```

```
          l1=Spread-cg;
          l2=Spread+cg;
          l3=WHLDPTH+0.5;
          l4=WHLDPTH+0.5;

          float div = 15;
          dts = dt/div;

          neqn = 14;    // 7 equations
          abserr = 0.000001;
          relerr = 0.000001;
          iflag = 1;
          if(AnimIndex)
          {
                    if(mode)//Harmonic Input
                    {

                              if(begin)
                              {
                                        y01 = ampl*sin(freq1*(SGipos+Spread-(WHLDPTH/tan(atan(40/bspd)))));
                                        y02 = ampl*sin(freq1*(SGipos-Spread-((WHLDPTH+1)/tan(atan(40/bspd)))));
                                        y03 = ampl*sin(freq1*(SGipos+Spread+(WHLDPTH/tan(atan(40/bspd)))));
                                        y04 = ampl*sin(freq1*(SGipos-Spread+((WHLDPTH+1)/tan(atan(40/bspd)))));
                                        yc[1] = (y01+y02)/2;
                                        yc[2] = 0;
                                        yc[3] = atan((((y02+y04)/2)-((y01+y03)/2))/(l1+l2));
                                        yc[4] = 0;
                                        yc[5] = atan((((y03+y04)/2)-((y01+y02)/2))/(l3+l4));
                                        yc[6] = 0;
                                        yc[7] = y01;
                                        yc[8] = 0;
                                        yc[9] = y02;
                                        yc[10] = 0;
                                        yc[11] = y03;
                                        yc[12] = 0;
                                        yc[13] = y04;
                                        yc[14] = 0;

                                        for(int i = 1; i < (NEQ+1); i++)
                                        {
                                                  w1[i] = 0;
                                                  w3[i] = 0;
                                                  w4[i] = 0;
                                                  w5[i] = 0;
                                                  w6[i] = 0;
                                                  w7[i] = 0;
                                        }
                                        w2 = 0;
                                        w8 = 0;
                                        w9 = 0;
                                        begin = 0;
                                        startime = pfGetTime();
                                        prevtime = pfGetTime();
                              }
                              else
                              {
                                        y01 = ampl*sin(freq1*(position+SGipos+Spread-(WHLDPTH/tan(atan(40/bspd)))));
                                        y02 = ampl*sin(freq1*(position+SGipos-Spread-
((WHLDPTH+1)/tan(atan(40/bspd)))));
                                        y03 = ampl*sin(freq1*(position+SGipos+Spread+(WHLDPTH/tan(atan(40/bspd)))));
                                        y04 = ampl*sin(freq1*(position+SGipos-
Spread+((WHLDPTH+1)/tan(atan(40/bspd)))));

                                        for(i = 0; i<div; i++)
                                        {
                                                  calcu(yc);
                                                  rkf45(neqn, yc, i*dts, i*dts+dts, relerr, abserr, &iflag);
                                                  if((int)i % 4 == 3)
                                                  {
                                                            for(int v = 0; v <=MAXPOINTS-2; v++)
                                                                      gnodes[v] = gnodes[v+1];

                                                            double Driver = yc[1]-(l1+cg-0.5)*yc[3] + .5*yc[5];
                                                            gnodes[MAXPOINTS-2] = Driver*100;//yc[1]*100;//(from feet
to cm)

                                                  }

                                        }
                                        CarOut->gupdate(gnodes,dts);
                                        //car frame position at four corners taken relative to the center
                                        x1= yc[1]-l1*yc[3]-l3*yc[5];
                                        x2= yc[1]+l2*yc[3]-l3*yc[5];
                                        x3= yc[1]-l1*yc[3]+l4*yc[5];
                                        x4= yc[1]+l2*yc[3]+l4*yc[5];
                              }


                    }
                    else //bump input
                    {

                              if(begin)
                              {
```

```
                                        for(int i = 1; i < (NEQ+1); i++)
                                        {
                                                yc[i] = 0;
                                                w1[i] = 0;
                                                w3[i] = 0;
                                                w4[i] = 0;
                                                w5[i] = 0;
                                                w6[i] = 0;
                                                w7[i] = 0;
                                        }
                                        w2 = 0;
                                        w8 = 0;
                                        w9 = 0;

                                        begin = 0;
                                        prevtime = pfGetTime();
                                        bumppos = Blength/2+Gipos;
                                }
                                else
                                {
                                        if(position > (-Gipos - Blength/2 + Spread + bspd) && position < (-Gipos +
Blength/2 + Spread + bspd))
                                        {
                                                y02 = (Bheight/2)*(1-cos((bumppos-Spread-bspd)*2*MY_PI/Blength));
                                        }
                                        else
                                        {
                                                y02 = 0;
                                        }

                                        if(position > (-Gipos - Blength/2 - Spread + bspd) && position < (-Gipos +
Blength/2 - Spread + bspd))
                                        {
                                                y01 = (Bheight/2)*(1-cos((bumppos+Spread-bspd)*2*MY_PI/Blength));
                                        }
                                        else
                                        {
                                                y01 = 0;
                                        }

                                        if(position > (-Gipos - Blength/2 + Spread - bspd) && position < (-Gipos +
Blength/2 + Spread - bspd))
                                        {
                                                y04 = (Bheight/2)*(1-cos((bumppos-Spread+bspd)*2*MY_PI/Blength));
                                        }
                                        else
                                        {
                                                y04 = 0;
                                        }

                                        if(position > (-Gipos - Blength/2 - Spread - bspd) && position < (-Gipos +
Blength/2 - Spread - bspd))
                                        {
                                                y03 = (Bheight/2)*(1-cos((bumppos+Spread+bspd)*2*MY_PI/Blength));
                                        }
                                        else
                                        {
                                                y03 = 0;
                                        }

                                        for(i = 0; i < div; i++)
                                        {
                                                calcu(yc);
                                                rkf45(neqn, yc, i*dts, i*dts+dts, relerr, abserr, &iflag);
                                                if((int)i % 4 == 3)
                                                {
                                                        for(int v = 0; v <=MAXPOINTS-2; v++)
                                                                gnodes[v] = gnodes[v+1];

                                                        double Driver = yc[1]-(l1+cg-0.5)*yc[3] + .5*yc[5];
                                                        gnodes[MAXPOINTS-2] = Driver*100;//yc[1]*100;//(from feet
to cm)

                                                }

                                        }
                                        CarOut->gupdate(gnodes,dts);
                                        bumppos = Move*dt+bumppos;
                                }
                        }
                        position = Move*dt+position;
                        ntime = pfGetTime();
                }
                dt = pfGetTime() - prevtime;
                prevtime = pfGetTime();
}

/********** BumpUpdate Function **********/
static int BumpUpdate(pfTraverser *trav,void *)
{
                if(bumpchange)
                {
                        bumpdcs->removeChild(bumpgeode);
                        bumpgeode = bump(Bheight,Blength);
                        bumpdcs->addChild(bumpgeode);
```

```
                    bumpchange = 0;
          }
          return PFTRAV_CONT;
}// BumpUpdate function
/*************************************************************/


/********************** BeamMotion Function *****************/
static int BeamMotion(pfTraverser *trav,void *)
{
          if(MakeCalc)///must be here to alow the notification to show (waitdcs)
          {
                    if(ControlMode == 1)
                    {
                              if(ParameterChange1)
                              {
                                        if(wait2go)
                                        {
                                                  wait2go = 0;
                                                  AnimIndex = 1;
                                        }
                              }
                              MakeCalc = 0;
                              waitdcs->setScale(0);
                    }
                    if(ControlMode == 2)
                    {
                              if(ParameterChange2)
                              {
                                        //Calculate LQR first

