2022

Effects on Vehicle Ride Comfort of an Adaptive Suspension System Using Neural Networks

Sylvia Yin Zhixian
University of Windsor

Effects on Vehicle Ride Comfort of an Adaptive Suspension System Using Neural Networks

By

Sylvia Yin Zhixian

A Thesis
Submitted to the Faculty of Graduates Studies
Through the Department of Mechanical, Automotive, & Materials Engineering
in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2022

Effects on Vehicle Ride Comfort of an Adaptive Suspension System Using Neural Networks

by

Zhixian Yin

APPROVED BY

N. Kar
Department of Electrical & Computer Engineering

J. Johrendt
Department of Mechanical, Automotive, & Materials Engineering

---

B. Minaker, Advisor
Department of Mechanical, Automotive, & Materials Engineering

August 31, 2022

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Suspension systems in the auto industry have always been a topic of interest, as they relate to so many aspects of vehicles. Various types of suspension are commonly used now, such as passive suspensions, semi-active suspensions and active suspensions. However, the current technology mainly focuses on the change of damping ratio. The aim of this thesis is to consider both spring and damper properties for suspensions of an off-road vehicle. In order to do this, a 10-degree of freedom model was built using the EoM software in Julia. The output state space matrices from EoM were used as an input in a Matlab/Simulink control loop to analyze the performance. A critical goal was to see the effects on ride comfort of using Neural Networks for property selection in the Matlab/Simulink control loop. By adding extra spring forces and damping forces, the level of ride comfort was modified. Preliminary road tests were conducted to serve as a proof of concept.

To our past, present and future.

ACKNOWLEDGEMENTS

"You can never cross the ocean until you have the courage to lose sight of the shore."

André Gide

For me, the whole journey in the University of Windsor was full of surprises. During these years, I met a lot of great people here, but also lost two devoted family members back in hometown China – my father and my grandfather. As with the saying in the quote, you can never lose your courage to cross the ocean. Canada is the place which bred my consistent interest for automotive engineering into my life and career. I would like to express my appreciation for my supervisor and thesis committee members for their ideas and feedback. It is impossible to finish my graduate studies without any one of you. Finally and particularly, my supervisor Dr. Bruce Minaker; his positive attitude to life, enthusiasm for knowledge and consideration of his students deeply affected and helped me. He is playing a role in my life like Mr. Keating in the movie Dead Poets Society.

This amazing journey could not happen without the support of my mom, not only for her financial support but also her encouragement. Even though the world will come to an end, my love to you will not.

Thanks for this fantastic world.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Motivation

It is commonly known that an off-road vehicle shows obvious advantages over a sedan or normal SUV under bad road or weather conditions. However, during normal daily driving, since most off-road vehicles use passive suspension, the ride quality is degraded when compared against a typical passenger sedan. The motions experienced by the driver include both side-to-side shaking and up-and-down bumps on roads with varying road conditions. Over time, this phenomenon can also cause damage to the vehicle. For example, the Jeep 'death wobble' vibration reported in the media and online that some owners experience is due to suspension or steering system fasteners that are slowly loosened or worn due to vibration in the original suspension.

In both Canada and the United States, there are many off-road vehicle enthusiasts, and as a result, there are many suspension products targeted at off-road vehicles on the North American market, for example, systems that provide the abilty to change the damper characteristics while driving. However, besides dampers, one might consider the possibility of dynamically changing the spring stiffness as well. This research focuses on simultaneous changes of parameters of both springs and dampers, to improve the flexibility of

suspension systems.

While manual control of these parameters would be a possibility, a far better solution would be the automatic selection of the appropriate parameters, using an automatic tuning system. It is notable that in recent years, *neural networks* and other machine learning techniques have been providing many advances in artificial intelligence technology. For this reason, the application of neural networks to make the selection of stiffness and damping ratios was explored. The goal was to see an improvement in both the ride comfort and vehicle safety.

1.2 Literature Review

1.2.1 Vehicle suspensions

The vehicle suspension system acts as the connector of the chassis (the 'sprung mass') and the wheel, tire and brake (the 'unsprung mass') and consists of linkages, springs, and shock absorbers (dampers). The suspension transmits the forces between the wheels and the vehicle body and passengers. The suspension springs support the body mass and isolate it from road disturbances, contributing to driver comfort. The dampers dissipate the relative motion, contributing to both driving safety and comfort[1].

Vehicle suspension types

For decades, vehicle suspension systems have been a popular topic for researchers. The importance of design of suspensions never fades. For the sake of driver and passenger safety, different types of vehicles must be equipped with proper suspension systems. Moreover, the suspension system also directly affects the vehicle handling, and depends strongly on the choice of spring stiffness and damping ratio.

In general, the types of vehicle suspension can be classified into three categories: *passive* suspensions, *active* suspensions and *semi-active* suspensions.

Passive suspension system

In this context, 'passive' indicates that the suspension elements cannot provide energy to the suspension system (see Figure 1.1). According to the targeted ride comfort and handling stability level, the passive suspension system limits the motion of the body and wheel by limiting their relative velocities to a rate that gives the required ride comfort. This is achieved by using some type of damping element placed between the body and the wheels of the vehicle, such as hydraulic shock absorbers[2].

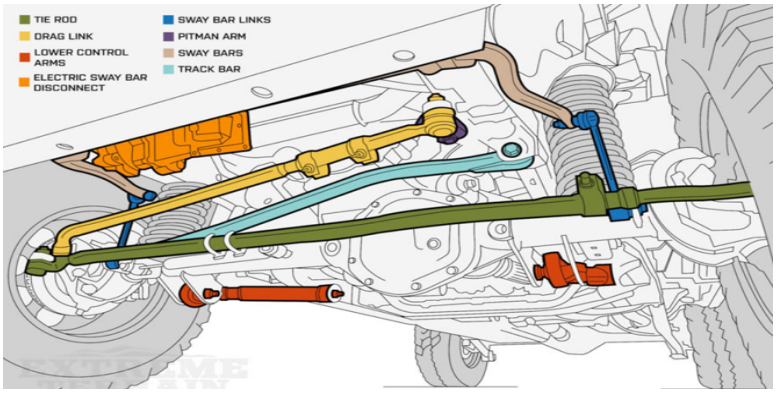

Figure 1.1: Structure of a passive suspension of an off-road vehicle. Image reproduced from www.wikipedia.org.

Active suspension systems

Active suspensions can be divided into two types: real active suspensions, and adaptive suspensions or so-called semi-active suspensions. It is typical for most active suspensions to imply varies kinds of actuators to raise and lower the chassis independently at each wheel. Adaptive suspensions change the firmness of the shock absobers in order to adapt the dynamic environment, for example, road disturbances, loading mass changes and weather conditions. There are many technologies to achieve this goal, for example Electro-Rheological and Magneto-Rheological fluids, solenoid-valves and piezoelectric actuators[1]. The difference between fully-active and semi-active suspension is whether the suspension system can generate forces between the vehicle body and wheels that are fully independent of the direction of suspension travel. The distinction is illustrated in Figure 1.2.

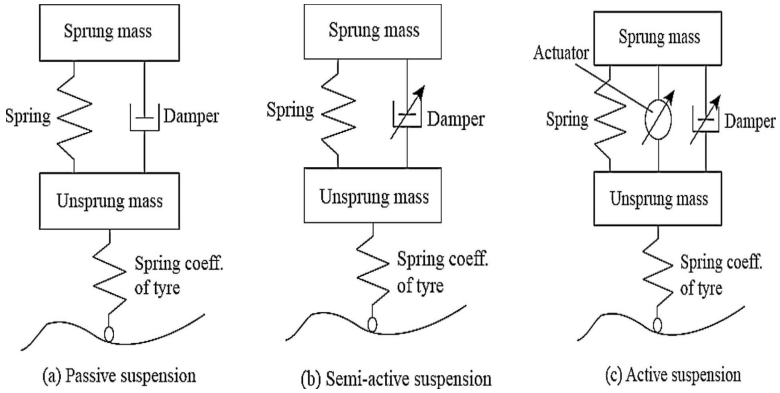

Figure 1.2: Quarter-car models illustrating the three categories of suspensions. Image reproduced from Omar[3].

1.2.2 Vehicle dynamics models

For certain research purposes, many classic vehicle models have been developed. For every model, different assumptions are made, e.g, differing degrees of freedom, according to the requirement of the sophistication of the model. Generally in a 3D space, a coordinate system is set at the vehicle mass center, where the x axis represents longitudinal dynamics, the y axis reflects lateral motions and the z axis stands for vertical movements of the vehicle. The rotation around the x, y, and z axes is called 'roll', 'pitch', and 'yaw', respectively.

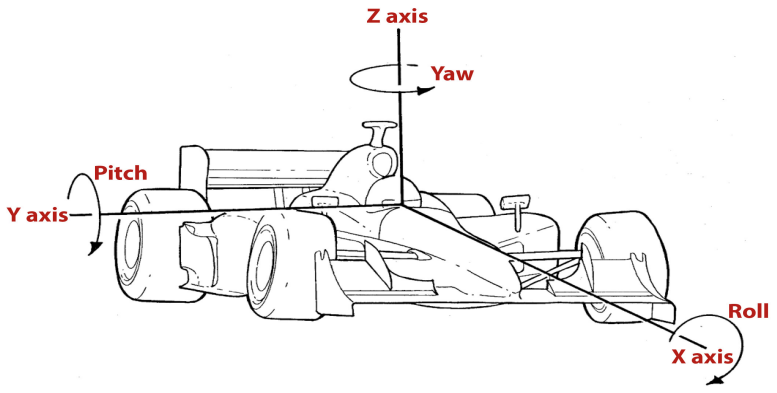

Figure 1.3: Motions of a vehicle – roll, pitch, and yaw. Image reproduced from www.racecar-engineering.com

The yaw plane model

The yaw plane model is also called the 'bicycle model'. In this case, the bicycle model has nothing to do with a real bicycle but the name is used because the vehicle width is ignored when the model is developed. The vehicle model is simplified to a two degrees of freedom model. Only the lateral velocity (v) and the yaw velocity (r, the angular velocity around the z axis) are considered, while the forward speed (u) of the vehicle is treated as constant. The condition of a constant forward speed is called a *nonholonomic*[1] constraint[4]. Several simplifying assumptions are made to result in a linear dynamic model, given in Eqn. 1.1. The critical assumption of this model is considering the width of the vehicle as unimportant during a the model development, and so both the left and right side tires are considered to have the same forward speed. Although the lateral weight transfer that occurs when the vehicle is cornering does play an important role as it influences the tire performance, the yaw plane model considers it as a secondary effect, and ignores it.

There are several common notations to be mentioned in this model: the tire cornering stiffness c_f and c_r, where 'f' represents the front and 'r' represents the rear; the lateral forces acting at each of the front and rear axles are Y_f and Y_r, respectively; the steering angle of the front tires, assumed to be the same on the left and right side is δ_f. Notice that the only moment of inertia that matters in this model is I_{zz}, because the yaw rotation is around the z axis.

Meanwhile, there are some interesting effects on building the Newton-Euler equations when studying the bicycle model. First, the longitudinal equation is irrelevant such that all the $\hat{\imath}$ component terms are eliminated. Second, while computing the cornering stiffness, one must double the amount as measured for a single tire, because in the bicycle model, although the width is neglected, it does have two tires at each axle.

$$\begin{bmatrix} m & 0 \\ 0 & I_{zz} \end{bmatrix} \begin{Bmatrix} \dot{v} \\ \dot{r} \end{Bmatrix} + \frac{1}{u} \begin{bmatrix} c_\mathrm{f} + c_\mathrm{r} & ac_\mathrm{f} - bc_\mathrm{r} + mu^2 \\ ac_\mathrm{f} - bc_\mathrm{r} & a^2 c_\mathrm{f} + b^2 c_\mathrm{r} \end{bmatrix} \begin{Bmatrix} v \\ r \end{Bmatrix} = \begin{bmatrix} c_\mathrm{f} \\ ac_\mathrm{f} \end{bmatrix} \{\delta_\mathrm{f}\} \tag{1.1}$$

[1] Nonholonomic systems are those where the number of position coordinates required exceeds the number of velocity coordinates. The yaw plane model has three position coordinates (x, y, ψ), but only two velocities (v, r).

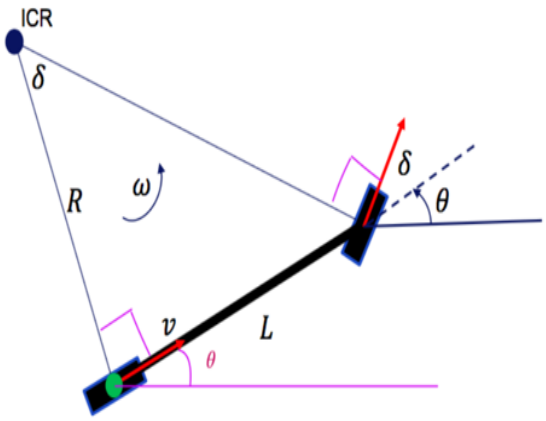

Figure 1.4: A bicycle model, reproduced from Escalona[5].

The bicycle model is typical for analyzing handling performance. It is necessary to calculate the yaw rate gain $\frac{r}{\delta_f}$, as it is important to define and predict whether the vehicle will *understeer* or *oversteer*. In the case of $ac_f < bc_r$, the vehicle is in an understeer condition. However, when $ac_f > bc_r$, which usually happens on a rear-heavy vehicle, is said to oversteer. There is special case when $ac_f = bc_r$, the vehicle is said to *neutral steer*.

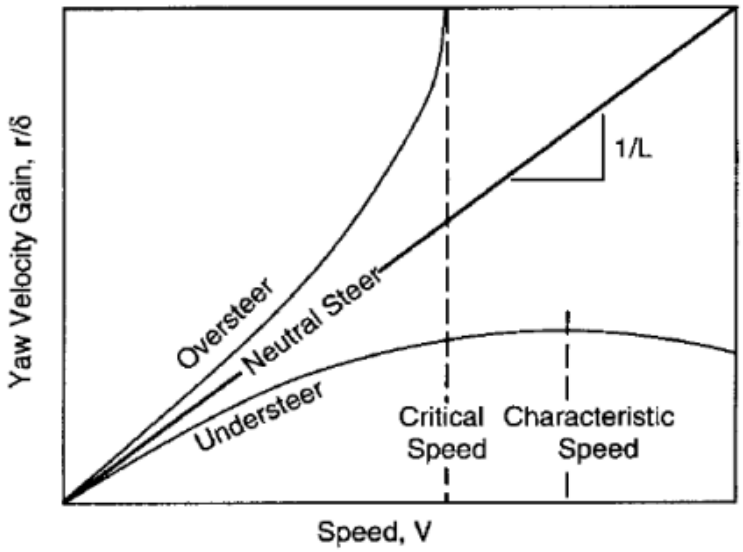

Figure 1.5: The steady state yaw rate gain $\frac{r}{\delta_f}$ is a function of speed.