                                        for(int i = 0; i < 100; i++)
                                        {
                                                  MM(S,Bd,TM1,14,14,14);
                                                  transpose(Bd,TM2,4,14);
                                                  MM(TM2,TM1,TM3,4,14,4);
                                                  add(TM3,R,TM1,4,4);
                                                  MatrixInversion(TM1,4,InvMat);
                                                  MM(S,Ad,TM1,14,14,14);
                                                  MM(TM2,TM1,TM3,4,14,14);
                                                  MM(InvMat,TM3,Kgain,4,4,14);

                                                  MM(Bd,Kgain,TM1,14,4,14);
                                                  sub(Ad,TM1,TM2,14,14);
                                                  MM(S,TM2,TM3,14,14,14);
                                                  transpose(Ad,TM1,14,14);
                                                  MM(TM1,TM3, TM2,14,14,14);
                                                  add(TM2,Q,S,14, 14);
                                        }
                                        ParameterChange2 = 0;
                                        if(wait2go)
                                        {
                                                  wait2go = 0;
                                                  AnimIndex = 1;
                                        }
                                        waitdcs->setScale(0);
                              }
                              MakeCalc = 0;
                    }

                    if(ControlMode == 3)
                    {
                              if(ParameterChange3)
                              {
                                        if(NEArep == 0)
                                        {
                                                  for(int i = 0; i<9; i++)
                                                  {
                                                            param[i] = 0;
                                                            psign[i] = -1;
                                                            pinterval[i] = 50;
                                                            oldpinterval[i] = 50;
                                                  }

                                                  mval = 0;
                                                  newmval = 0;
                                                  oldmval = 0;
                                                  limit = 10.0;
                                                  intreturn = 0;

                                                  printf("exit program = %i\n", exit_program);

                                                  yr01[0] = 0.0;
                                                  yr02[0] = 0.0;
                                                  yr03[0] = 0.0;
                                                  yr04[0] = 0.0;
                                                  srand ( time(NULL) );
                                                  dtn = dts;

                                                  myfile = fopen("grandnum.txt", "r");
                                                  myoutfile = fopen("wutiread.txt","w");
                                                  if(!myfile)
                                                  {
                                                            printf("Can't open random number file!");
```

```
                    exit_program = 1;
                }
                if(!myoutfile)
                {
                    printf("Can't open What I Read file!");
                    exit_program = 1;
                }

                float nextnum;
                for(long ij = 1; ij < neareps; ij++)
                {
                    fscanf(myfile,"%f",&nextnum);
                    fprintf(myoutfile,"%f",nextnum);
                    yr01[ij] = yr01[ij-1]+nextnum;
                    fgetc(myfile);
                    fputc('\t',myoutfile);

                    fprintf(myoutfile,"%f",nextnum);
                    fscanf(myfile,"%f",&nextnum);
                    yr02[ij] = yr02[ij-1]+nextnum;
                    fgetc(myfile);
                    fputc('\t',myoutfile);

                    fprintf(myoutfile,"%f",nextnum);
                    fscanf(myfile,"%f",&nextnum);
                    yr03[ij] = yr03[ij-1]+nextnum;
                    fgetc(myfile);
                    fputc('\t',myoutfile);

                    fprintf(myoutfile,"%f",nextnum);
                    fscanf(myfile,"%f",&nextnum);
                    yr04[ij] = yr04[ij-1]+nextnum;
                    fgetc(myfile);
                    fputc('\n',myoutfile);
                    if(nextnum != nextnum)
                    {
                        printf("Read wrong number!");
                        exit_program = 1;
                    }

                }
                fclose(myfile);
                fclose(myoutfile);
                newmval = calculateE(param, yr01, yr02, yr03, yr04);
                mval = newmval;
                printf("%f\n",mval);
                printf("**************************************************\n");
                if(mval == eub)
                    mval = (eub-1);
            }

            int istart;
            int ifinish;
            if(NEArep <9)
            {
                istart = NEArep;
                ifinish = NEArep+1;
            }
            else
            {
                if(NEArep%2 == 1)
                {
                    istart = 1;
                    ifinish = 5;
                }
                else
                {
                    istart = 5;
                    ifinish = 9;
                }
            }
            for(int i = istart; i < ifinish; i++)
            {
                param[i] = param[i] + psign[i]*pinterval[i];
                while(fabs(limit) > .001)
                {

                    newmval = calculateE(param, yr01, yr02, yr03, yr04);
                    limit = newmval-mval;

                    if(newmval < mval)
                    {
                        if((1-(newmval/mval)) < 0.2  && (direcs > 4))
                        {
                            float Atemp, Btemp, Ctemp, Dtemp;
                            Atemp = mval-newmval;
                            Btemp = oldmval-mval;
                            Ctemp = Btemp-Atemp;
                            Dtemp = Atemp/Ctemp;
                            if(Dtemp > 0)
                            {
                                oldpinterval[i] =
pinterval[i];

                                pinterval[i] =
pinterval[i]*4;//Dtemp;
```

```
                                                                                intreturn = 1;
                                                                                direcs = 0;
                                                                        }
                                                                        else
                                                                        pinterval[i] = pinterval[i]*4;
                                                                }
                                                                oldmval = mval;
                                                                mval = newmval;
                                                                direcs++;
                                                                if(direcs > 6)
                                                                        direcs = 6;
                                                        }
                                                        else
                                                        {


                                                                if(direcs == 5)
                                                                        pinterval[i] = pinterval[i]*0.2;
                                                                if(newmval == eub)
                                                                {
                                                                        param[i] = param[i] -
psign[i]*pinterval[i];
                                                                        psign[i] = psign[i]*-1;
                                                                        pinterval[i] = pinterval[i]*0.5;
                                                                        limit = 10;
                                                                        direcs = 0;
                                                                }
                                                                else
                                                                {
                                                                        oldmval = mval;
                                                                        mval = newmval;
                                                                        psign[i] = psign[i]*-1;
                                                                        pinterval[i] = pinterval[i]*0.5;
                                                                        direcs = 0;
                                                                }
                                                        }
                                                        if(CAVEgetbutton(CAVE_NKEY))
                                                        {
                                                                limit = 0;
                                                                NEArep = 100;
                                                        }
                                                        param[i] = param[i] + psign[i]*pinterval[i];

                                                        if(intreturn)
                                                        {
                                                                pinterval[i] = oldpinterval[i];
                                                                intreturn = 0;
                                                        }
                                                }
                                                limit = 10.0;
                                        }
                                        NEArep++;

                                        sprintf(waitChar,"Calculating NEA Gains: %i %% done\n mval=%f", NEArep*2,
mval);
                                        wait_string->setString(waitChar);
                                                printf("%f\t",mval);
                                                printf("Parameters:");
                                                        for(int ijk = 1;ijk<9; ijk++)
                                                printf("%f\t",param[ijk]);
                                                printf("\n");

                                        if(tempint <0)
                                        {
                                                printf("%f\t",mval);
                                                printf("Parameters:");
                                                        for(int ijk = 1;ijk<9; ijk++)
                                                printf("%f\t",param[ijk]);
                                                printf("\n");
                                                tempint++;
                                        }
                                        if(NEArep == 50)
                                        {
                                                printf("Parameter Change = %i\t Rep = %i\n",ParameterChange3,
NEArep);
                                                ParameterChange3 = 0;
                                                NEArep = 0;
                                                MakeCalc = 0;
                                                waitdcs->setScale(0);
                                                if(wait2go)
                                                {
                                                        wait2go = 0;
                                                        AnimIndex=1;
                                                }
                                        }
                                }//end if(ParameterChange3)

                        }//end if(controlmode3)

                }//end if makecalc


                pfDCS *dcss = (pfDCS *) trav->getNode();
                Calculations();
                if(ButtonTimer < 20)
```

```
                    ButtonTimer++;

          if(CAVEgetbutton(CAVE_GKEY) && (ButtonTimer >= 10))
          {
                    if(AnimIndex)
                    {
                              AnimIndex=0;
                    }
                    else
                    {
                              if(((ControlMode == 1) && ParameterChange1) || ((ControlMode == 2) && ParameterChange2)
|| ((ControlMode == 3) && ParameterChange3))
                                        wait2go = 1;
                              else
                                        AnimIndex=1;

                              precalc();
                              Calculations();
                    }


                    ButtonTimer = 0;
          }


          if((CAVEgetbutton(CAVE_RKEY) || CAVEBUTTON2) && (ButtonTimer >= 10))
          {

                    if(((ControlMode == 1) && ParameterChange1) || ((ControlMode == 2) && ParameterChange2) ||
((ControlMode == 3) && ParameterChange3))
                    {
                              if(AnimIndex)
                              {
                                        AnimIndex = 0;
                                        wait2go = 1;
                              }
                    }
                    precalc();
                    for(int i = 0; i< 50; i++)
                    {
                              yc[1] = 0;
                              yc[2] = 0;
                              yc[3] = 0;
                              yc[4] = 0;
                              yc[5] = 0;
                              yc[6] = 0;
                              yc[7] = 0;
                              yc[8] = 0;
                              yc[9] = 0;
                              yc[10] = 0;
                              yc[11] = 0;
                              yc[12] = 0;
                              yc[13] = 0;
                              yc[14] = 0;


                              for(int i = 1; i < (NEQ+1); i++)
                              {
                                        w1[i] = 0;
                                        w3[i] = 0;
                                        w4[i] = 0;
                                        w5[i] = 0;
                                        w6[i] = 0;
                                        w7[i] = 0;
                              }
                              w2 = 0;
                              w8 = 0;
                              w9 = 0;

                              y01 = 0;
                              y02 = 0;
                              y03 = 0;
                              y04 = 0;

                              u[1] = 0;
                              u[2] = 0;
                              u[3] = 0;
                              u[4] = 0;

                              rkf45(neqn, yc, i*dts, i*dts+dts, relerr, abserr, &iflag);
                    }
                    prevtime = pfGetTime();
                    startime = pfGetTime();
                    position = 0;
                    bumppos = Blength/2+Gipos;
                    begin = 1;
                    precalc();
                    ButtonTimer = 0;

                    CarOut2->energy = CarOut->energy;
                    CarOut2->gupdate(gnodes,dts);
                    CarOut->reset(gnodes);

          }
```

```
if(CAVEBUTTON1 && (ButtonTimer >= 10))
{
          if(NodeChoiceMode == 0)
                    NodeChoiceMode = 1;
          else
          {
                    NodeChoiceMode = 0;
                    Node1dcs->setScale(0);
                    Node2dcs->setScale(0);
                    Node3dcs->setScale(0);
                    Node4dcs->setScale(0);
                    Node5dcs->setScale(0);
                    Node6dcs->setScale(0);
                    Node7dcs->setScale(0);
          }

          ButtonTimer = 0;
}


if(CAVEgetbutton(CAVE_SKEY) && (ButtonTimer >= 10))
{

          if(!SceneMode)
          {
                    headT_dcs->setTrans(10,80,0);
                    headT_dcs->setRot(30,0,0);
                    SceneMode = (SceneMode+1)%2;
          }
          else
          {
                    headT_dcs->setTrans(0,0,0);
                    headT_dcs->setRot(0,0,0);
                    SceneMode = (SceneMode+1)%2;
          }
          ButtonTimer = 0;

}
if(CAVEgetbutton(CAVE_BKEY) && (ButtonTimer >= 10))
{
          switch(carbody)
          {
                    case 1:
                              carbody = 2;
                              Chassisdcs->setScale(HCarisc);
                              HCardcs->setScale(HCarisc);
                              Walldcs->setScale(0);
                              CGdcs->setScale(0);
                                        break;

                    case 2:
                              carbody = 3;
                              Chassisdcs->setScale(HCarisc);
                              HCardcs->setScale(0);
                              Walldcs->setScale(0);
                              CGdcs->setScale(0);
                              break;

                    case 3:
                              carbody = 1;
                              Chassisdcs->setScale(0);
                              HCardcs->setScale(0);
                              Walldcs->setScale(Walliscz, Walliscx, Walliscy);
                              CGdcs->setScale(1);
                              break;
          }
          ButtonTimer = 0;

}

if(CAVEgetbutton(CAVE_MKEY) && (ButtonTimer >= 10))
{
          if(mode)
          {
                    mode = 0;
                    bumpchange = 1;
          }
          else
          {
                    mode = 1;
                    bumpchange = 1;
          }

          ButtonTimer = 0;

}

if(CAVEgetbutton(CAVE_CKEY) && (ButtonTimer >= 20))
{

          if(CAVEgetbutton(CAVE_VKEY))
                    ControlMode = ((ControlMode+1)%3)+1;
```

```
                    else
                              ControlMode = ((ControlMode)%3)+1;



                    if(ControlMode==2)
                    {

                              sprintf(freq1Char,"Control = LQR");
                       freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.7, 0.3, 0.3, 1.0);
                    }

                    if(ControlMode==3)
                    {

                              sprintf(freq1Char,"Control = NEA");
                              freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.3, 0.3, 0.7, 1.0);
                    }


                    if(ControlMode==1)
                    {

                              sprintf(freq1Char,"Control = OFF");
                    freq1_string->setString(freq1Char);
                       freq1_string->setColor(0.7, 0.5, 0.5, 1.0);
                    }
                    ButtonTimer = 0;
          }



          if(AnimIndex)
          {

                    dcss->setTrans(yc[1]+Wallipos-yc[3]*Spread+yc[5]*(0.2-WHLDPTH),Spread+Gipos+position+0.1,-
WHLDPTH*1.05);
                    dcss->setRot(yc[3]*180/(MY_PI),0,yc[5]*180/(MY_PI));

                    CGdcs->setTrans(Wallipos+0.3/3.2808+yc[1],Gipos+position,0);
                    CGdcs->setRot(yc[3]*180/(MY_PI),0,yc[5]*180/(MY_PI));


                    Wheel1dcs->setTrans(WHLipos+y01,Spread+Gipos+position,-WHLDPTH);
                    Wheel1dcs->setRot(0,90,position*180/(MY_PI*.4389/*radius of tire*/));

                    Wheel2dcs->setTrans(WHLipos+y02,-Spread+Gipos+position,-WHLDPTH-1*.3/3.28);
                    Wheel2dcs->setRot(0,90,position*100);//*180/(MY_PI*.4389/*radius of tire*/));

                    Wheel3dcs->setTrans(WHLipos+y03,Spread+Gipos+position,WHLDPTH);
                    Wheel3dcs->setRot(0,-90,-position*100);//*180/(MY_PI*.4389/*radius of tire*/));