In seen in Figure 1.5, the oversteering vehicle becomes increasing sensitive near critical the speed u_{crit}, while the understeering vehicle reaches a maximum sensitivity at the characteristic speed u_{char}.

The expressions for the characteristic speed and critical speed are given in Eqns. 1.2 and 1.3, respectively:

$$u_{\text{char}} = \sqrt{\frac{c_{\text{f}}c_{\text{r}}(a+b)^2}{m(bc_{\text{r}} - ac_{\text{f}})}} = \sqrt{\frac{g(a+b)}{k_{\text{us}}}} \tag{1.2}$$

$$u_{\text{crit}} = \sqrt{\frac{c_{\text{f}}c_{\text{r}}(a+b)^2}{m(ac_{\text{f}} - bc_{\text{r}})}} = \sqrt{\frac{g(a+b)}{-k_{\text{us}}}} \tag{1.3}$$

where k_{us} is known as the *understeer gradient.*

The quarter car model

One of the classic vehicle models that is well known for predicting ride quality is the *quarter car* model. It is a simple two degree of freedom model, with two bodies constrained to vertical translation, representing the sprung mass m_{s} (the chassis, powertrain, driver, cargo, etc.) and the unsprung mass m_{u} (the wheel, hub, brake rotor or drum, etc.)[4].

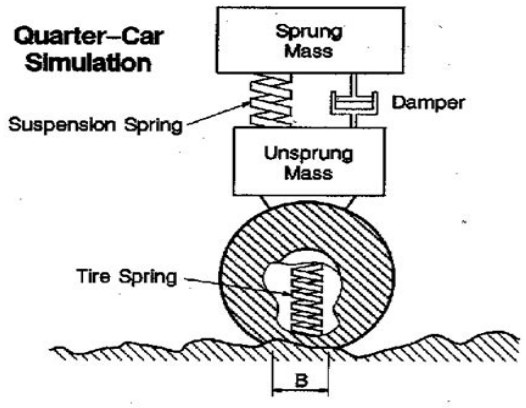

Figure 1.6: A quarter car model, reproduced from www.wikipedia.org

The model is formed by writing Newton's equation for each of the two bodies. For the unsprung mass:

$$m_s\ddot{z}_s + c_s\dot{z}_s - c_s\dot{z}_u + k_s z_s - k_s z_u = 0 \tag{1.4}$$

For the unsprung mass:

$$m_u\ddot{z}_u + c_s\dot{z}_u - c_s\dot{z}_s + k_s\dot{z}_u - k_s z_s + k_t z_u = k_t z_g \tag{1.5}$$

The two equations are combined as a vector equation:

$$\begin{bmatrix} m_\mathrm{s} & 0 \\ 0 & m_\mathrm{u} \end{bmatrix} \begin{Bmatrix} \ddot{z}_\mathrm{s} \\ \ddot{z}_\mathrm{u} \end{Bmatrix} + \begin{bmatrix} c_\mathrm{s} & -c_\mathrm{s} \\ -c_\mathrm{s} & c_\mathrm{s} \end{bmatrix} \begin{Bmatrix} \dot{z}_\mathrm{s} \\ \dot{z}_\mathrm{u} \end{Bmatrix} + \begin{bmatrix} k_\mathrm{s} & -k_\mathrm{s} \\ -k_\mathrm{s} & k_\mathrm{s} + k_\mathrm{t} \end{bmatrix} \begin{Bmatrix} z_\mathrm{s} \\ z_\mathrm{u} \end{Bmatrix} = \begin{bmatrix} 0 \\ k_\mathrm{t} \end{bmatrix} \begin{Bmatrix} z_\mathrm{g} \end{Bmatrix} \tag{1.6}$$

or:

$$\mathbf{M}\ddot{\boldsymbol{z}} + \mathbf{L}\dot{\boldsymbol{z}} + \mathbf{K}\boldsymbol{z} = \mathbf{F}\boldsymbol{u} \tag{1.7}$$

In the transient condition, when a vehicle is driving on the road with disturbances, the most obvious motion is the vertical motion, as it is directly influenced by the road input. A quarter car model is suitable for analyzing the frequencies of both the sprung mass and the unsprung mass. Typically, the damping terms are omitted, as they have a relatively small effect on the resulting frequencies of motion, and this simplification allows a sinusoidal solution to the equations. Suppose that the road is smooth (i.e., $z_\mathrm{g} = 0$), and that the motions can be written:

$$\boldsymbol{z} = \begin{Bmatrix} z_\mathrm{s} \\ z_\mathrm{u} \end{Bmatrix} = \boldsymbol{z}_0 \cos(\omega \boldsymbol{t}) \tag{1.8}$$

As a result,

$$\ddot{z} = -\omega^2 z_0 \cos(\omega t) = -\omega^2 z \tag{1.9}$$

The equation of motion is then:

$$-\omega^2 \mathbf{M}z + \mathbf{K}z = 0 \tag{1.10}$$

or:

$$[\mathbf{K} - \omega^2 \mathbf{M}]z = 0 \tag{1.11}$$

With an examination of the resulting equation, one should recognize that it is effectively an eigenvector problem, and therefore must have a singular coefficient matrix. Expanding the determinant of this matrix term gives the characteristic equation:

$$(k_\mathrm{s} - \omega^2 m_\mathrm{s})(k_\mathrm{s} + k_\mathrm{t} - \omega^2 m_\mathrm{u}) - k_\mathrm{s}^2 = 0 \tag{1.12}$$

After manipulation and some simplification, two simple linear equations that can be solved for ω. First,

$$\omega = \sqrt{\frac{k_\mathrm{s} + k_\mathrm{t}}{m_\mathrm{u}}} \tag{1.13}$$

And second:

$$-(k_\mathrm{s} + k_\mathrm{t}) m_\mathrm{s} \omega^2 + k_\mathrm{t} k_\mathrm{s} = 0 \tag{1.14}$$

$$\omega = \sqrt{\frac{k_s k_t}{(k_s + k_t) m_s}} \tag{1.15}$$

For typical values of the vehicle parameters, the lower frequency is typically about 2π rad/s or 1 Hz, while the higher is around 20π rad/s or 10 Hz. For the higher frequency, it shows a condition where the unsprung mass bounces against the suspension and tire as

two parallel springs while the sprung mass is relatively stationary. However, the lower frequency works inversely as the sprung mass bounces against the suspension and tire as two springs in series, and the unsprung mass is ignored. The two motions are typically called *wheel hop* mode (high frequency) and the *bounce* or *heave* mode (low frequency). A property of the quarter car model is that the motions tend to be very discrete; the low frequency is associated almost entirely with the vehicle body motion and the high frequency with the wheel[4]. The frequency response of a typical quarter car model is shown in Figure 1.7.

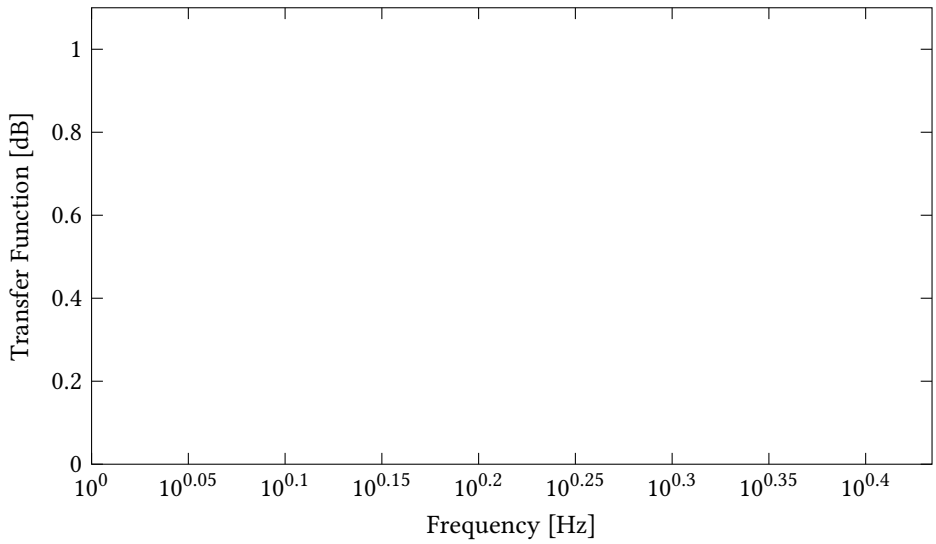

Figure 1.7: Frequency response of the quarter car model, showing body motion at low frequencies, suspension travel at midrange frequencies, and tire compression at high frequencies.

1.2.3 Full car model

As its name, a full car model considers bounce, pitch, and roll motions of the vehicle body. Typically a full car model is used extensively to study the steady state cornering on various road profiles which includes a range of smooth to rough road surfaces. In the free body diagram of the full call car model has vertical and lateral forces acting at each of the tire contact patches. In regard to many studies on suspension system, it is researcher's choice of how many degrees of freedom a model has, according to their own research

interests. Meanwhile, since it has always hard for researchers to predict performance of tires, the famous 'Magic Formula tire model' is widely use to do such a job. However, in order to validate the results from the magic formula, a test rig based on a full car model is necessary for taking the actual performance data of tires. A schematic diagram of a typical full car model is shown in Figure 1.8.

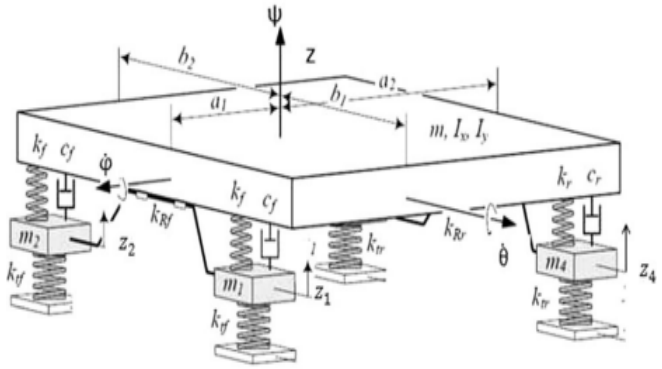

Figure 1.8: A full car model. Figure reproduced from Jazar[6]

Steady state analysis

When doing this analysis, assumptions has to made first. Assume the acceleration is constant and the vehicle is treated as a single rigid body. If one assumes that the vehicle is cornering in steady state, then the lateral acceleration $\dot{v} + ur$ simplifies to just ur. Moreover, the roll rate and pitch rate is assumed to be zero while the yaw rate is constant. In this case, the equation of the vertical motion and the stiffness of the system is fairly important as the system is 'statically indeterminate', where the vertical motion at any point is assumed to be equilibrium:

$$\sum Z = -Z_{rf} - Z_{lf} - Z_{rr} - Z_{lr} + mg = m\ddot{z} = 0 \tag{1.16}$$

The reference point to build an axis in the ground plane to evaluate the roll moment is not at the same location as where the center of mass is. In this way, additional inertial moment is generated and the assumption keeps the angular acceleration of the roll to be zero such that to describe the steady state.

$$\sum L = \frac{t_f}{2}(Z_{lf} - Z_{rf}) + \frac{t_r}{2}(Z_{lr} - Z_{rr}) = I_{xx}\dot{p} + murh_G = murh_G \tag{1.17}$$

For the pitch motion, remembering the pitch rate (angular acceleration in the pitch direction) is assumed to be zero when the vehicle is still moving at a constant speed. Although the vertical offset off the reference axis at the center of mass does not has any effects, the weight distribution of the vehicle can add an non-zero I_{xz} to the equation when the sprung mass at the rear is significantly higher than that in the front. The term of I_{xz} can be positive when the rear of the vehicle carries its mass further above the ground. However, even this configuration combined with a angular velocity around the z axis produce a pitching moment which tends to lift the vehicle front, the effect is still small usually. Now equation for the pitch motion is:

$$\sum M = a(Z_{rf} + Z_{lf}) - b(Z_{rr} + Z_{lr}) = I_{yy}\dot{q} - I_{xz}r^2 = -I_{xz}r^2 \tag{1.18}$$

Considering the normal forces, which are assumed to have a linear relationship with the suspension compression, the coupling moments across each anti-roll bar has to be taken into account. Nevertheless, as the tire stiffness is significantly higher than the suspension stiffness such that the tire stiffness can be neglected, not to say the stiffness of two springs in series is less than one individual spring. The deflection of the anti-roll bar caused by force transferred from the wheel is also calculated under the assumption that the stiffness of the anti-roll bar is a linear stiffness. If quoting the roll stiffness as a moment per unit of rotation, the conversation factor is the square of the track width (e.g., k_{fb} [Nm/rad] = $t^2 k_{fb}$ [N/m]).

$$Z_{rf} = k_f z_{rf} + k_{fb}(z_{rf} - z_{lf}) \tag{1.19}$$

$$Z_{lf} = k_f z_{lf} + k_{fb}(z_{lf} - z_{rf}) \tag{1.20}$$

$$Z_{rr} = k_r z_{rr} + k_{rb}(z_{rr} - z_{lr}) \tag{1.21}$$

$$Z_{lr} = k_r z_{lr} + k_{rb}(z_{lr} - z_{rr}) \tag{1.22}$$

1.2.4 Ride comfort

Ride comfort standards

As a general rule, customer demands for vehicles are not only limited to driving performance on various terrains, but also related to the performance. Performance is a general term in the automotive industry, and can include many standards. Among all these standards, 'ride comfort' and 'handling performance' are most widely known. This thesis is more focused on ride comfort than on the handling performance while both are considered.

When vehicle engineers try to develop a suspension system that can isolate the passengers from road disturbances, it often results in a sacrifice of the vehicle handling performance. The parameter selection for ride comfort and vehicle handling is a compromise. In an effort to balance the effects, much research has been done with the aim of maximizing the ride comfort, while keeping the handling performance sufficient to ensure driver safety.

According to Karnopp[7], vehicle ride comfort should be considered in terms of frequencies, and the key to the suspension design is to isolate the body from high road input frequencies. Meanwhile, at lower frequencies, the accelerations of the wheels and the driver more directly correspond to the road input. The suspensions should be sufficiently damped to control resonance such that the road disturbances are not amplified, and that wheel hop and loss of wheel contact with the road is restricted. Apart from this, when the payload, or the forces generated from braking and cornering, or aerodynamic forces change, the suspension should have the ability to control the energy translation to the human body. For example, in a passive suspension system, all these requirements are not feasible because a passive suspension does not include any form of energy regeneration. The parameters of the suspension are set and cannot be changed under real driving

conditions, while an active system or semi-active system implies a dynamic energy management so that it is capable of affecting the ride comfort level. A typical driver comfort response is shown in Figure 1.9.

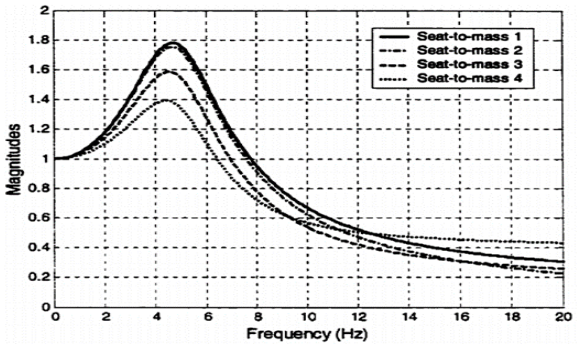

Figure 1.9: Frequency response of the disturbance transmitted through the vehicle to the driver.