                    Wheel4dcs->setTrans(WHLipos+y04,-Spread+Gipos+position,WHLDPTH+1*.3/3.28);
                    Wheel4dcs->setRot(0,-90,-position*100);//*180/(MY_PI*.4389/*radius of tire*/));

                    Spring1dcs->setTrans(Springipos+y01,Spread+Gipos+position,1.2*.3/3.28-WHLDPTH);
                    Spring1dcs->setScale(Springisc,Springisc+springdsf*(y01-x1),Springisc);

                    Spring2dcs->setTrans(Springipos+y02,-Spread+Gipos+position,0.2*.3/3.28-WHLDPTH);
                    Spring2dcs->setScale(Springisc,Springisc+springdsf*(y02-x2),Springisc);

                    Spring3dcs->setTrans(Springipos+y03,Spread+Gipos+position,WHLDPTH-1.2*.3/3.28);
                    Spring3dcs->setScale(Springisc,Springisc+springdsf*(y03-x3),Springisc);

                    Spring4dcs->setTrans(Springipos+y04,-Spread+Gipos+position,WHLDPTH-0.2*.3/3.28);
                    Spring4dcs->setScale(Springisc,Springisc+springdsf*(y04-x4),Springisc);

                    HCardcs->setTrans(HCaripos+yc[1],Gipos+position,0);
                    HCardcs->setRot(90+yc[3]*180/(MY_PI),yc[5]*180/(MY_PI),90);

                    Chassisdcs->setTrans(HCaripos+yc[1],Gipos+position,0);
                    Chassisdcs->setRot(90+yc[3]*180/(MY_PI),yc[5]*180/(MY_PI),90);


          }

          return PFTRAV_CONT;
}// Beam Motion
/*************************************************************/

static int NodeUpdate(pfTraverser *trav,void *)
{
          if(NodeChoiceMode)
          {
                    Node1dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5);
                    Node2dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5+1);
                    Node3dcs->setTrans((Gipos+position-0.5)*0.4,30,0.3*(Wallipos+0.5+yc[1])-5+2);

                    Node4dcs->setTrans((Spread+Gipos+position)*0.4,30+0.3*(-WHLDPTH),0.3*(WHLipos+y01)-5);
                    Node5dcs->setTrans((-Spread+Gipos+position)*0.4,30+0.3*(-WHLDPTH-1),0.3*(WHLipos+y02)-5);
                    Node6dcs->setTrans((Spread+Gipos+position)*0.4,30+0.3*(WHLDPTH),0.3*(WHLipos+y03)-5);
                    Node7dcs->setTrans((-Spread+Gipos+position)*0.4,30+0.3*(WHLDPTH+1),0.3*(WHLipos+y04)-5);
```

```
CAVEGetPosition(CAVE_WAND_NAV, gpos1.vec);
CAVEGetVector(CAVE_WAND_FRONT_NAV, gdir1.vec);

int test = isectTest(Node1dcs,gpos1,gdir1,gpos2,gdir2);

if(test)
          {
                    Node1dcs->setScale(2);
                    if(CAVEBUTTON3)
                    {
                              NodeChoice = 1;
sprintf(amplChar,"Graphing X (cm)");
                    CarOut->Ylabel(amplChar);
                    CarOut2->Ylabel(amplChar);
                              CarOut->scale = 0.3;
                              CarOut2->scale = 0.3;
                    }
          }
else
          Node1dcs->setScale(1);


test = isectTest(Node2dcs,gpos1,gdir1,gpos2,gdir2);

if(test)
          {
                    Node2dcs->setScale(2);
                    if(CAVEBUTTON3)
                    {
                              NodeChoice = 2;
                    sprintf(amplChar,"Graphing Theta (rad)");
                    CarOut->Ylabel(amplChar);
                    CarOut2->Ylabel(amplChar);
                              CarOut->scale = 5;
                              CarOut2->scale = 5;
                    }
          }
else
          Node2dcs->setScale(1);

test = isectTest(Node3dcs,gpos1,gdir1,gpos2,gdir2);

if(test)
          {
                    Node3dcs->setScale(2);
                    if(CAVEBUTTON3)
                    {
                              NodeChoice = 3;
                    sprintf(amplChar,"Graphing Phi (radians)");
                    CarOut->Ylabel(amplChar);
                    CarOut2->Ylabel(amplChar);
                              CarOut->scale = 5;
                              CarOut2->scale = 5;
                    }

          }
else
          Node3dcs->setScale(1);

test = isectTest(Node4dcs,gpos1,gdir1,gpos2,gdir2);

if(test)
          {
                    Node4dcs->setScale(2);
                    if(CAVEBUTTON3)
                    {
                              NodeChoice = 4;
                    sprintf(amplChar,"Graphing x1 (cm)");
                    CarOut->Ylabel(amplChar);
                    CarOut2->Ylabel(amplChar);
                              CarOut->scale = 0.2;
                              CarOut2->scale = 0.2;
                    }


          }
else
          Node4dcs->setScale(1);
test = isectTest(Node5dcs,gpos1,gdir1,gpos2,gdir2);

if(test)
          {
                    Node5dcs->setScale(2);
                    if(CAVEBUTTON3)
                    {
                              NodeChoice = 5;
                    sprintf(amplChar,"Graphing x2 (cm)");
                    CarOut->Ylabel(amplChar);
                    CarOut2->Ylabel(amplChar);
                              CarOut->scale = 0.2;
                              CarOut2->scale = 0.2;
                    }

          }
else
```

```
                    Node5dcs->setScale(1);

                test = isectTest(Node6dcs,gpos1,gdir1,gpos2,gdir2);

                if(test)
                    {
                        Node6dcs->setScale(2);
                        if(CAVEBUTTON3)
                        {
                            NodeChoice = 6;
                        sprintf(amplChar,"Graphing x3 (cm)");
                        CarOut->Ylabel(amplChar);
                        CarOut2->Ylabel(amplChar);
                            CarOut->scale = 0.2;
                            CarOut2->scale = 0.2;
                        }

                    }
                else
                    Node6dcs->setScale(1);

                test = isectTest(Node7dcs,gpos1,gdir1,gpos2,gdir2);

                if(test)
                    {
                        Node7dcs->setScale(2);
                        if(CAVEBUTTON3)
                        {
                            NodeChoice = 7;
                        sprintf(amplChar,"Graphing x4 (cm)");
                        CarOut->Ylabel(amplChar);
                        CarOut2->Ylabel(amplChar);
                            CarOut->scale = 0.2;
                            CarOut2->scale = 0.2;
                        Node1dcs->setScale(0);
                        Node2dcs->setScale(0);
                        Node3dcs->setScale(0);
                        Node4dcs->setScale(0);
                        Node5dcs->setScale(0);
                        Node6dcs->setScale(0);
                        Node7dcs->setScale(0);
                        NodeChoiceMode = 0;
                precalc();
                        }

                    }
                else
                    Node7dcs->setScale(1);

                if(CAVEBUTTON3)
                {


                }
        }

        return PFTRAV_CONT;
}

int isectTest(pfNode *node,pfVec3 pos,pfVec3 dir,pfVec3 *retPoint,pfVec3 *retNorm)
{
 pfSegSet segset;
 pfHit **hits[32];
 segset.activeMask = 1;
 segset.isectMask = 0xffffffff;
 segset.discFunc = NULL;
 segset.bound = NULL;
 segset.mode = PFTRAV_IS_GSET | PFTRAV_IS_NORM | PFTRAV_IS_CULL_BACK;
 segset.segs[0].pos = pos;
 segset.segs[0].dir = dir;
 segset.segs[0].length = 100.0f;


 if (node->isect(&segset, hits))
        {
        pfVec3 pnt, norm;
        pfMatrix xmat;
        (*hits[0])->query(PFQHIT_POINT, pnt.vec);
        (*hits[0])->query(PFQHIT_XFORM, (float*)xmat.mat);

        pnt.xformPt(pnt, xmat);
        *retPoint = pnt;

        (*hits[0])->query(PFQHIT_NORM, norm.vec);
        norm.xformVec(norm, xmat);
        *retNorm = norm;
        return 1;
        }

 return 0;
}

/*******************  ScreenText  ******************/
void ScreenText(void)
{
```

```
font = pfdLoadFont_type1("Helvetica-Bold", PFDFONT_FILLED);
pfMatrix freq1_matrix;
pfMatrix ampl_matrix;
pfMatrix WHub_matrix;
pfMatrix stiff1_matrix;
pfMatrix damp_matrix;
pfMatrix mode_matrix;
pfMatrix psh_matrix;
pfMatrix wait_matrix;

freq1_matrix.makeTrans(-4.0, 4.0, 6.4);
freq1_matrix.preScale(0.2, 0.2, 0.2, freq1_matrix);
freq1_scs = new pfSCS(freq1_matrix);

ampl_matrix.makeTrans(-4.0, 4.0, 6.2);
ampl_matrix.preScale(0.2, 0.2, 0.2, ampl_matrix);
ampl_scs = new pfSCS(ampl_matrix);

WHub_matrix.makeTrans(-4.0, 4.0, 7.0);
WHub_matrix.preScale(0.2, 0.2, 0.2, WHub_matrix);
WHub_scs = new pfSCS(WHub_matrix);

stiff1_matrix.makeTrans(-4.0, 4.0, 6.8);
stiff1_matrix.preScale(0.2, 0.2, 0.2, stiff1_matrix);
stiff1_scs = new pfSCS(stiff1_matrix);

damp_matrix.makeTrans(-4.0, 4.0, 6.6);
damp_matrix.preScale(0.2, 0.2, 0.2, damp_matrix);
damp_scs = new pfSCS(damp_matrix);

psh_matrix.makeTrans(-4.0, 4.0, 6.2);
psh_matrix.preScale(0.2, 0.2, 0.2, psh_matrix);
psh_scs = new pfSCS(psh_matrix);
/***********************************/

freq1_string = new pfString;
ampl_string = new pfString;
WHub_string = new pfString;
stiff1_string = new pfString;
damp_string = new pfString;
psh_string = new pfString;
/***********************************/

freq1_string->setFont(font);
freq1_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
freq1_string->setColor(0.7, 0.5, 0.5, 1.0);

ampl_string->setFont(font);
ampl_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
ampl_string->setColor(0.7, 0.5, 0.5, 1.0);

WHub_string->setFont(font);
WHub_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
WHub_string->setColor(0.7, 0.5, 0.5, 1.0);

stiff1_string->setFont(font);
stiff1_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
stiff1_string->setColor(0.7, 0.5, 0.5, 1.0);

damp_string->setFont(font);
damp_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
damp_string->setColor(0.7, 0.5, 0.5, 1.0);

psh_string->setFont(font);
psh_string->setMode(PFSTR_JUSTIFY, PFSTR_LEFT);
psh_string->setColor(0.7, 0.5, 0.5, 1.0);


/***************************************/
sprintf(freq1Char,"Control = OFF");
freq1_string->setString(freq1Char);

sprintf(amplChar,"Graphing X (cm)");
ampl_string->setString(amplChar);

sprintf(WHubChar,"Mass=%1.1f kg",MASS);
WHub_string->setString(WHubChar);

sprintf(stiff1Char,"Siffness=%1.1f N/m",k1);
stiff1_string->setString(stiff1Char);

sprintf(dampChar,"Damping=%1.1f Ns/m",b1);
damp_string->setString(dampChar);

sprintf(pshChar,"Phase Shift=%1.2f rad", psh1);
psh_string->setString(pshChar);

/***************************************/
freq1_text = new pfText;
freq1_text->addString(freq1_string);
freq1_scs->addChild(freq1_text);

ampl_text = new pfText;
ampl_text->addString(ampl_string);
```

```
          WHub_text = new pfText;
          WHub_text->addString(WHub_string);
          WHub_scs->addChild(WHub_text);

          stiff1_text = new pfText;
          stiff1_text->addString(stiff1_string);
          stiff1_scs->addChild(stiff1_text);

          damp_text = new pfText;
          damp_text->addString(damp_string);
          damp_scs->addChild(damp_text);

          psh_text = new pfText;

}//ScreenText


/**********************CAR CONTROL************************/
// matrix inversioon
// the result is put in Y
void MatrixInversion(float **A, int order, float **Y){

    // get the determinant of a
    float det = 1.0/CalcDeterminant(A,order);

    // memory allocation
    float *temp = new float[(order-1)*(order-1)];
    float **minor = new float*[order-1];
    for(int i=0;i<order-1;i++)
        minor[i] = temp+(i*(order-1));

    for(int j=0;j<order;j++)
    {
        for(int i=0;i<order;i++)
        {
            // get the co-factor (matrix) of A(j,i)
            GetMinor(A,minor,j,i,order);
            Y[i][j] = det*CalcDeterminant(minor,order-1);
            if( (i+j)%2 == 1)
                Y[i][j] = -Y[i][j];
        }
    }
    // release memory
    delete [] minor[0];
//     for(int i=0;i<order-1;i++)
//         delete [] minor[i];
    delete [] minor;
}
// calculate the cofactor of element (row,col)
int GetMinor( float **src, float **dest, int row, int col, int order){
    // indicate which col and row is being copied to dest
          int colCount=0,rowCount=0;

    for(int i = 0; i < order; i++ )
    {
        if( i != row )
        {
            colCount = 0;
            for(int j = 0; j < order; j++ )
            {
                // when j is not the element
                if( j != col )
                {
                    dest[rowCount][colCount] = src[i][j];
                    colCount++;
                }
            }
            rowCount++;
        }
    }

    return 1;
}
// Calculate the determinant recursively.
float CalcDeterminant( float **mat, int order){

    // order must be >= 0
          // stop the recursion when matrix is a single element
          if( order == 1 )
          return mat[0][0];

    // the determinant value
          float det = 0;

     // allocate the cofactor matrix
          float **minor;
          minor = new float*[order-1];
          for(int i = 0; i < order-1;i++)
                    minor[i] = new float[order-1];

          for(i = 0; i < order; i++ )
          {
                    // get minor of element (0,i)
                    GetMinor(mat, minor, 0, i , order);
                    // the recusion is here!
```

```
                    det += pow( -1.0, i ) * mat[0][i] * CalcDeterminant( minor,order-1 );
          }

    // release memory
    for(i=0;i<order-1;i++)
        delete [] minor[i];
    delete [] minor;

    return det;
}
void eye(float ** theEye, int size){

          for(int i = 0; i<size; i++)
                    theEye[i][i] = 1;

}
void zero(float ** theMat, int rsize, int csize){

          for(int i = 0; i<rsize; i++){
                    for(int j = 0; j<csize; j++){
                              theMat[i][j] = 0.0;
                    }
          }
}

void printMat( float ** theMat, int rsize, int csize)
{
          for(int i = 0; i<rsize; i++){
                    for(int j = 0; j<csize; j++){
                              printf("%f\t",theMat[i][j]);
                    }
                    printf("\n\n");
          }
          printf("\n\n\n");