As scientists and researchers have proposed many different standards to evaluate ride comfort, the limits of acceptable ride comfort vary. In North America, the method of calculating the *AAP* - the Average Absorbed Power[8] is common. The human body is considered as elastically-behaved, absorbing forces from each displacement of the car until all the energy is dissipated. During this process, the ratio of energy to time is calculated using the Fast Fourier transform or the Laplace transformation. Graphically, the x axis usually stands for frequency when the y axis stands for magnitude or phase calculated from the comfort matrix. A full car model similar to the one in Figure 1.10 is used as part of this process.

Ride comfort tests

There are some parameters that should be considered during ride evaluation events. As mentioned in the last section on ride comfort standards, vehicle ride comfort should be considered at levels of frequencies. Usually, ride tests are conducted mainly in two road input frequency ranges: 0.5–3 Hz and 5–20 Hz. It is meaningful to test within two separate frequency ranges as two peaks are expected to show in the result graph at two natural frequencies of the system. The time of energy dissipation is the main data to acquire and usually for a certain road profile there will be two sets of data: the energy

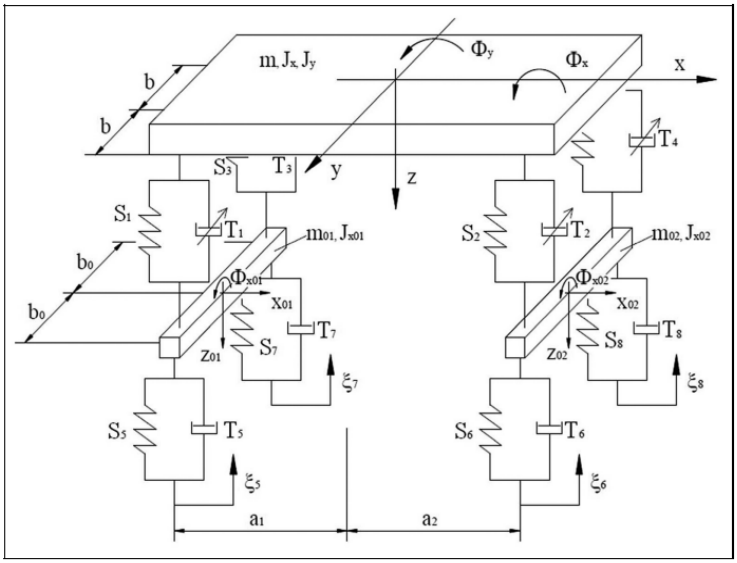

Figure 1.10: Model of a vehicle used to select optimal damping force in semi-active shock absorbers. Figure reproduced from Makowski[9].

dissipation time for the peak occurring under 5 Hz and the second peak between 5–20 Hz. In virtual analysis of ride comfort using software such as ADAMS, the input road profiles tested typically include a cleat road test, an English road test and a geddes road test. However, the plots of accelerations in longitudinal and vertical direction are similar. A typical ADAMS cleat test is shown in Figure 1.11.

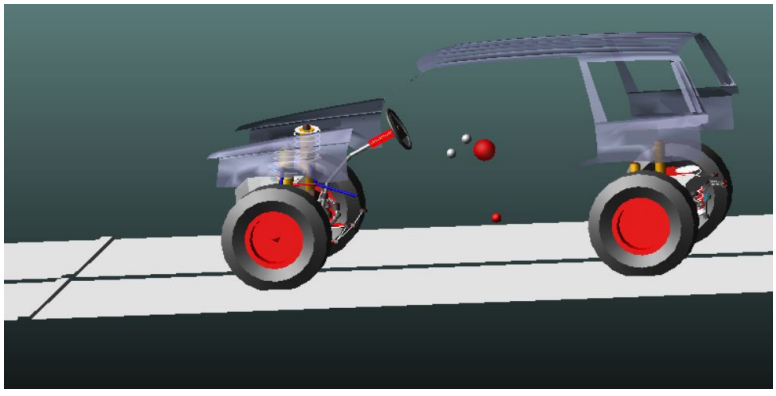

Figure 1.11: A sample of cleat ride comfort analysis in ADAMS. The size of the obstacle is 100 mm x 25 mm

1.3 An introduction of AI technology development and artificial neural networks

Artificial intelligence has been developed for many years, and it is applied widely for modern living. For example, during the lock-down period caused by the ongoing Covid-19 pandemic, AI technology is playing an important role, in quickly diagnosing whether a person is infected or not through analysis of the chest CT image. Going back to the 20th century, a great question was raised by a famous scientist – Alan Turing. The question is: 'Can machines think like human beings?' Afterwards, the famous *Turing test* was born. To pass the test, by communicating in words between humans and machines, the machines will be recognized as AI if over 30% of human testers cannot realize the 'person' they were talking with was a machine.

In 1956, the definition of 'Artificial Intelligence' was validated by three famous scientists, Marvin Lee Minsky, John McCarthy, and Claude Shannon, during the 'Dartmouth Summer Research Project on Artificial Intelligence'[10]. Since then, AI technology has experienced three revolutions and two recessions. Today, society is experiencing the third revolution. In 1997, IBM developed the first chess-playing computer – Deep Blue. It became the first computer to win both a chess game and a chess match against 12-year reigning world champion Garry Kasparov. Moreover, the algorithms of artificial intelligence were proposed by many great scientists, such as the 'Back Propagation (BP)' algorithm created by Geoffrey Hinton from University of Toronto, the 'Convolutional Neural Networks (CNN)' by Yann LeCun from New York University, and 'Deep Learning', 'Neural Machine Translation', 'Attention Model', etc., by Yoshua Bengio from University of Montreal[11].

As a core, the most usual algorithm in AI technology is the Artificial Neural Network (NN), which is conceptually similar to the biological structure of a human brain. In NN, every circular unit in the structure is called a *neuron*. This structure can be separated into two main regions: a region for receiving and processing incoming information from other cells (inputs) and a region for conducting and transmitting information to other cells (outputs), as shown in Figure 1.12.

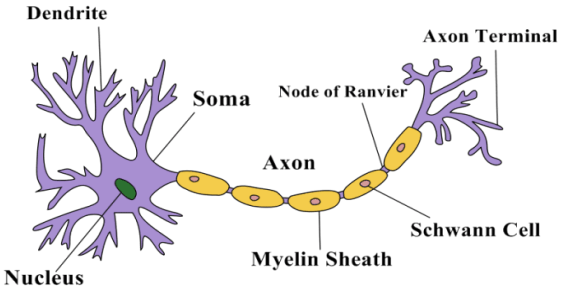

Figure 1.12: Structure of a human neuron. Figure reproduced from Sinkov[12].

The first mathematical model of a neuron, called the 'McCulloch-Pitts Neuron' (MP-Neuron), was created in 1943 by Warren Sturgis McCulloch and Water Pitts[11]. They defined every neuron as a multi-input and single-output system, as in Figure 1.13. However, there were problems that came with this model. First, it has fixed weights and thresholds, such that the system cannot learn. Second, it was a great challenge to minimize size of a MP-neurons network. Third, it was not suitable for cases where the process is non-discrete or non-binary. In 1986, the term 'Back-propagation' was announced by Rumelhart, Hinton and Williams[11], as illustrated in Figure 1.14. Instead of calculating the whole system from inputs to outputs, it computes the gradient from outputs to inputs. It significantly decreased the complexity of calculation and improved Neural Network performance to a new level. By introducing the loss function into the calculation, it keeps changing the values of the weights until they are optimized. Moreover, it avoids duplicated calculations and unnecessary intermediate values.

Activation functions

An activation function is also called a transfer function. It decides whether a neuron should be activated or not, like yes or no. For example, depending upon the function, it maps the resulting values in between 0 to 1 or −1 to 1 etc. To explain this in another way, an activation function decides whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations. In nature, a function can be linear or nonlinear which is the same as the activation functions in NN. The linear activation function, also known as 'no activation' or 'identity function'

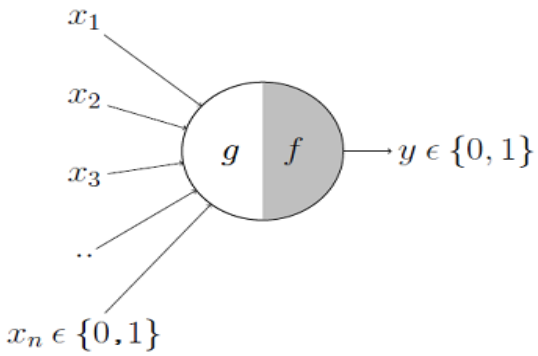

Figure 1.13: McCulloch-Pitts Neuron. Figure reproduced from Sinkov[12].

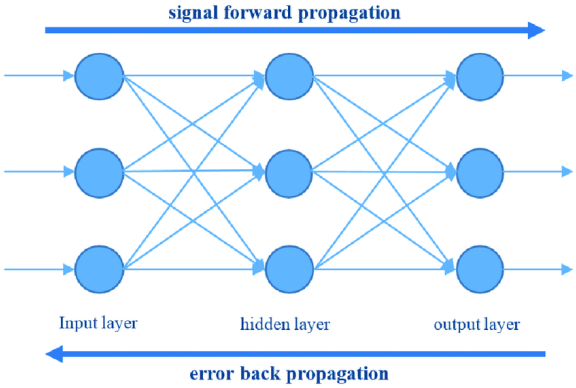

Figure 1.14: Backpropagation structure. Figure reproduced from Jiang[13].

(multiplied by 1.0), is where the activation is proportional to the input. The function itself doesn't do anything to the weighted sum of the input, but simply returns the value it was given. Mathematically it can be represented as:

$$F(x) = x \tag{1.23}$$

Due to the simplicity of a linear activation function, it has two major drawbacks in comparison with a nonlinear activation function. First, it is unable to back-propagate data in the network, as the derivative of a linear function is a constant value. In this case, the derivative has nothing to do with the input value so that it cannot process even at the first level of derivatives. Another problem is, all layers of the neural network will collapse into

one if a linear activation function is used. No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer. So, essentially, a linear activation function turns the neural network into just one layer [14].

The most widely used activation function type is nonlinear. It allows backpropagation since the input is still needed in the derivative function. In this case it is possible to go back and understand which weights in the input neurons can provide a better prediction. Meanwhile, it is also capable of including multiple layers of neurons, where the network can be potentially developed to a deep neural network. By allowing a non-linear combination of input passed through multiple layers, any output can be represented as a functional computation in a neural network.

One of the most widely used transfer function been used by people is the sigmoid function, represented as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.24}$$

The reasons for its popularity are obvious. In the case of a neural network that is designed to predict the probability of a certain case, a sigmoid function makes the results clean and tidy because it has a range of output from 0 to 1. Moreover, due to the S-shape of the sigmoid function, the resulting gradients are also smooth. It helps keep continuous output values, unlike some of the other activation functions. Therefore, according to people's needs, the sigmoid function can be a great choice to become the activation/-transfer function. Another similar activation function is called Tanh Function (Hyperbolic Tangent). Mathematically it can be represented as:

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{1.25}$$

The tanh function is also a S-shape function but with the difference in output range of −1 to 1, see Figure 1.15. In tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to −1.0. It is different as it is an zero-centered function. This gives the advantage to tanh function over than sigmoid function. In other words, because the output of the

tanh activation function is in the range of negative 1 to positive 1, the output values can be easily mapped as strongly negative, neutral, or strongly positive. As a result, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

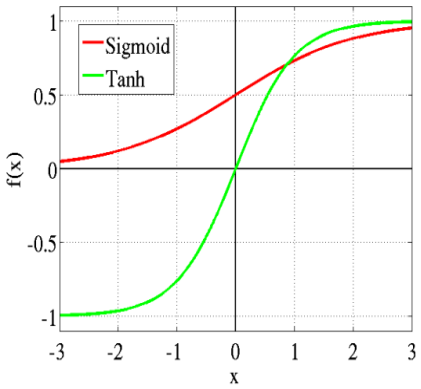

Figure 1.15: Tanh Function and Sigmoid function. Notice that the range is different.

During recent years, an activation function that is applied widely in convolutional networks and the deep learning area is called ReLU. ReLU stands for Rectified Linear Unit, the function shown in Figure 1.16. ReLU has an output range of zero to infinity. It earns its name as it is rectified at when input value equals 0. What makes ReLU special is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than zero. One advantage of using ReLU as an activation function is that since not all neurons are activated on the network, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions. What's more, ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.

At the same time, because the value of ReLU and its derivative equals zero when the input is negative or zero, some neurons may be killed by mistake, so that during back propagation process, the weights and biases for some neurons are not updated. As a result of this fact, the ReLU decreases the model's ability to fit or train from the data properly.

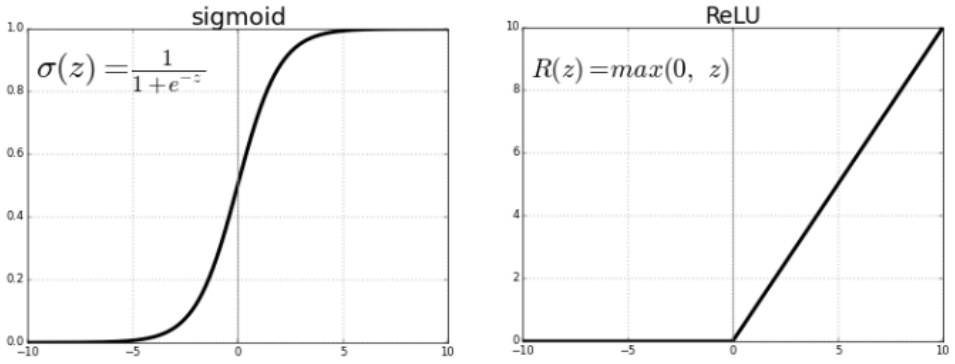

Figure 1.16: ReLU Function and Sigmoid Function. The ReLU function is far more computationally efficient when compared to the sigmoid function.

To deal with this issue, many other kinds of transfer functions variants are developed based on ReLU. For example, the Leaky ReLU function, the Parametric ReLU function, the Exponential Linear Units(ELUs) function, etc.

Another popular activation function used for classification type of NN is Softmax. It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class. It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification. The mathematical equation of Softmax function is:

$$\text{softmax}(Z_i) = \frac{\exp(Z_i)}{\sum_j^1 \exp(Z_i)} \tag{1.26}$$

CHAPTER 2

PRELIMINARY ROAD TEST

This chapter is a description and analysis of a road test using Neural Networks (NN) to assess the ride quality, as a proof of concept. The goal of the thesis is to explore a potentially improved adaptive suspension system. Based on the state of current suspension products, improved technology can make them more intelligent and better able to satisfy customer expectations. Moreover, the use of NN is expected to significantly improve the suspension performance, by improving the efficiency and accuracy of suspension tuning. In the end, the objective is that a real-drive-testing-data trained neural network can switch the suspension mode for the virtual vehicle ride simulation at any time, and show improved passenger comfort.