}

void initMatrices(void)
{
          Q = new float*[14];
          for(int i=0;i<14;i++)
                    Q[i] = new float[14];

          R = new float*[4];
          for(i=0;i<4;i++)
                    R[i] = new float[4];

          P = new float*[14];
          for(i=0;i<14;i++)
                    P[i] = new float[14];

          A = new float*[14];
          for(i=0;i<14; i++)
                    A[i] = new float[14];

          Ad = new float*[14];
          for(i=0;i<14; i++)
                    Ad[i] = new float[14];

          B = new float*[14];
          for(i=0;i<14; i++)
                    B[i] = new float[4];

          Bd = new float*[14];
          for(i=0;i<14; i++)
                    Bd[i] = new float[4];

          S = new float*[14];
          for(i=0;i<14; i++)
                    S[i] = new float[14];

          TM1 = new float*[14];
          for(i=0;i<14; i++)
                    TM1[i] = new float[14];

          TM2 = new float*[14];
          for(i=0;i<14; i++)
                    TM2[i] = new float[14];

          TM3 = new float*[14];
          for(i=0;i<14; i++)
                    TM3[i] = new float[14];

          InvMat = new float*[4];
          for(i=0;i<4; i++)
                    InvMat[i] = new float[4];

          Kgain = new float*[4];
          for(i=0;i<4; i++)
                    Kgain[i] = new float[14];

          NEAgain = new float*[4];
          for(i=0;i<4; i++)
                    NEAgain[i] = new float[14];
```

```
          FAKEgain = new float*[4];
          for(i=0;i<4; i++)
                    FAKEgain[i] = new float[14];

          v1 = new float*[1];
          for(i=0;i<1; i++)
                    v1[i] = new float[14];

          v2 = new float*[1];
          for(i=0;i<1; i++)
                    v2[i] = new float[14];


}
void delMatrices(void)
{
          for(int i=0;i<14;i++)
                    delete [] Q[i];
          delete [] Q;

          for(i=0;i<4;i++)
                    delete [] R[i];
          delete [] R;

          for(i=0;i<14;i++)
                    delete [] P[i];
          delete [] P;

          for(i=0;i<14; i++)
                    delete [] A[i];
          delete [] A;

          for(i=0;i<14; i++)
                    delete [] Ad[i];
          delete [] Ad;

          for(i=0;i<14; i++)
                    delete [] B[i];
          delete [] B;

          for(i=0;i<14; i++)
                    delete [] Bd[i];
          delete [] Bd;

          for(i=0;i<14; i++)
                    delete [] S[i];
          delete [] S;

          for(i=0;i<14; i++)
                    delete [] TM1[i];
          delete [] TM1;

          for(i=0;i<14; i++)
                    delete [] TM2[i];
          delete [] TM2;

          for(i=0;i<14; i++)
                    delete [] TM3[i];
          delete [] TM3;

          for(i=0;i<4; i++)
                    delete [] InvMat[i];
          delete [] InvMat[;

          for(i=0;i<4; i++)
                    delete [] Kgain[i];
          delete [] Kgain;

          for(i=0;i<4; i++)
                    delete [] NEAgain[i];
          delete [] NEAgain;

          for(i=0;i<4; i++)
                    delete [] FAKEgain[i];
          delete [] FAKEgain;

          for(i=0;i<1; i++)
                    delete [] v1[i];
          delete [] v1;

          for(i=0;i<1; i++)
                    delete [] v2[i];
          delete [] v2;
}
void fillA(void)
{

    int aindex= 0 ;
    int lcounter = 6;
    float divisor;


          for(int i= 1; i < 14; i= i+2)
          {
                    switch(i)
```

```
          {
          case 1:
          divisor = MASS;
            break;
                case 3:
                divisor = J1;
          break;
                              case 5:
            divisor = J2;
            break;
          case 7:
            divisor = m1;
            break;
          case 9:
            divisor = m2;
            break;
          case 11:
            divisor = m3;
            break;
          case 13:
            divisor = m4;
            break;
          default:
            divisor = 1;
                    }

        if(i <6)
            for(int j = 0; j < 14; j++)
            {
                A[i][j] = a[aindex]/divisor;
                aindex++;

            }
        else
        {
            for(int j = 0; j < 6; j++)
            {
                A[i][j] = a[aindex]/divisor;
                aindex++;

            }
            A[i][lcounter] = a[aindex]/divisor;
            A[i][lcounter+1] = a[aindex+1]/divisor;
            aindex = aindex+2;
            lcounter = lcounter+2;

        }
    }



    for(i = 0; i <14; i = i+2)
        A[i][i+1] = 1;

}


void fillB(void){


          for(int i= 0; i < 4; i++)
                    B[1][i] = 1/MASS;

          B[3][0] = -l1/J1;
          B[3][1] = l2/J1;
          B[3][2] = -l1/J1;
          B[3][3] = l2/J1;
          B[5][0] = -l3/J2;
          B[5][1] = -l3/J2;
          B[5][2] = l4/J2;
          B[5][3] = l4/J2;
          B[7][0] = -1/m1;
          B[9][1] = -1/m2;
          B[11][2] = -1/m3;
          B[13][3] = -1/m4;
}
void MM(float ** matA, float ** matB, float ** Ans , int first, int mid, int last){

          //cout << "size of matA" << sizeof matA/sizeof matA[0] << " anything behind me?" << endl;
          zero(Ans, first, last);
          for(int i = 0; i < first  ; i++){
                    for(int j = 0; j < last ; j++){
                              for(int k = 0; k < mid ; k++){
                                        Ans[i][j] = Ans[i][j] + (matA[i][k]* matB[k][j]);
                              }
                    }
          }

}
void add(float ** matA, float ** matB, float ** Ans , int rsize, int csize){
          //zero(Ans, rsize, csize);
          for(int i = 0; i < rsize ; i++){
                    for(int j = 0; j < csize ; j++){
                              Ans[i][j] = matA[i][j] + matB[i][j];
                    }
```

```
	}

}
void scalarM(float scalar, float ** mat, float ** Ans, int rsize, int csize){
	for(int i = 0; i < rsize ; i++){
		for(int j = 0; j < csize ; j++){
			Ans[i][j] = scalar*mat[i][j];
		}
	}

}
void sub(float ** matA, float ** matB, float ** Ans , int rsize, int csize){
	for(int i = 0; i < rsize ; i++){
		for(int j = 0; j < csize ; j++){
			Ans[i][j] = matA[i][j] - matB[i][j];

		}
	}

}
void transpose(float ** mat, float ** Ans ,int rsize, int csize){
	zero(Ans, rsize, csize);
	for(int i = 0; i<csize; i++)
		for(int j = 0; j < rsize; j++)
			Ans[j][i] = mat[i][j];