Before doing the simulation and the study of the ride comfort of off-road vehicles, a real road test was conducted locally in Windsor with micro gyro measurement devices capable of recording acceleration in three directions, and angular velocity around three axes. The car used in this test was a 2018 Jeep Wrangler 4x4 Unlimited Sport, shown in Figure 2.1.

Due to COVID-19, the access to conducting road tests was very limited. It could only be done before the nightly curfew started. Exposure in public areas for a long time was not acceptable during this special time. In this situation, it was decided that the test would

Figure 2.1: A 2018 Jeep Wrangler was used to gather the preliminary road test data.

only be conducted on a relatively straight road with some bumps. The measuring device was mounted at the left rear floor of the car, with the measuring frequency of 100 Hz. The car speed was set at 35 km/h using cruise control, and the total time of data acquisition is 14 seconds, although several hours were spent gathering data to test and refine the data acquisition procedure. The testing road geometry is shown in Figure 2.2. The mounted device is shown in Figure 2.3.

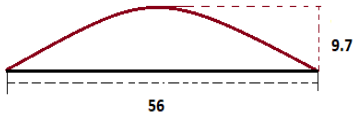

Figure 2.2: Bump geomtry used for the road test.

The resulting data is shown in Figure 2.4. As expected, the vertical acceleration rose to the largest at every bump, followed by the lateral acceleration. However, the wave of the lateral acceleration was bigger than expected. Because the gyroscope was mounted on the rear seats of the floor, the vibrations from seats and the floor can not be ignored. In the meantime, the accuracy of the device was not fully tested. Beside these, the results fit the prediction, as it also can be noticed that the angular pitch motion was larger than either roll or yaw.

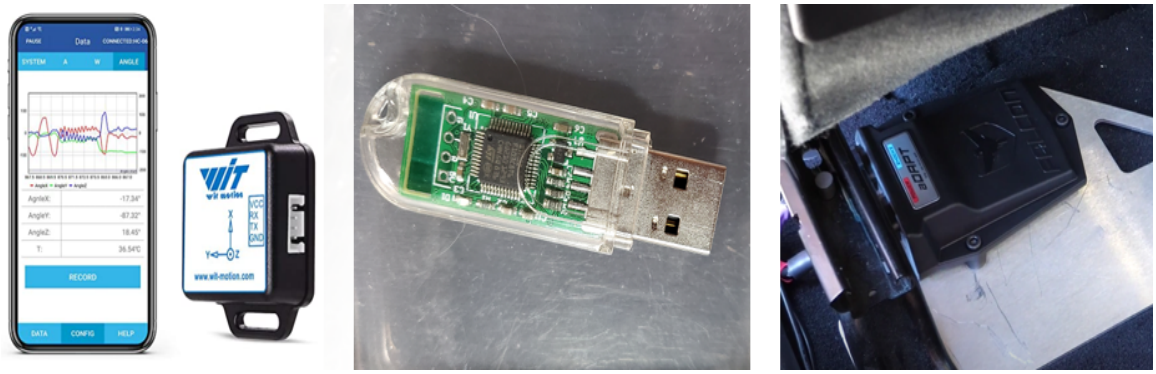

Figure 2.3: First at the left is the gyroscope, in the middle is the USB linking device and at the right is the mounting device

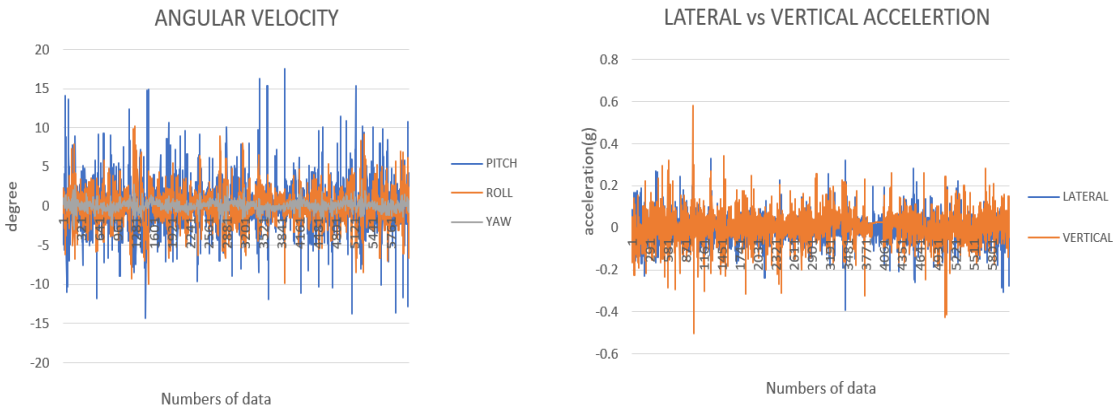

Figure 2.4: Angular velocity and linear acceleration measurements in the vertical, lateral and longitudinal directions. The motions were measured at 100 Hz. The vertical acceleration is the most obvious result, as expected.

2.1 Collecting data and building Neural Networks

Among the data, three groups were picked for Neural Network training. The data was sorted into three categories: 1. 500 points for smooth road condition, 2. 500 points for medium road condition, and 3. 500 for rough road condition. The Neural Network structure is Back Propagation Neural Network with a structure of 5 inputs, 10 neurons and 3 outputs. There are 1000 groups for training and 500 groups of data for testing. The choice of transfer function was the sigmoid function because it is continuously differentiable as

desired in neural networks, and they allow for the back propagation of the error during the training phase, which is necessary for the adjustment of weight and bias to finally achieve convergence. The logic of the calculation, from input layer to hidden layer and finally the output layer, is explained below.

In the following equations, l stands for the number of neurons, m stands for the number of outputs, n stands for the number of inputs, while j presents the neuron in hidden layer.

1. Hidden layer input

$$H_j = f\left(\sum_{i=1}^{n} \omega_{ij} X_i - a_j\right),\ j = 1, 2 \cdots l \tag{2.1}$$

2. Transfer function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

3. Output

$$O_k = \sum_{j=1}^{l} H_j \omega_{jk} - b_k,\ k = 1, 2 \cdots m \tag{2.3}$$

4. Error calculation (Y_k is the expected output)

$$e_k = Y_k - O_k, ;\ k = 1, 2 \cdots m \tag{2.4}$$

5. Weight factor renewal (η = study rate)

$$\omega_{ij} = \omega_{ij} + \eta H_i (1 - H_j) x(i) \sum_{k=1}^{m} \omega_{jk} e_k \tag{2.5}$$

$$\omega_{jk} = \omega_{jk} + \eta H_i e_k \tag{2.6}$$

6. Constant renewal

$$a_j = a_j + \eta H_i(1 - H_j) \sum_{k=1}^{m} \omega_{jk} e_k \tag{2.7}$$

$$b_k = b_k + e_k \tag{2.8}$$

7. Repeating step 2-6 until finished

2.2 Result and proof of concept

The result shows that Neural Network has reasonably good accuracy of detecting each case. In all three modes, the soft mode has an accuracy of 50–80%, the accuracy of the medium mode was always 100%, and the firm mode's accuracy was higher than 90%. To be noticed, the number of input neurons and hidden neurons can be increased. For example, one group of data can be transformed as square root, sine of the data, tangent of the data, while two groups of data can also be multiplied together. The same also applies for the hidden layers. However, an increase in the quantities of the neurons does not ensure an increase in accuracy; it depends on the pattern and distribution of the data. Two results are selected randomly as shown, the straight lines show the errors in one simulation, as 1 stands for smooth mode, 2 stands for medium mode and 3 stands for rough mode.

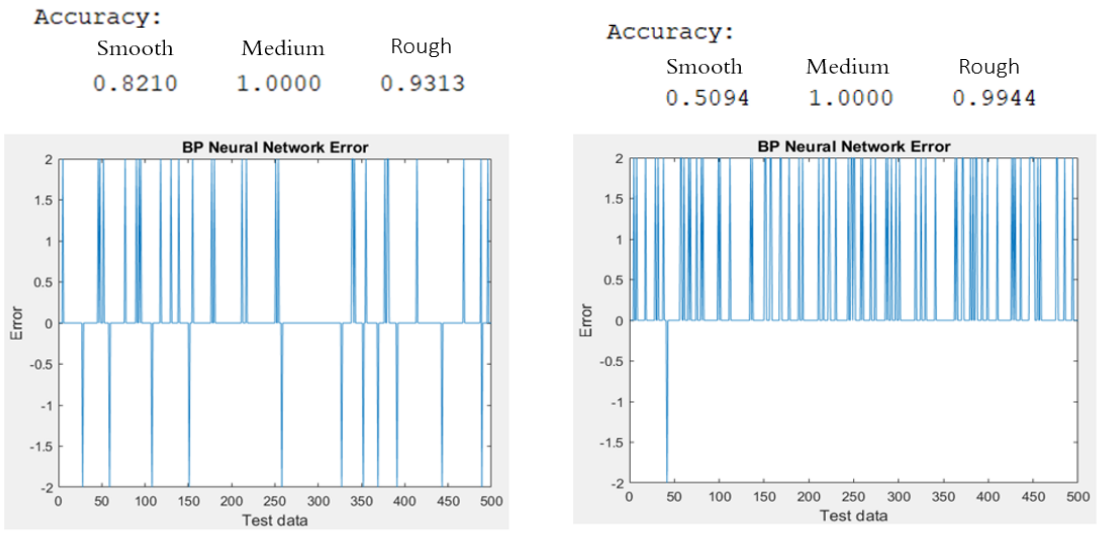

Figure 2.5: There are only errors on smooth and firm mode when the accuracy of medium mode is 1. The plots shows error values for each NN predictions, therefore the value of 2 means misidentifying smooth mode to rough mode, −2 means the opposite.

CHAPTER 3

VEHICLE MODELING

This chapter is a detailed discussion of the process of building a reliable car model as a platform for this study, what extra considerations were involved, and how the issues were resolved. The whole model was built with Julia Language, which is fast, powerful and free. After careful study, a 10 degree-of-freedom car model was built. In this full car model, bounce and roll motions are highlighted. Meanwhile, the forward speed was considered to be constant and the main focus was on vertical and lateral motions. The car model included a suspension system with four springs, two anti-roll bars, and linkages for suspension constraints.

Because the vehicle model was built in Julia and the suspension system training process was completed in MatLab Simulink, the steady state matrices were extracted from the result of simulation in Julia by a MatLab script as the source of data for the Simulink model. The state matrices were the system matrix, the feed-through matrix, and the input and output matrices.

The model was simulated using a random road time history. By comparing the results of before-trained and after-trained vehicle model, the improvement of drive comfort was shown by applying Neural Networks to select the suspension properties.

3.1 Model properties

The whole model figure 3.2 was focus on the chassis. The suspension system was based on a four-link style, so that it had four upper control bars and four lower control bars. In the visualization of the model, they were represented by 8 suspension links. The front axle linked thye two front tires together, similarly with the rear axle. By using two random road functions, one for each side of the car, the front and rear axles could be excited by the road input, thus pitch and roll angles were generated as time series data. Meanwhile, in the consideration of the car length, the front and rear road excitation signals cannot happen simultaneously. In this situation, the road input of the rear axle was delayed by $t = (a + b)/u$, which was the wheelbase of the chassis divided by time.

In Figure 3.2, the center of gravity was put at origin, while four wheels were connected to the axle by four constraints allowing a single rotation representing the wheel bearings. Flexible joints wer used for the front and rear anti-roll bars. Moreover, to monitor the trace of four wheels, flexible joints representing the tire stiffness were also created. However, in the state of constant speed, only vertical and lateral motions were of interest. Therefore, four flexible joints were set to represent the vertical movement, with the lateral stiffness presented by another four flexible joints. As shown in Figure 3.1, the structure of a typical double wishbone suspension system includes an upper control arm, an lower control arm and a shock absorber that is mounted in between them. This brings constraints to the system modeled in this thesis. The upper suspension links were set to be moved with the top of the shock absorbers, when the lower suspension links would move together with the bottom of the shock absorbers. This means, the shock absorbers had exactly the same motions as there suspension links. Meanwhile, the sway bar should be mounted on the lower control arms such that in the model there were two rigid points located at the center of the front and rear axle. In this way it accomplish the goal of matching the mechanism in real world.

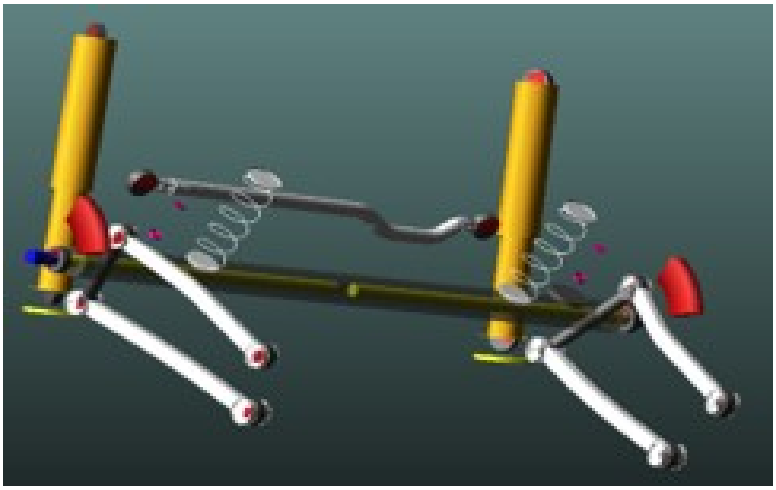

Figure 3.1: An example of the front suspension system of Jeep Wrangler in Adams

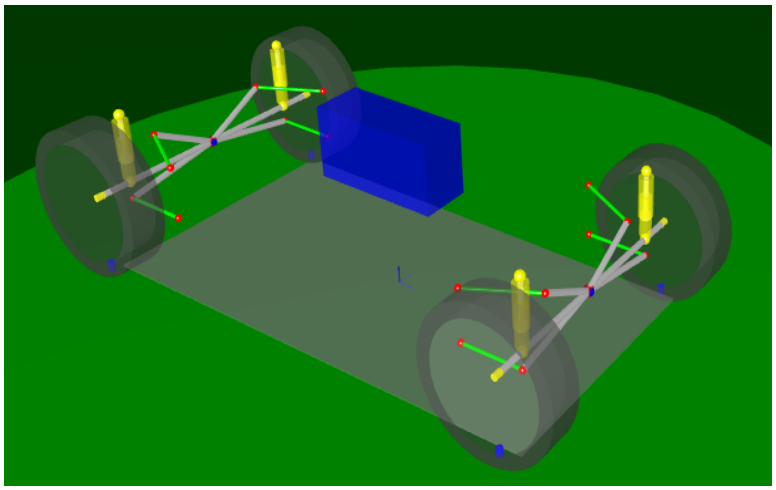

Figure 3.2: A full car model with 10 degrees of freedom was built using the EoM vehicle simulation software[4]

3.2 Software tools

3.2.1 EoM package

To handle the kinematic system with multiple parts and mechanisms like a car, it is nearly impossible to prepare the equations of motion and resulting system matrices by hand. Due to the increasing of complexity of the models, the matrices become excessively

Table 3.1: Parameters of the full car model

Parameter	Notation	Value
Wheelbase dimension	a	1.384 m
Wheelbase dimension	b	1.561 m
Sprung mass	m	2000 kg
Front unsprung mass	m_{uf}	80 kg
Rear unsprung mass	m_{ur}	80 kg
Pitch inertia	I_{y}	3200 kg m^2
Yaw inertia	I_{z}	3200 kg m^2
Roll inertia	I_{x}	800 kg m^2
Front suspension stiffness	k_{f}	25000 N/m
Rear suspension stiffness	k_{r}	25000 N/m
Front suspension damping	c_{f}	1000 N/m
Rear suspension damping	c_{r}	1200 N/m
Front car width	t_{f}	1.572 m
Rear car width	t_{r}	1.572 m
Tire stiffness	k_t	210000 N/m
Tire radius	r	0.419 m
Forward speed	u	10 m/s

large. The model of a vehicle can be treated as lots of parts or bodies being connected by various joints that mimic the real-life constraints. To help resolve this issue, the EoM software package is a powerful tool that can automatically generate the equations of motion, presented as matrices in the state space form, in order to predict the states of the system. When many different rigid bodies move together under certain constraints for different mechanisms, such a system is described as a 'multibody dynamic system'. Generally, the equations of motion of this kind of system are a set of coupled nonlinear differential algebraic equations. In EoM, systems are formed from three parts: 1. the kinematic differential equations, 2. the Newton-Euler equations, 3. the constraint equations.

The coordinates

The first task to complete the analysis is to set the coordinates. It is typical for a vehicle to have six coordinates to define the motion of the system. In a full analysis, there are three coordinates to determine lateral, longitudinal and vertical movement. Apart from these, the other three coordinates are for bounce, pitch and roll motions. To write the position and orientation coordinates, one can use $\boldsymbol{p}$, which is a $6n \times 1$ combined vector:

$$\boldsymbol{p} = \begin{bmatrix} \boldsymbol{x}_1' & \boldsymbol{\theta}_1' & \boldsymbol{x}_2' & \boldsymbol{\theta}_2' & \cdots & \boldsymbol{x}_n' & \boldsymbol{\theta}_n' \end{bmatrix}' \tag{3.1}$$

in which n represents the number of rigid bodies. Moreover, the velocities, as well as the angular velocities, are similarly represented as a combined vector term $\boldsymbol{w}$.

Newton-Euler equations

The most important step is to write the Newton-Euler equations for each body. According to Newton's second law, the forces acting of the system can be described as:

$$\mathbf{M}\dot{\boldsymbol{w}} = \sum f(\boldsymbol{p}, \boldsymbol{w}, t) \tag{3.2}$$

where the matrix of mass is a combination of masses and moments of inertia of the system:

$$\mathbf{M} = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 & 0 & \\ 0 & m_1 & 0 & 0 & 0 & 0 & \\ 0 & 0 & m_1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & I_{\mathrm{xx}_1} & -I_{\mathrm{xy}_1} & -I_{\mathrm{xz}_1} & \cdots \\ 0 & 0 & 0 & -I_{\mathrm{xy}_1} & I_{\mathrm{yy}_1} & -I_{\mathrm{yz}_1} & \\ 0 & 0 & 0 & -I_{\mathrm{xz}_1} & -I_{\mathrm{yz}_1} & I_{\mathrm{zz}_1} & \\ & & & \vdots & & & \ddots \end{bmatrix} \tag{3.3}$$

There are various forces applied on the system, like the inertia force $\boldsymbol{f}_{\mathrm{i}}$, elastic force $\boldsymbol{f}_{\mathrm{e}}$, constraint force $\boldsymbol{f}_{\mathrm{c}}$, and applied force $\boldsymbol{f}_{\mathrm{a}}$. Therefore the equation of motion can be expressed as:

$$\mathbf{M}\dot{\boldsymbol{w}} = \sum \boldsymbol{f}_{\mathrm{i}} + \sum \boldsymbol{f}_{\mathrm{e}} + \sum \boldsymbol{f}_{\mathrm{c}} + \sum \boldsymbol{f}_{\mathrm{a}} \tag{3.4}$$

After linearizing the Newton-Euler equations, the elastic forces and damping forces result in the stiffness (**K**) and damping (**L**) matrices. Notice that the **L** matrix also includes the inertial force terms.

$$\begin{bmatrix} \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{Bmatrix} \delta\dot{\boldsymbol{p}} \\ \delta\dot{\boldsymbol{w}} \end{Bmatrix} + \begin{bmatrix} \mathbf{K} & \mathbf{L} \end{bmatrix} \begin{Bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{w} \end{Bmatrix} = \delta\boldsymbol{f}_{\mathrm{c}} + \delta\boldsymbol{f}_{\mathrm{a}} \tag{3.5}$$

Constraints of a multibody system

A both challenging and important part of the multibody dynamics equations is to conclude the constraint forces. The term 'multibody' means a system that is composed of at least two bodies. All the bodies have their own motions but the constraint matrix links them together by their own mechanisms and also defines the degrees of freedom of the system. When analyzing the general motion of a rigid body, there are six degrees of freedom; similarly in a multibody system, each body also has six degrees of freedom: the displacements on x, y, z axis and the rotations around each axis. However the situation becomes more complex when the bodies are connected. Depending on how many bodies there are in the system, the more the number of constraints, the less the number of degrees of freedom. Assuming that there are n bodies in the system and the number of constraints from the connections of the bodies is m, the final number of degrees of freedom will be $6n - m$.

By using the Jacobian matrix of the constraint equations, the equations are written with both holonomic and nonholonomic constraints included. The $\mathbf{J}_{\mathrm{h}}$ matrix is for holonomic constraints while the $\mathbf{J}_{\mathrm{nh}}$ matrix stands for nonholonomic constraints. The $\mathbf{V}$ is a consequence of the kinematic differential equations.

$$\begin{bmatrix} \mathbf{J}_{\mathrm{h}} & \mathbf{0} \\ \mathbf{J}_{\mathrm{h}}\mathbf{V} & \mathbf{J}_{\mathrm{h}} \\ 0 & \mathbf{J}_{\mathrm{nh}} \end{bmatrix} \begin{bmatrix} \delta\dot{\boldsymbol{p}} & \delta\boldsymbol{p} \\ \delta\dot{\boldsymbol{w}} & \delta\boldsymbol{w} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{3.6}$$

The task of the constraint Jacobian $\boldsymbol{J}$ is to minimize the size of the matrix for equations of motions, which is completed by the orthogonal matrix $\mathbf{J}$. Then the next step is to find a $\mathbf{T}$ matrix, such that:

$$\mathbf{JT} = \mathbf{0} \tag{3.7}$$

By multiplying $\mathbf{T}$ with the new coordinates vector $\boldsymbol{x}$:

$$\mathbf{T}\boldsymbol{x} = \begin{Bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{w} \end{Bmatrix} \tag{3.8}$$

And:

$$\mathbf{J}\mathbf{T}\boldsymbol{x} = \mathbf{J}\begin{Bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{w} \end{Bmatrix} = \mathbf{0} \tag{3.9}$$

Meanwhile, add a matrix $\mathbf{U}$ to satisfy:

$$\mathbf{U}\begin{Bmatrix} \mathbf{0} \\ f_{\mathrm{c}} \end{Bmatrix} = \mathbf{0} \tag{3.10}$$

Note that in most cases, the matrix $\mathbf{U} = \mathbf{T}'$. While minimizing the number of coordinates, the equation is also reduced linear first order form:

$$\mathbf{U}\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix}\mathbf{T}\dot{\mathbf{x}} + \mathbf{U}\begin{bmatrix} \mathbf{V} & -\mathbf{I} \\ \mathbf{K} & \mathbf{L} \end{bmatrix}\mathbf{T}\mathbf{x} = \mathbf{U}\begin{Bmatrix} \mathbf{0} \\ f_{\mathrm{a}} \end{Bmatrix} \tag{3.11}$$

Then:

$$\mathbf{A} = -\mathbf{U}\begin{bmatrix} \mathbf{V} & -\mathbf{I} \\ \mathbf{K} & \mathbf{L} \end{bmatrix}\mathbf{T} \tag{3.12}$$

$$\mathbf{B}\boldsymbol{u} = \mathbf{U}\begin{Bmatrix} \mathbf{0} \\ f_{\mathrm{a}} \end{Bmatrix} \tag{3.13}$$

$$\mathbf{E} = \mathbf{U}\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix}\mathbf{T} \tag{3.14}$$

Now, after substituting all the terms, the state-space describing matrices are found:

$$\begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{Bmatrix} \dot{\boldsymbol{x}} \\ \boldsymbol{y} \end{Bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{Bmatrix} \tag{3.15}$$

in which the input is $\boldsymbol{u}$ and the output is $\boldsymbol{y}$. As $\boldsymbol{x}$ is defined as the state vector, **A** is the system matrix; **B** is the input matrix; **C** is the output matrix; **D** is the feedthrough matrix.

In some cases, the equation of motion may contain rate sensitive inputs. Consider for example, a spring mass damper with base excitation:

$$m\ddot{x} + c\dot{x} + kx = fu + g\dot{u} \tag{3.16}$$

The EoM software can accommodate such systems. In the form of a matrix, f and g coefficients become the **F** and **G** matrices, and the equations of motion (Equation 3.11) can be modified to accommodate the extra terms. The reduction process follows the same approach.

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} & -\mathbf{G} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{w}} \\ \dot{\boldsymbol{u}} \end{Bmatrix} + \begin{bmatrix} \mathbf{V} & -\mathbf{I} & \mathbf{0} \\ \mathbf{K} & \mathbf{L} & -\mathbf{F} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{p} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{Bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \end{bmatrix} \left\{ \boldsymbol{u} \right\} \tag{3.17}$$

A user-defined package

Based on of all the theory above, the EoM multibody dynamics software was developed by University of Windsor Vehicle Dynamics and Control Research Group lead by Dr. Minaker. This package automatically generates linearized equations by fully applying the mathematical method above. The package was originally prepared in the Matlab environment, but developmnet has now converted to the Julia programming language. It is completely free to access this package, and it is available to download from the online source code repository GitHub (www.github.com).

When starting to use this package, it is the user's task to define the parameters, the inputs and outputs, and the structure of the models to run. Notice that certain different types of 'items' are defined in EoM as the building blocks of a model. As mentioned earlier, Table 3.2 lists and describes all types of items that are available in EoM.

Table 3.2: Types of items in EoM

Type of item	Definition
body	a rigid body
spring	a two point elastic spring, with linear or torsional stiffness and damping, non-zero free length
link	a two point massless rigid link
rigid_point	a generic point constraint with a variable number of constraint forces and moments
flex_point	a point spring with translational and/or rotational stiffness and damping
nh_point	a non-holonomic constraint to prevent velocity but not displacement
beam	a zero mass beam spring with bi-directional bending and shear stiffness
load	constant forces or moments applied to the system
actuator	applied force or moment, proportional to an input signal
sensor	used to measure displacement, velocity, or acceleration

After the parameters are defined, EoM will automatically proceed to generate and solve the equations of motion. EoM finds the eigenvalues and natural frequencies of different types of motion, by applying the equation:

$$\det[\mathbf{E}s - \mathbf{A}] = \mathbf{0} \tag{3.18}$$

An eigenvalue contains two parts, $s = a \pm ib$. The real part a represents decay of motion, while the imaginary part b of the roots shows the frequency. The more negative the real parts are, the more stable the system is, and it will decay approaching to zero with time. If it is not the case, the system will become unstable by the increase of the motion generated in the system. Meanwhile, the imaginary part shows the frequency of oscillation. For example, when imaginary part is zero, there is no oscillation in the system.

The output file of EoM is a system report in various formats (pdf, html) including the state space form of the equations. Each moment of the system has corresponding eigenvalues and frequencies to describe the behavior of the system, Figure 3.3 is a sample table generated by the vehicle model for this research:

Eigenvalue	x	y	z	u_x	u_y	u_z
-6.192766 + 1.479243im	-0.15791 + 1.565645im	-0.0	2.45193 - 0.666359im	0.820407 + 0.222961im	-0.0	0.052836 + 0.523859im
-6.192766 - 1.479243im	-0.15791 - 1.565645im	-0.0	2.45193 + 0.666359im	0.820407 - 0.222961im	-0.0	0.052836 - 0.523859im
-0.827485 + 6.603941im	-4.813641 - 1.24087im	-0.0	-0.065013 - 0.000605im	0.0	-0.968343 + 0.249622im	0.0
-0.827485 - 6.603941im	-4.813641 + 1.24087im	-0.0	-0.065013 + 0.000605im	0.0	-0.968343 - 0.249622im	0.0
-0.950066 + 7.173178im	0.000382 + 0.000687im	-0.0	-0.102828 - 0.113962im	0.999987	-0.0	0.004989 - 0.00115im
-0.950066 - 7.173178im	0.000382 - 0.000687im	-0.0	-0.102828 + 0.113962im	0.999987	-0.0	0.004989 + 0.00115im
-1.248461 + 7.567884im	0.323328 - 0.093145im	0.0	-0.065042 - 0.000697im	-0.0	1.0	-0.0
-1.248461 - 7.567884im	0.323328 + 0.093145im	0.0	-0.065042 + 0.000697im	-0.0	1.0	-0.0
-7.796449 + 53.338188im	0.979681 + 0.038908im	0.0	-0.063796 - 0.004647im	-0.0	1.0	-0.0
-7.796449 - 53.338188im	0.979681 - 0.038908im	0.0	-0.063796 + 0.004647im	-0.0	1.0	-0.0
-6.468193 + 53.654379im	-1.254574 + 0.056406im	0.0	-0.063695 - 0.004586im	-0.0	-0.998991 - 0.044915im	0.0
-6.468193 - 53.654379im	-1.254574 - 0.056406im	0.0	-0.063695 + 0.004586im	-0.0	-0.998991 + 0.044915im	0.0
-18.211267 + 56.732994im	-0.676337 + 0.346047im	0.0	0.211733 + 0.41347im	0.237772 - 0.464319im	-0.0	0.759512 + 0.388604im
-18.211267 - 56.732994im	-0.676337 - 0.346047im	0.0	0.211733 - 0.41347im	0.237772 + 0.464319im	-0.0	0.759512 - 0.388604im
-19.173416 + 56.480725im	0.770952 + 0.234036im	0.0	0.281684 + 0.815649im	0.238598 - 0.690888im	-0.0	-0.653028 + 0.198238im
-19.173416 - 56.480725im	0.770952 - 0.234036im	0.0	0.281684 - 0.815649im	0.238598 + 0.690888im	-0.0	-0.653028 - 0.198238im
-0.0	3.733393135575834e9	-1.777832653702e6	1.723482889383e6	0.000462	0.001519	-0.999997

Figure 3.3: An output table generated by EoM

3.2.2 Julia

The model was solved by EoM in Julia language. The name of the input file is `input_jeep.jl`; the file contents are listed in Appendix A. This input file listed all the dimensions and locations of the bodies; rigid points; flex points; suspension links and shock absorbers. To integrate the model with the Simulink environment, eight actuators were used for the input with twelve sensors measuring the outputs. After running the simulation, the related tools in the EoM package will automatically generate and save all of the outputs in a folder. For example, all the eigenvalues, the plots of the displacements and frequencies, as well as the 3D animations for each mode shape of the simulation.