}
float calculateE(float * param, float * yr01, float * yr02, float * yr03, float * yr04){
	float u1, u2, u3, u4;
	float yp[15];
	float yncalc[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
	yncalc[1] = 0.01;
	float sum = 0;
	float tempvec1[15];
	float tempvec2[15];
	float Qcomponent;
	float Rcomponent;
	float uvec[5] = {0,0, 0, 0, 0};
	dtn = 0.000001;

	for(long k = 0; k < neareps; k++)
	{

		in1 = kt*yr01[k]/m1;
		in2 = kt*yr02[k]/m2;
		in3 = kt*yr03[k]/m3;
		in4 = kt*yr04[k]/m4;


		u1=-(param[1]*pow(yncalc[1]-l1*yncalc[3]-l3*yncalc[5]-
yncalc[7],3)+param[2]*(yncalc[2]+l1*yncalc[4]-l3*yncalc[6])
			+param[3]*pow(yncalc[7]-in1,3)+param[4]*yncalc[8]);

		u3=-(param[1]*pow(yncalc[1]-l1*yncalc[3]+l3*yncalc[5]-yncalc[11],3)+param[2]*(yncalc[2]-
l1*yncalc[4]-l3*yncalc[6])
			+param[3]*pow(yncalc[11]-in2,3)+param[4]*yncalc[12]);

		u2=-(param[5]*pow(yncalc[1]+l2*yncalc[3]+l4*yncalc[5]-
yncalc[9],3)+param[6]*(yncalc[2]+l2*yncalc[4]+l4*yncalc[6])
			+param[7]*pow(yncalc[9]-in3,3)+param[8]*yncalc[10]);

		u4=-(param[5]*pow(yncalc[1]-l2*yncalc[3]+l4*yncalc[5]-yncalc[13],3)+param[6]*(yncalc[2]-
l2*yncalc[4]+l4*yncalc[6])
			+param[7]*pow(yncalc[13]-in4,3)+param[8]*yncalc[14]);


		yp[1] = yncalc[2];
		yp[2] = (a[0]*yncalc[1] + a[1]*yncalc[2] + a[2]*yncalc[3] + a[3]*yncalc[4] + a[4]*yncalc[5] +
a[5]*yncalc[6]
		+ a[6]*yncalc[7] + a[7]*yncalc[8] + a[8]*yncalc[9] + a[9]*yncalc[10] + a[10]*yncalc[11] +
a[11]*yncalc[12] \
		+ a[12]*yncalc[13] + a[13]*yncalc[14] + (u1 + u2 + u3 + u4)/MASS);

		yp[3] = yncalc[4];
		yp[4] = (a[14]*yncalc[1] + a[15]*yncalc[2] + a[16]*yncalc[3] + a[17]*yncalc[4] + a[18]*yncalc[5]
		+ a[19]*yncalc[6] + a[20]*yncalc[7] + a[21]*yncalc[8] + a[22]*yncalc[9] + a[23]*yncalc[10] +
a[24]*yncalc[11]
		+ a[25]*yncalc[12] + a[26]*yncalc[13] + a[27]*yncalc[14] +(- l1*u1 + l2*u2 - l1*u3 + l2*u4)/J1);

		yp[5] = yncalc[6];
		yp[6] = (a[28]*yncalc[1] + a[29]*yncalc[2] + a[30]*yncalc[3] + a[31]*yncalc[4] + a[32]*yncalc[5]
		+ a[33]*yncalc[6] + a[34]*yncalc[7] + a[35]*yncalc[8] + a[36]*yncalc[9] + a[37]*yncalc[10] +
a[38]*yncalc[11]
		+ a[39]*yncalc[12] + a[40]*yncalc[13] + a[41]*yncalc[14] +(- l3*u1 - l3*u2 + l4*u3 + l4*u4)/J2);

		yp[7] = yncalc[8];
		yp[8] = (a[42]*yncalc[1] + a[43]*yncalc[2] + a[44]*yncalc[3] + a[45]*yncalc[4] + a[46]*yncalc[5]
			+ a[47]*yncalc[6] + a[48]*yncalc[7] + a[49]*yncalc[8] + in1 + u1/m1);

		yp[9] = yncalc[10];
		yp[10] = (a[50]*yncalc[1] + a[51]*yncalc[2] + a[52]*yncalc[3] + a[53]*yncalc[4] + a[54]*yncalc[5]
			+ a[55]*yncalc[6] + a[56]*yncalc[9] + a[57]*yncalc[10] + in2 + u2/m2);

		yp[11] = yncalc[12];
```

```
                    yp[12] = (a[58]*yncalc[1] + a[59]*yncalc[2] + a[60]*yncalc[3] + a[61]*yncalc[4] + a[62]*yncalc[5]
                              + a[63]*yncalc[6] + a[64]*yncalc[11] + a[65]*yncalc[12] + in3 + u3/m3);

                    yp[13] = yncalc[14];
                    yp[14] = (a[66]*yncalc[1] + a[67]*yncalc[2] + a[68]*yncalc[3] + a[69]*yncalc[4] + a[70]*yncalc[5]
                              + a[71]*yncalc[6] + a[72]*yncalc[13] + a[73]*yncalc[14] + in4 + u4/m4);


                    yncalc[1] = yncalc[1]+yp[1]*dtn;
                    yncalc[2] = yncalc[2]+yp[2]*dtn;

                    yncalc[3] = yncalc[3]+yp[3]*dtn;
                    yncalc[4] = yncalc[4]+yp[4]*dtn;

                    yncalc[5] = yncalc[5]+yp[5]*dtn;
                    yncalc[6] = yncalc[6]+yp[6]*dtn;

                    yncalc[7] = yncalc[7]+yp[7]*dtn;
                    yncalc[8] = yncalc[8]+yp[8]*dtn;

                    yncalc[9] = yncalc[9]+yp[9]*dtn;
                    yncalc[10] = yncalc[10]+yp[10]*dtn;

                    yncalc[11] = yncalc[11]+yp[11]*dtn;
                    yncalc[12] = yncalc[12]+yp[12]*dtn;

                    yncalc[13] = yncalc[13]+yp[13]*dtn;
                    yncalc[14] = yncalc[14]+yp[14]*dtn;


                    ytemp1[k] = yncalc[1];


                    for(int i = 0; i <14; i++)
                    {
                              tempvec1[i] = 0;
                              tempvec2[i] = 0;
                    }

                    Qcomponent = 0;
                    Rcomponent = 0;
                    uvec[1]= u1;
                    uvec[2]= u2;
                    uvec[3]= u3;
                    uvec[4]= u4;

                    for(i = 0; i <14; i++)
                              for(int j = 0; j < 14; j++)
                                        tempvec1[i] = tempvec1[i]+yncalc[j+1]*Q[j][i];

                    for(i = 0; i <14; i++)
                              Qcomponent = Qcomponent + tempvec1[i]*yncalc[i+1];


                    for(i = 0; i <4; i++)
                              for(int j = 0; j < 4; j++)
                                        tempvec2[i] = tempvec2[i]+uvec[j+1]*R[j][i];

                    for(i = 0; i <4; i++)
                              Rcomponent = Rcomponent + tempvec2[i]*uvec[i+1];

                    sum = sum + Qcomponent + Rcomponent;


                    if(isnan(sum))//sum != sum)
                    {
                              printf("NaN Error%f\t%f\n",yncalc[1],sum);
                              return eub;
                    }

          }
          return sum;
}
```

Bibliography

[1] Akhtar, J., "Virtual Reality: Effective Surroundings, Enormous Demonstration and Mediator System in the Games, Industrial Design and Manufacturing", Blekinge Institute of Technology, June 2008

[2] Kim, J.H., Park, S.T., Lee, H., Yuk, K.C., and Lee, H., "Virtual Reality Simulations in Physics Eduction", Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 2001

[3] Burdea, G., Coiffet, P.,Virtual Reality Technology, John Wiley and Sons, Inc., New York, NY 1994

[4] Sherman, W. R., Craig, A. B., Understanding Virtual Reality, Morgan Kaufmann Publishers, New York, NY 2003

[5] Schirski, M., Bischof, C. and Kuhlen, T., "Interactive Exploration of Large Data in Hybrid Visualization Environments", IPT-EGVE 2007 - EG/ACM Symposium Proceedings, Eurographics Association, Aire-la-Ville, 2007

[6] Waldner, M., Kalkusch, M., and Schmalstieg, D., "Optical Magic Lenses and Polarization-Based Interaction Techniques", IPT-EGVE 2007 - EG/ACM Symposium Proceedings, Eurographics Association, Aire-la-Ville, 2007

[7] Burdea, G., Coiffet, P.,Virtual Reality Technology, John Wiley and Sons, Inc., New York, NY 2003

[8] Brill L., "Metaphors for the Travelling Cybernaut", Virtual Reality World Vol.1, No.1, Spring 1993

[9] Biocca, F., Levy, M., Communication in the Age of Virtual Reality, Lawrence Erlbaum Associates, Inc. , Hillsdale, NJ 1995

[10] Maxwell, Bruce, "Physical Systems Analysis: Lab #4 Speed Bumps", Swarthmore College, webpage, http://palantir.swarthmore.edu/maxwell/classes/e12/S04/labs/lab04/

[11] Gardner, John, "The Quarter-Car Model", Boise State University, webpage, http://coen.boisestate.edu/jgardner/ME474/QuarterCar.doc

[12] Rao, S. S., Mechanical Vibrations, Second Edition., Addison Wesley Publishing Company, Reading, MA, 1990, pgs. 144-145, 182-186, 228-246, 269-273

[13] Vakakis, A.F., "Inducing Passive Nonlinear Energy Sinks in Vibrating Systems", Journal of Vibration and Acoustics, 2001, Vol. 123, p324-332

[14] Zheng, L., Li, Y.N., Baz, A., "Fuzzy-Sliding Mode Control of Semi-Active Syspension Systems with MR Dampers", World Scientific, 2006

[15] Yao, Yi-Sheng; Chellappa, Rama, "Estimation of vehicle dynamics from monocular noisy images," University of Maryland, Center for Automation Research, Computer Vision Laboratory, College Park, MD, 1993

[16] Taghirad, H.D. and Esmailzadeh, E, "Automobile Passenger Comfort Assured Through LQG/LQR Active Suspension", Journal of Vibration and Control, Vol. 4, 1997, p603-618

[17] Yoahimura, T., Sagimori, K., and Hino, J., "Active suspension of a half car model based on linear control with dynamic absorbers", International Journal of Vehicle Design, Vol. 25, No. 4, 2001, pgs. 283-294

[18] Zheng, L., Baz, A., "Control of Vehicle Suspension Using Nonlinear Energy Sink Controller", International Journal of Vehicle Noise and Vibration, Vol. 3, Num. 1, 2007 , pp. 27-45

[19] Zheng, L., Li, Y.N., Baz, A., "Improving Vehicle Performance with Nonlinear Energy Sink Controller", Journal of Vehicle Dynamics, 2007

[20] Lu, J., "Multiobjective Optimal Suspension Control to Achieve Integrated Ride and Handling Performance", IEEE Transactions on Control systems Technology, Vol. 10, No. 6, 2002

[21] Kruzek, A. and Stribrsky, A., "A Full Car Model for Active Suspension – Some Practical Aspects", Proccedings of IEEE International Conference on Mechatronics, Istanbul, 2004

[22] Kim, C and Ro, P.I., "An Accurate Full Car Ride Model Using Model Reducing Techniques", Journal of Mechnical Design, Vol. 124, 2002

[23] Groom, N.J., Simplified Analytical Model of a Six-Degree-of Freedom Large-Gap Magnetic Suspension System", NASA Technical Memorandum 112868, 1997

[24] Groom, N.J. and Schaffner, P.R.., "An LQR Controller Design Approach for a Large Gap Magnetic Suspension System (LGMSS)", NASA Technical Memorandum 101606, 1990

[25] Close, Charles M., Fredrick, Dean K., Newell, Jonathan C., Modeling and Analysis of Dynamic Systems, Third Edition, John Wiley and Sons, Inc., Hoboken, NJ 2002

[26] "Mitsubishi Motors History", Mercedez Benz South Africa, webpage http://www.mitsubishi-motors.co.za/featuresites/mm_history/1980-1989.asp

[27] "Delphi Magne Ride", Delphi, http://delphi.com/shared/pdf/ppd/chsteer/magneride.pdf

[28] Hrovat, D., "Application of Optimal Control to Advanced Automotive Suspension Design", ASME Journal of Dynamic Systems, Measurement, and Control, 1993, p328-342

[29] Willims, D.E. and Haddan, W.M., "Active Suspension Control to Improve Ride and Handling", Vehicle System Dynamics, Vol 28, 1997,p1-24

[30] Smith, M.C. and Walker, G.W., "Performance Limitations and Constraints for Active and Passive Suspensions: A Mechanical Multi-Port Approach", IEEE Transactions on Control Systems Technology, 2002, Vol. 10, p393-407

[31] Smith, M.C. and Wang, F.C., "Controller Parameterization for Disturbance Response Decoupling: Application to Vehicle Active Suspension Control", IEEE Transactions on Control Systems Technology, 2002, Vol. 10, p393-407

[32] Sam, Y.M., Ghani, M.R.H.A., Ahmad, N., "LQR Controller for Active Car Suspension", TENCON 2000. Proceedings, Vol. 1, 2000, p441-444

[33] Vakakis, A.F., "Shock Isolation Through the Use of Nonlinear Energy Sinks", Journal of Vibration and Acoustics, 2001, Vol. 123, p324-332

[34] Potter, J.E., "Matrix Quadratic Solutions", SIAM Journal of Applied Mathematics, Vol. 14, 1966, p496-501

[35] Laub, A.J., "A Schur Method for Solving Algebraic Riccati Equations" , IEEE Transactions on Automatic Control, AC-24, 1979, p913-921

[36] Kleinman, D.L., "On an Iterative Technique for Riccati Equations Computations", IEEE Transactions on Automatic Control, AC-13, 1968, p114-115

[37] Smith, R.A., "Matrix Equation XA + BX = C", SIAM Journal of Applied Mathematics, Vol. 16, 1968, p198-201

[38] Banks, H.T., and Ito, K., "A Numerical Algorithm for Optimal Feedback Gains in High Dimensional LQR Problems", NASA Contractor Report 178207, 1986

[39] Lewis, F.L., Syrmos, V.L., *Optimal Control*, Second Edition, John Wiley & Sons, Inc., New York, 1995

[40] Smoker, J., Baz, A, and Zheng, L., "Virtual Reality Simulation of a Full Car Active Suspension System with Nonlinear Energy Sink Controller", World forum on smart material and smart structures technology, May 2007

[41] Close, Charles M., Fredrick, Dean K., Newell, Jonathan C., *Modeling and Analysis of Dynamic Systems,* Third Edition, John Wiley and Sons, Inc., Hoboken, NJ 2002

[42] Thomson, William T., Dahleh, Marie Dillon, *Theory of Vibration with Applications,* Fifth Edition, Prentice Hall, Upper Saddle River, NJ, 1998 pgs. 126-143

[43] Moreau, J-P, "Integrate a System of Ordinary Differential Equations By the Runge Kutta Fehlberg Method", webpage, http://perso.wanadoo.fr/jean-pierre.moreau/c_eqdiff.html