Input data

As mentioned in 3.2, EoM is a user-defined package. This research used the package both in Julia and Matlab, while the first step was to carefully define the inputs so that the model was suitable. In Table 3.3, all the geometry of the model is listed.

After the simulation was completed, there was another Julia script to save the state space matrices **A**, **B**, **C**, **D** , **E**, and send them as inputs to the Matlab script that was the next step. Therefore, in the next step the processed data from Matlab script would be feed into Simulink control loops, where the Neural Networks were inserted. See Figure 3.4.

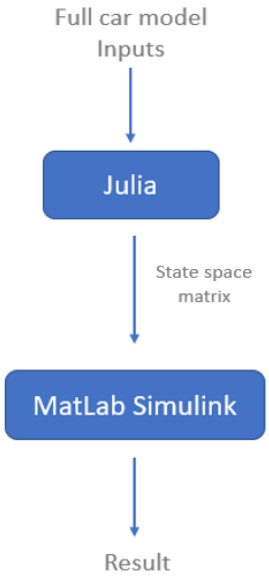

Figure 3.4: Simulation progress flowchart

Table 3.3: The geometry of the full car model

Parameter	Coordinates
Chassis	$[0, 0, 0.8]$
front axle	$[a, 0, r]$
rear axle	$[-b, 0, r]$
Left front wheel and hub	$[a, t_f/2, r]$
Rear front wheel and hub	$[a, -t_f/2, r]$
Left Rear wheel and hub	$[-b, t_r/2, r]$
Right rear wheel and hub	$[-b, -t_r/2, r]$
Front anti-roll	$[a, 0, r]$
Rear anti-roll	$[-b, 0, r]$
Wheel bearing Left front	$[a, t_f/2, r]$
Wheel bearing Right front	$[a, -t_f/2, r]$
Wheel bearing Left rear	$[-b, t_r/2, r]$
Wheel bearing right rear	$[-b, -t_r/2, r]$
Left front tire, vertical	$[a, t_f/2, 0]$
Right front tire, vertical	$[a, -t_f/2, 0]$
Left rear tire, vertical	$[-b, t_r/2, 0]$
Right rear tire, vertical	$[-b, t_r/2, 0]$
Left front tire, horizontal	$[[a, t_f/2, 0]]$
Right front tire, horizontal	$[a, -t_f/2, 0]$
Left front tire, horizontal	$[-b, t_r/2, 0]$
Right rear tire, horizontal	$[-b, -t_r/2, 0]$
Suspension link 1	$[a - 0.4, t_f/2 - 0.2, r - 0.1]$
Suspension link 2	$[a - 0.4, -(t_f/2 - 0.2), r - 0.1]$
Suspension link 3	$[a - 0.4, t_f/2 - 0.2, r + 0.2]$
Suspension link 4	$[a - 0.4, -(t_f/2 - 0.2), r + 0.2]$
Suspension link 5	$[-b + 0.4, t_r/2 - 0.2, r - 0.1]$
Suspension link 6	$[-b + 0.4, -t_r/2 - 0.2, r - 0.1]$
Suspension link 7	$[-b + 0.4, t_r/2 - 0.2, r + 0.2]$
Suspension link 8	$[-b + 0.4, -(t_r/2 - 0.2), r + 0.2]$
Left front spring	$[a, t_f/2 - 0.2, r]$
Right front spring	$[a, -(t_f/2 - 0.2), r]$
Left rear spring	$[-b, t_r/2 - 0.2, r]$
Right rear spring	$[-b, -(t_r/2 - 0.2), r]$

CHAPTER 4

MATLAB SIMULINK

This chapter is to describe the process of transferring result data from simulation in Julia to MatLab as inputs, when an entire control system was built with two trained Neural Networks. To run such a vehicle model in MatLab, many auxiliary functions were needed. All the details are given in the following sections and subsections.

4.1 MatLab script

4.1.1 Random road

The first task was to build road profiles that the vehicle could run over to simluate a random road. To observe the performance on roads with different roughness, such road profile generation function should be able to provide various conditions of road according to a certain criteria. Meanwhile, the function changes the road profile in every single simulations in order to satisfying the needs of data collected.

The random road is a function included in the EoM package and used in conjunction with the quarter car model or half car model to demonstrate the response from the suspension. According to ISO 8608 [15], there are eight ranges of roughness of the road, from the smooth (A) to roughest (H), where ranges F, G, and H are considered equivalent

to an off-road condition. In this model, the road roughness, an integer ranging from 3-9 was defined in the script. A class 3 road is very smooth (on the boundary of ISO 8608 classes A and B), where class 9 is extremely rough (boundary of ISO 8608 classes G and H). In this research, road roughness of 3–6 (from D to E) were used for simulations and data collections. For the random road function, the standard of roughness is based on the power spectral density (PSD) plots4.1 represented by an unevenness index G_d, which could be expressed as function of two different types of spatial frequencies Ω_0 and n_0. The corresponding standard index of G_d for different level road roughness is defined while Ω_0 = 1.0 rad/m or n_0 = 0.1 cycles/m and are listed in 4.1.

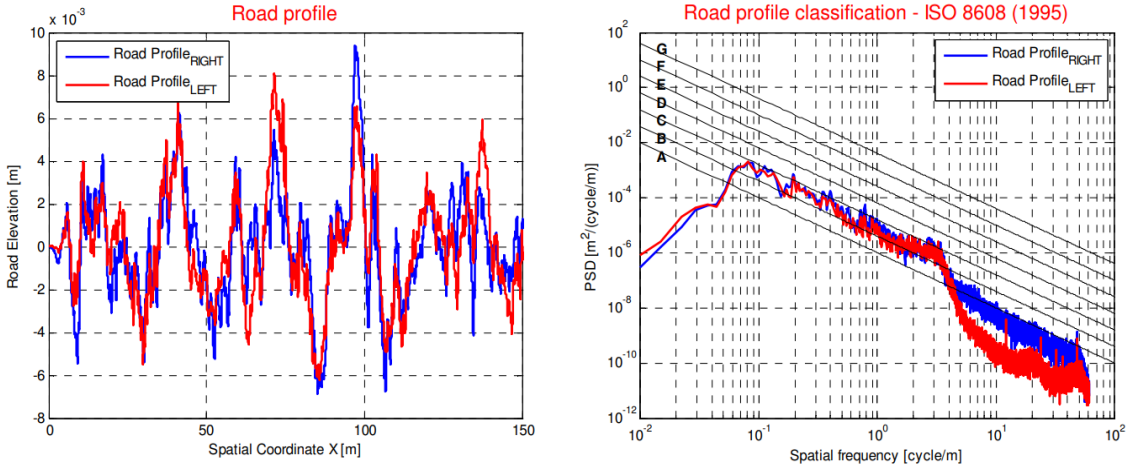

Figure 4.1: An example of road roughness Class 4

Table 4.1: ISO 8608 values of $G_d(n_0)$ and $G_d(\Omega_0)$ [15]

Road class	$G_d(n_0)(10^{-6}m^3)$		$G_d(\Omega_0)(10^{-6}m^3)$	
	Lower limit	Upper limit	Lower limit	Upper limit
A	-	32	-	2
B	32	128	2	8
C	128	512	8	32
D	512	2048	32	128
E	2048	8192	128	512
F	8192	32768	512	2048
G	32768	131072	2048	8192
H	131072	-	8192	-

The index G_d can be expressed as a function of frequency as:

$$G_d(n) = G_d(n_0) \cdot \left(\frac{n}{n_0}\right)^{-2} \tag{4.1}$$

or:

$$G_d(\Omega) = G_d(\Omega_0) \cdot \left(\frac{\Omega}{\Omega_0}\right)^{-2} \tag{4.2}$$

Using the concept of power spectral density, a random road of equivalent roughness can be generated as a sum of a series of sine waves, where the amplitude and frequency vary according to a specific relationship, and the phase angle is random. The relationship between G_d and the amplitude of each wave A_i is:

$$G_d(n_i) = \frac{A_i^2}{2\Delta n} \tag{4.3}$$

Therefore, the amplitude can be expressed as:

$$A_i = \sqrt{2\Delta n \cdot G_d(n_0) \cdot \left(\frac{n}{n_0}\right)^{-2}} \tag{4.4}$$

Since the actual height of the random road h is function of amplitude A and expression is as shown:

$$h(x) = \sum_{i=0}^{N} A_i \cos(2\pi \cdot i \cdot \Delta n \cdot x + \phi_i) \tag{4.5}$$

expanded as:

$$h(x) = \sum_{i=0}^{N} \sqrt{\Delta n} \cdot 2^k \cdot 10^{-3} \cdot \left(\frac{n_0}{i \cdot \Delta n}\right) \cdot \cos(2\pi \cdot i \cdot \Delta n \cdot x + \phi_i) \tag{4.6}$$

where x represents the location along the road from 0 to L, the length. The spatial frequency interval $\Delta n = 1/L$, and the number of frequencies will be expressed as $N = L/B$. The integer value k is the index that shows the level of the road roughness. It ranges from 3 to 9, corresponding to the class A-B transition to class G-H transition, as shown in

Table 4.2. The random phase angle ϕ for each frequency is from 0 to 2π.

Table 4.2: k values for ISO road roughness classification

Road Class		k
Upper limit	Lower limit	
A	B	3
B	C	4
C	D	5
D	E	6
E	F	7
F	G	8
G	H	9

The random road function returns a function handle that gives back 'z' as a function of 'x'. However, in the model, x needed to be converted to time index, where x=ut.

$$x = ut \tag{4.7}$$

A forward speed of $u = 10$ m/s is assumed. In order to capture the random road definition accurately, at least two time samples per wavelength were needed. At a forward speed of 10 m/s, the shortest wavelength was covered in 0.05/10 = 0.005 seconds. This gives a time step of 0.005/2 = 0.0025 s. This turns out to be the limiting factor, as 0.0025 s was very short compared to the timescale of vehicle response, which gave lots of time steps per time constant and/or wavelength of the vehicle model.

$$t = 0 : 0.0025 : 60 \tag{4.8}$$

To compute the resulting lag and from the wheelbase between the front and rear axle:

$$\text{lag} = \frac{2.946}{10} = 0.2946 \text{ s} \tag{4.9}$$

The random road function in EoM incorprates the ability to generate two 'similar' random roads, such the the left and right side of the vehicle do not receive exactly the

same input. The difference is a tunable dimensionless parameter ranging from 0–1, where a choice of 0 gives two identical roads, and a choice of 1 gives two completely different roads.

4.2 Simulink control loop

This section explains the methodology of building the control loop in MatLab Simulink. The final goal was to compare the suspension performances before and after the training. The simulation block diagram starts with the creation of a random road profile at the very left edge. The function transferred different road input data to both of the front wheels. Meanwhile, there were two Time-Delay function blocks connected to left rear wheel and right rear wheel separately (see Figure 4.2). This allows the vehicle to pass over a profile so that the wheels at rear received identical road input after a short time delay, calculated by the length of wheelbase divided by the constant speed of the vehicle. In this research, the vehicle model was running at a constant speed of 10 km/h. During all the of the simulation, the road input was consistent at a certain roughness level, according to the road class chosen before simulation started. Based on Newton's equations, the force input from road profile went into a system processing block where the steady state matrices were stored. Therefore, the right end of the state space block was the output of the vehicle system, supplying the data of twelve outputs. Eight of the outputs were the displacements of the springs and dampers for left front, left rear, right front and right rear suspensions. The remaining system outputs included the pitch and roll velocities of the chassis, together with the acceleration in the vertical and lateral directions of a point fixed on the chassis.

As described in Chapter 2, ride quality assessment relied on the data output of the gyroscope mounted on the seat rail. Because the gyroscope couldn't give the displacements of either springs or dampers, but the velocity and acceleration in longitudinal, lateral and vertical directions, the training data of neural networks in next step were the four outputs from the vehicle system: pitch velocity, roll velocity, vertical acceleration, and lateral acceleration. These specific parameters would be train by the first neural network to identify which road class was the input road profile. Afterwards, the road grade would be another

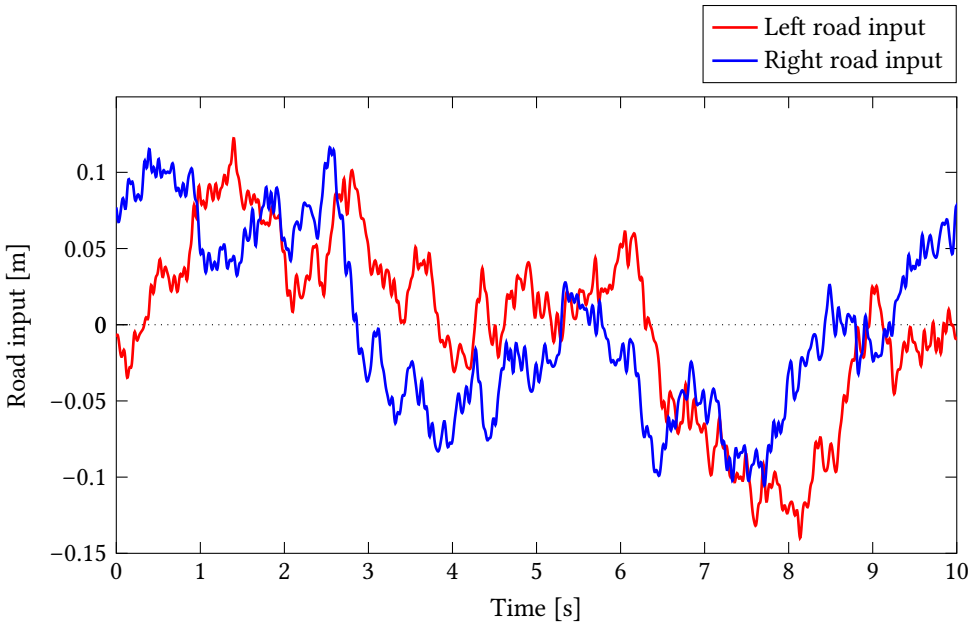

Figure 4.2: Random road grade 6. This was a more agressive road profile used in the simulation.

input together with the outputs of velocities and accelerations, which in total was 5 inputs, entering the second neural network. The function of the second neural network was to finally determine how much the extra suspension force would be added to the original suspension system to achieve a better ride quality. See Figure 4.3

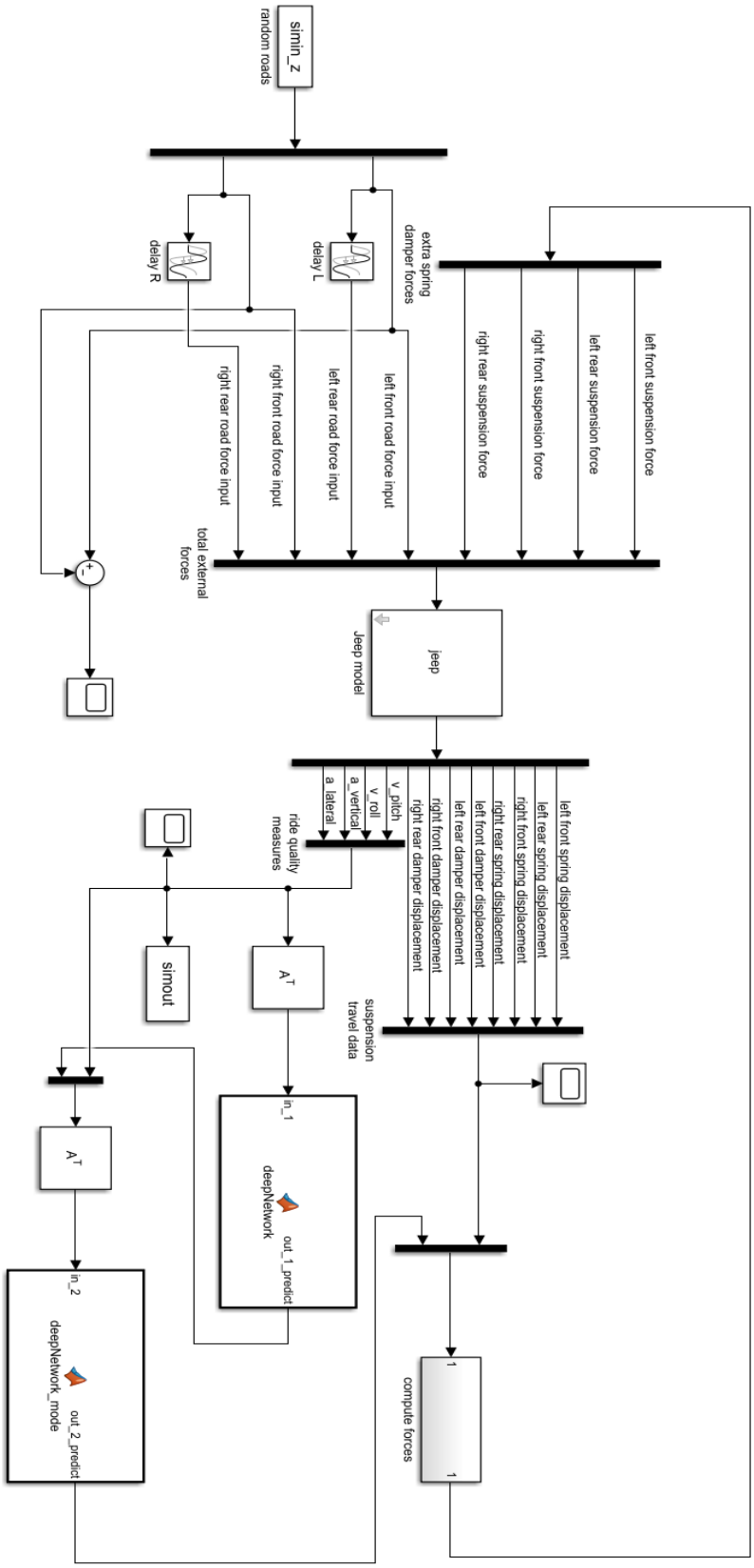

Figure 4.3: Simulink control loop

CHAPTER 5
NEURAL NETWORKS

5.1 Neural Networks

There were two neural networks included in the simulation loop. The aim of the first neural network was to recognize which road grade the vehicle was experiencing. When driving in the real world, switches between road grades happen depending on different routes. In this case, being able to tell the vehicle system which road classes the vehicle is driving on is the goal of the first neural network. In preparation to train the first neural network, 4000 sets of data were collected from designated simulations. For each road grade, 1000 sets of data were gathered; the road grades were class 3, class 4, class 5, and class 6. The first NN had four inputs and one output, the inputs were pitch velocity, roll velocity, vertical acceleration and lateral acceleration, while the output was the road class(3, 4, 5, 6). The second neural network take took five inputs and gave one output. The five inputs were the four inputs, the one output of the first neural network. Finally, the second neural network gave a number of 0, 0.5 or 1. By multiplying the output number with the extra stiffness and damping, the additional forces were added to the suspension system.

5.1.1 Road grade identification

As mentioned above, the first neural network was to identify the grade of road roughness. This type of neural network is aiming to do the classification. Because only grade 3 to grade 6 was used to collect the data, there were four possible numbers that the output could give. When creating the database for the neural network, as per category was classification, the very first thing to do was to split the data. In this database, columns 1–4 were the time history data of pitch velocity, roll velocity, vertical acceleration and lateral acceleration. Column 5 was the road grade, it should be specified as 'categorial' data in order to clarify the task. Briefly speaking, the first step was marking columns 1–4 as 'feature data' and column 5 as the category data. Afterwards, in this research, 50% of the total data became training data when the other 50% was further split with half (i.e., 25%)

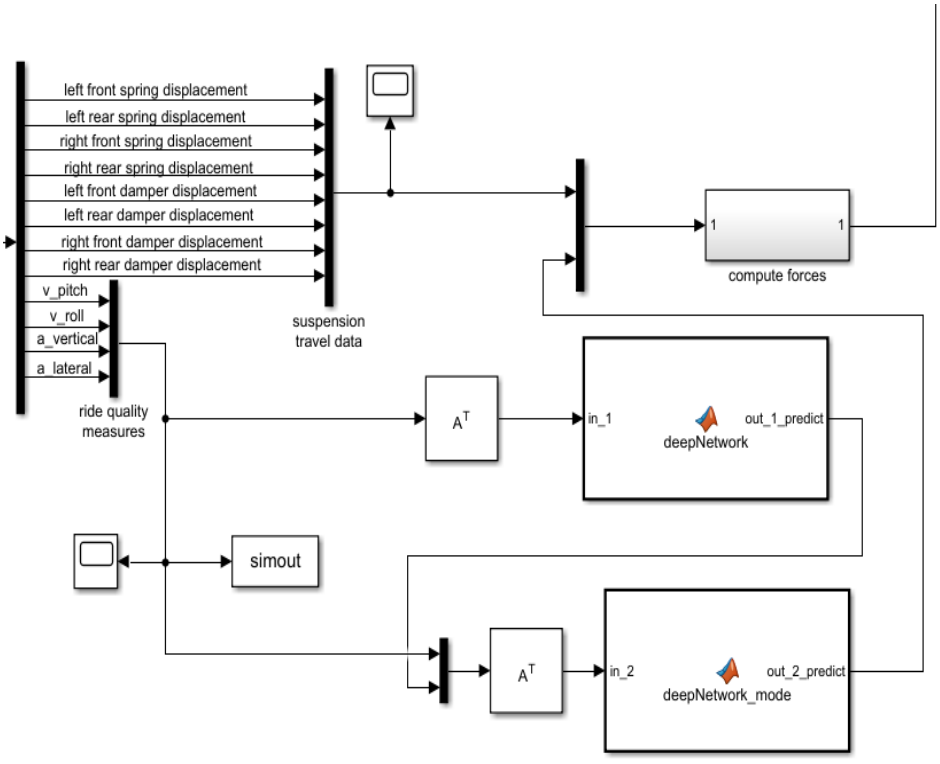

FIGURE 5.1: The upper block of deep network is the first neural network, which provided an additional feature into the lower block-the second neural network to choose the suspension mode and submit it into the 'compute forces' block

used for testing and the other half for validation. Meanwhile, there were also other training option parameters to define for the network: the Batchsize was 50 and the number of epochs was 200. The equations are:

$$\frac{\text{number of training samples}}{\text{BatchSize}} = \text{iterations per epoch} \tag{5.1}$$

which in this case was:

$$\frac{1000}{50} = 20 \tag{5.2}$$

$$\text{iterations per epoch} \times \text{Epochs} = \text{iterations} \tag{5.3}$$

which in this case was:

$$20 \times 200 = 4000 \tag{5.4}$$

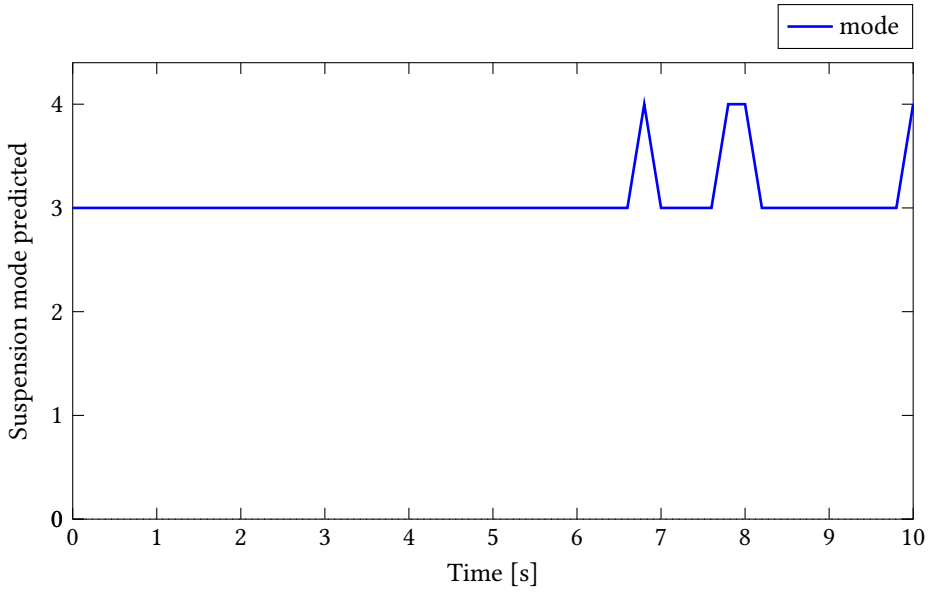

Figure 5.2: Mode identification for Random road grade 3

The next step was to define the structure of neural network. All the input values have to be normalized before entering neural network. In the road grade identification NN, in addition of the input layer and the output layer, five layers were defined in total. Since

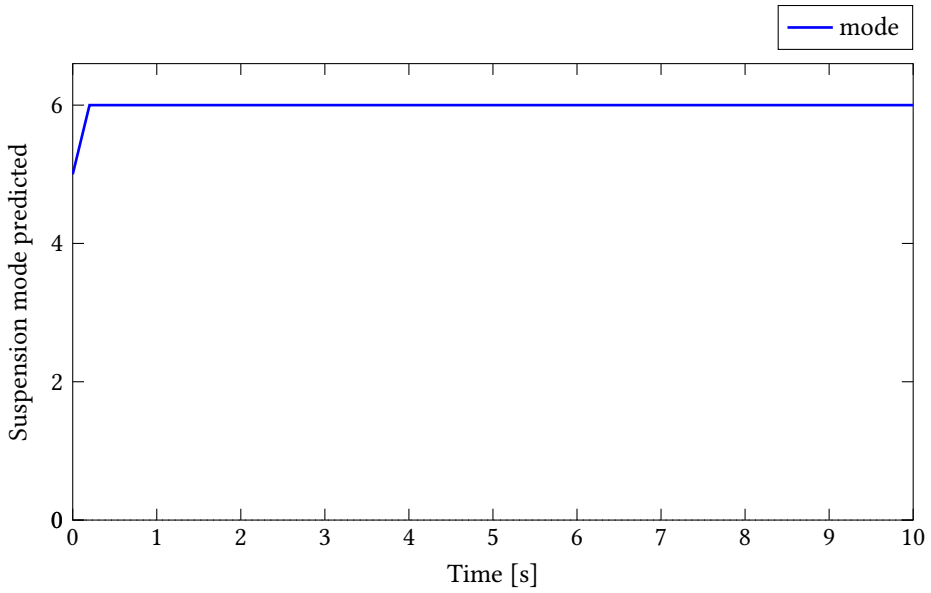

Figure 5.3: Mode identification for Random road grade 6

it was a classification neural network, the activation functions for the five layers in the middle were all ReLU function. Meanwhile, to convert the output to road grade number, the Softmax function was used in the output layer, also called the 'classification' layer. After the training process was finished, the accuracy of this neural network was around 90%, and the whole training progress was visible, as seen in Figure 5.4

5.1.2 Selection of the suspension mode

The second neural network had five inputs: the four inputs and the one output of the first neural network. Finally, the second neural network gave a number either 0, 0.5 or 1. By multiplying the output number with the extra stiffness and damping, the additional forces were added to the suspension system. The output from the first neural network was directly fed to the second neural network together with the former outputs from the vehicle system model. Therefore, the number of features in the second neural network was five, i.e., it was the same as the first one, also a classification neural network. In the database, columns 1–5 were the pitch velocity, roll velocity, vertical acceleration, lateral acceleration and the road grade, while column 6 was the classification data 0, 0.5and1. For these three modes, mode 0 was for road grade 3, which means no additional spring and

damping force would be added. Mode 0.5 was switched on when the road grade was 4 or 5, where half of the additional spring and damping force were added, and mode 1 was chosen at road grade 6 when all the extra forces were added.

Unlike the first neural network, in the second one, 75% of the data became training data. Since the number of inputs increased, the percentage of data used for training purpose raised as well to achieve a decent percentage of accuracy. The rest of the data were also half used for test and the other half for validation. When defining training option parameters, the batch size was the same but number of epochs increased from 200 to 300. The theory behind adding numbers of epochs was similar to raising of the percentage of training data. The more epochs that are present, the longer the simulation will run, and hopefully the higher the accuracy that the neural network can achieve. At the same time, the structure of the network was the same as the first one, where the activation functions for the five layers in the middle were all the ReLU function and the Softmax function was used in the output layer. However, the final accuracy of this network was lower than the first one, as it had an accuracy of typically below 85%. Training progress is shown in Figure 5.5

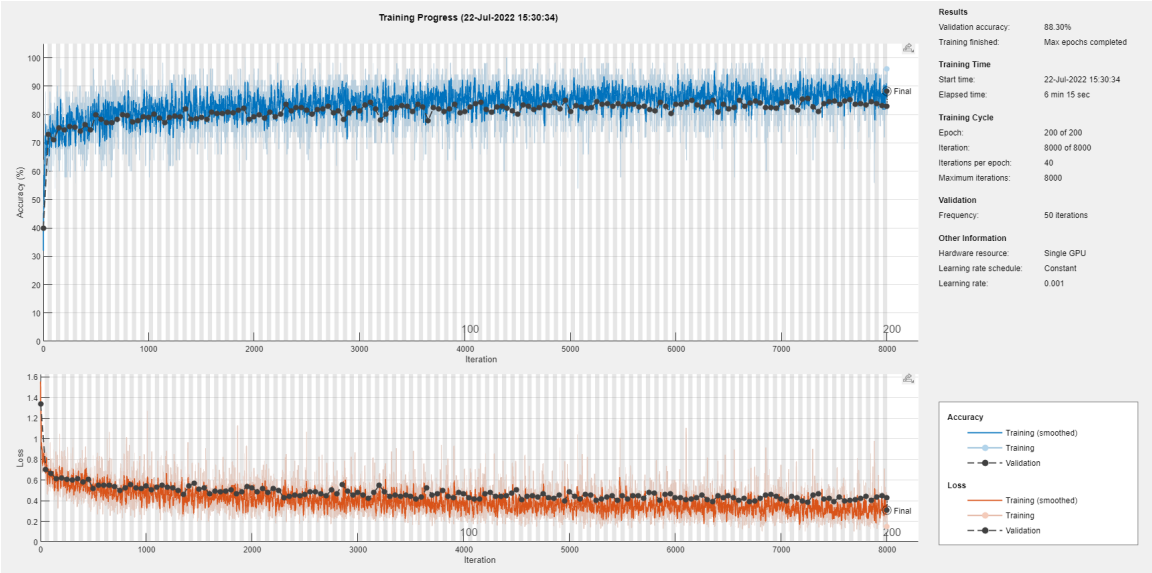

Figure 5.4: A training process of the road grade classification neural network

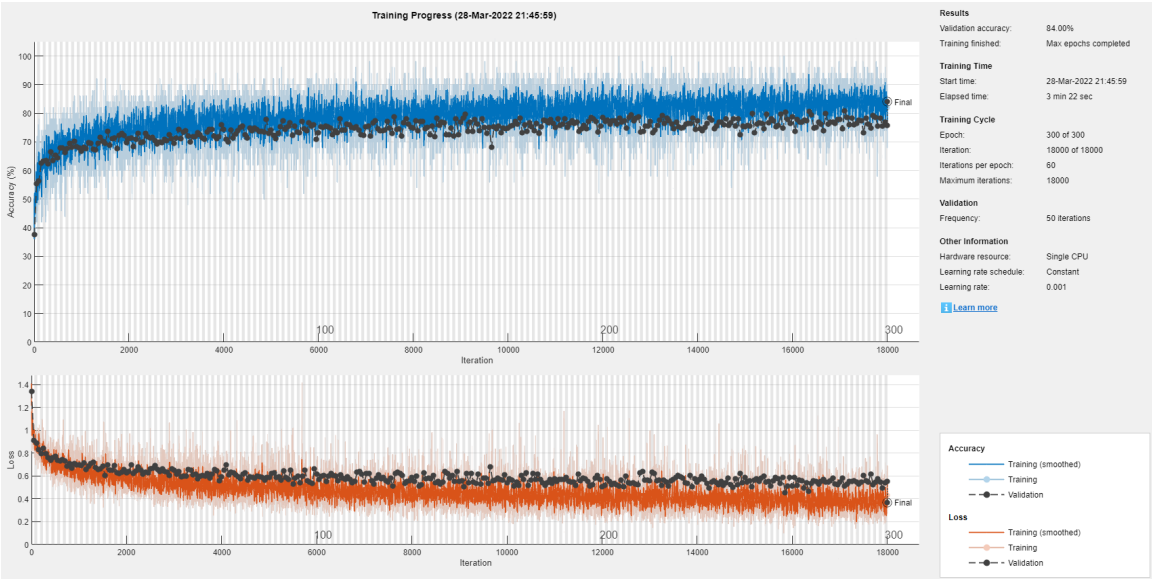

Figure 5.5: A training process of the suspension mode selection

CHAPTER 6
SIMULATION RESULTS AND ANALYSIS

This chapter includes all the results from simulations and discussions of the results. It was already shown that the both neural networks can be trained to reach an average accuracy around 85 % tested by the data collected in advance. Nevertheless, the final goal was to see their performances through the Simulink control loop. The first part of this chapter is a conclusion of the accuracy of neural networks in visual ride event simulations before sending the suspension mode number to the block of computing extra forces. The second part will introduce the final suspension system performances both with and without the trained neural networks, where by comparing the data of before and after, the effects that networks brought are studied as well.

6.1 Performance of Neural Networks

6.1.1 Road grade identification Neural Network

When evaluating the ability of recognizing road grade, the level of prediction performance of neural network varies on different road grades. As shown in following Figures 5.1–5.4, the accuracy of predicting road grade 3 and 6 were close to 100%. On the contrary,

the abilities of predicting road grade 4 and 5 were very unstable. To be specific, by calculating from the data in the plots, it was found that the accuracy of predictions for road grade 4 was only 40% when the accuracy of predictions for road grade 5 was 64%. At the same time, either predictions had random noises like excitements and crashes during the progresses. However, it was not hard to understand why the accuracy for road grade 3 and 6 were higher than that of the road grade 4 and 5, as road grade 3 was the smoothest road input while road grade 6 was the most uneven road profile.

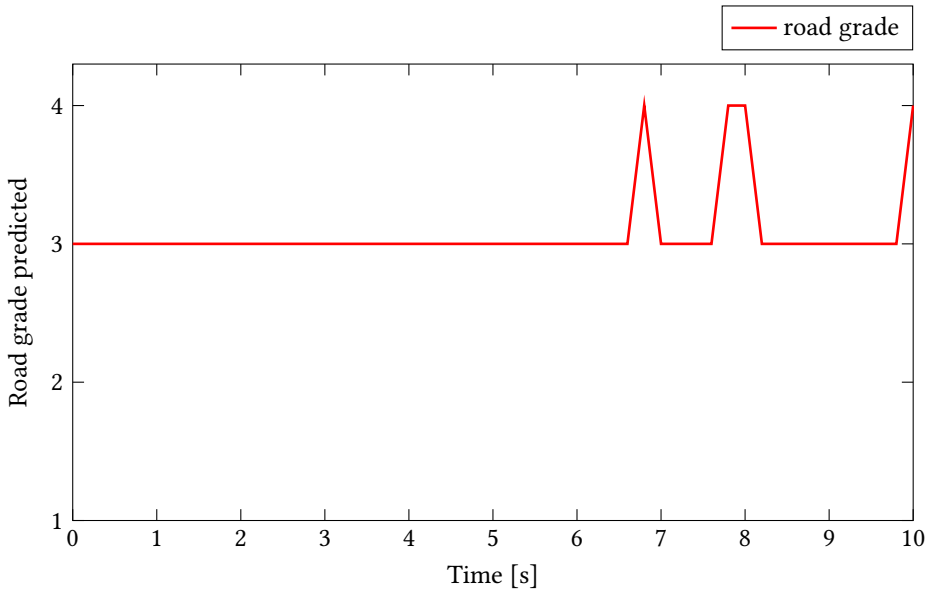

Figure 6.1: Classification performance for random road with grade 3 roughness

6.1.2 Prediction results of suspension mode

In the vehicle model, the front suspension mass and the rear suspension stiffness was set at 25000 N/m, when the front suspension damping c_f was 1000 N/m and the rear suspension damping C_r was 1200 N/m. According to the original specifications of suspension, the extra spring stiffness was set as 20000 N/m and the extra damping was 600 N/m. In this case, as mode value can be 0, 0.5 and 1, to compute the extra force:

$$\text{mode value} \times 20000 \times \text{spring displacement} = \text{the amount of extra spring force} \tag{6.1}$$

$$\text{mode value} \times 600 \times \text{damper speed} = \text{the amount of extra damping force} \tag{6.2}$$

Specifically, mode value 0 was switched on when the road grade was 3; mode value 0.5 was on when the road grade was 4 and 5; mode value 1 applied when the road grade was 6. In Figures 5.5 – 5.8, the results show that when the input road class was 3, the accuracy of prediction was high, while in the case of road grade 4, the output value from the second neural network were switching back and forth from 0 to 0.5. For road class 5, although half of the predictions are correct (0.5), the other half of the prediction included both 0 and 1. Besides, when the simulation completed on a grade 6 road, the predictions were given showed 84% of the suspension mode was 1. The other 16% of prediction results gave the suspension mode value 0.5. The performance of both neural networks were similar as they were performing much better for road grade 3 and 6, than for road grade 4 and 5.

6.2 Suspension performances comparison

The goal of the whole simulation progress was to see how suspension performances changes by comparing the data of before and after, so that the amount of effects that networks brought were shown. The evaluation criteria were pitch velocity and roll velocity. The reason why the accelerations were not included was the level of difficulty to calculate from the before and after data of accelerations, they were found out as very randomly such that there were no value to compare the data of accelerations.

As seen in the plots for road grade 3, it was not surprising at all as the suspension performance had very little change because the default set for road grade 3 was 0. Since there were no additional forces been added to the suspension system, the pitch and roll velocities measured should not change. According to the performance of the second neural network, the accuracy of prediction was fairly high so that the output numbers were almost all zeros, which also helped the after-trained suspension system to perform as before.

In the case of road grade 4, the range of pitch velocity of the untrained system was larger then the range of pitch velocity after trained by the neural networks. However, it was unstable and the difference between performances was small. From the aspect of roll velocity, it was hard to say the performance was improved as it was improved in some time segments but also decreased in some others. At the same time, the lowest roll velocity before training was lower than the lowest roll velocity after training, while the highest roll velocity for the trained system was higher than the untrained system. This result were predictable as neural network2 was not accurate in the case of road grade 4, it was swinging between 0 to 0.5 which means the right suspension modes weren't inputted correctly during the simulation.

In terms of the performance of suspension system on a grade 5, it was not fair by saying the suspension performance was improved during the whole simulation progress, but the waves of the trained curves look obviously smaller than they were. Looking back at the performance of neural network 2 on road grade 5, the results of mode selection mostly lying on the level of 0.5 with some random increase to 1 and decrease to 0. If to evaluate the performance according to the following equation:

$$Average\ improvement\% = \frac{\sum \frac{(trained\ velocity - untrained\ velocity)}{trained\ velocity}}{Amount\ of\ datasets}\% \tag{6.3}$$

The performance in terms of pitch velocity was improved 65% and the average roll velocity was decreased 17% after adding extra forces to the suspension system. What's more, with regard to the performance of suspension system on a grade6, the waves of the plots looks obviously smaller than they were. Meanwhile, looking back at the performance of neural network2 on road grade6, the results of mode selection mostly lying on the level of 1 with a very small amount of output decreased to 0.5. In conclusion, the suspension system performance also got improved in the case of road grade6, with an average improvement on pitch performance of 70% and an average improvement on roll performance of 49%. A clarification has to be made here is that the way of calculating the accuracy was only for reference because the decreases in velocities were not happening during the entire process of the simulation.

6.3 Conclusion

For both neural networks, the levels of accuracy for road grade 3 and road grade 6 were fairly high while the levels were low for the other road grades. Because the boundary between grade 4 and grade 5 was vague, chances were the neural networks can't recognize between them correctly. In addition, the amount of training data was not enough as the accuracy of neural networks increase with more groups of data being trained. In the final section of comparing the suspension performances before and after the neural networks were been trained, the levels of ride comfort evaluated by pitch and roll velocities were very similar in the cases of road grade 3 and road grade 4. Improvements became observable when the model was driving on the road input grade 5 and 6. Although the calculations can prove the existences of improvements, the amount of improvement can't be confirmed just by the value of average percentages.

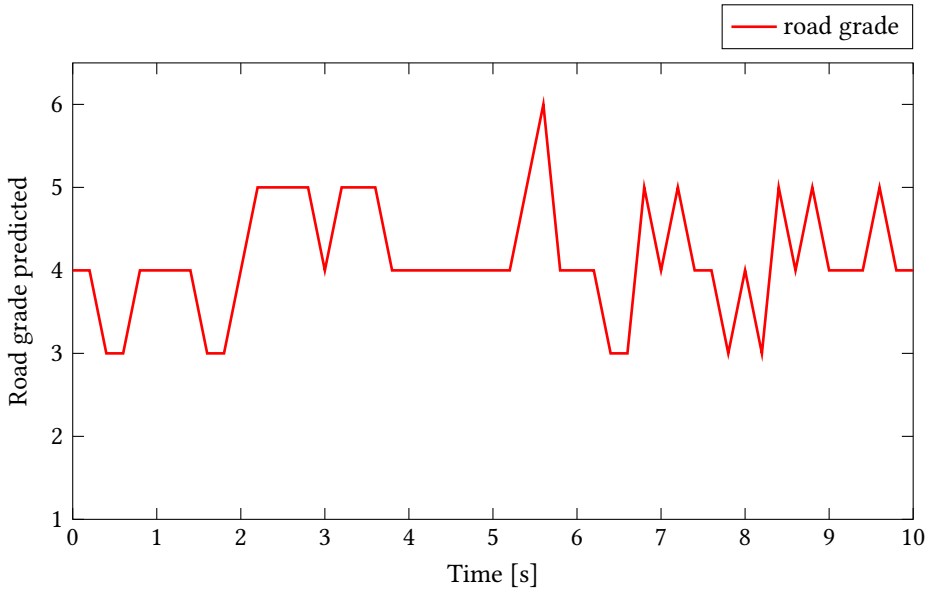

Figure 6.2: Classification performance for random road with grade 4 roughness

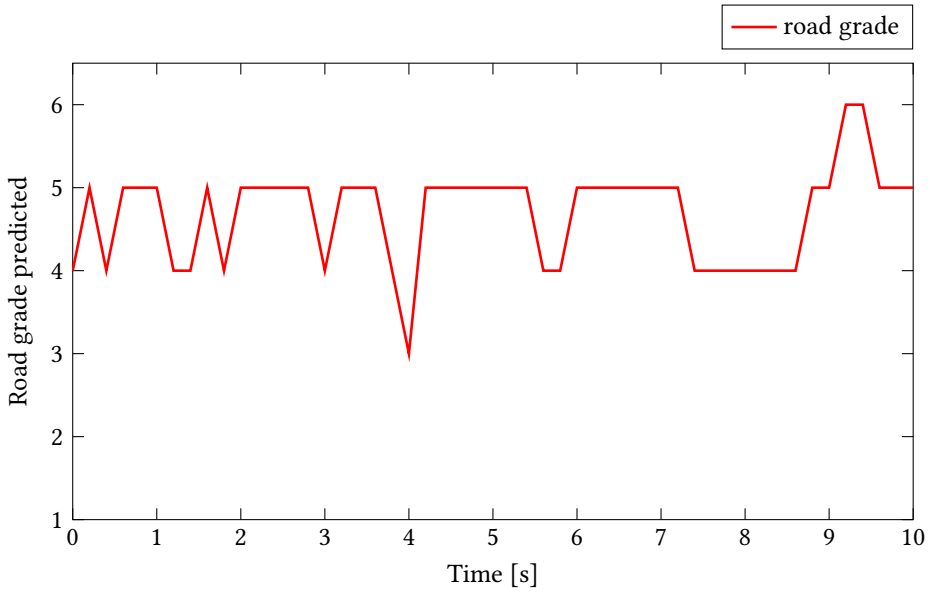

Figure 6.3: Classification performance for random road with grade 5 roughness

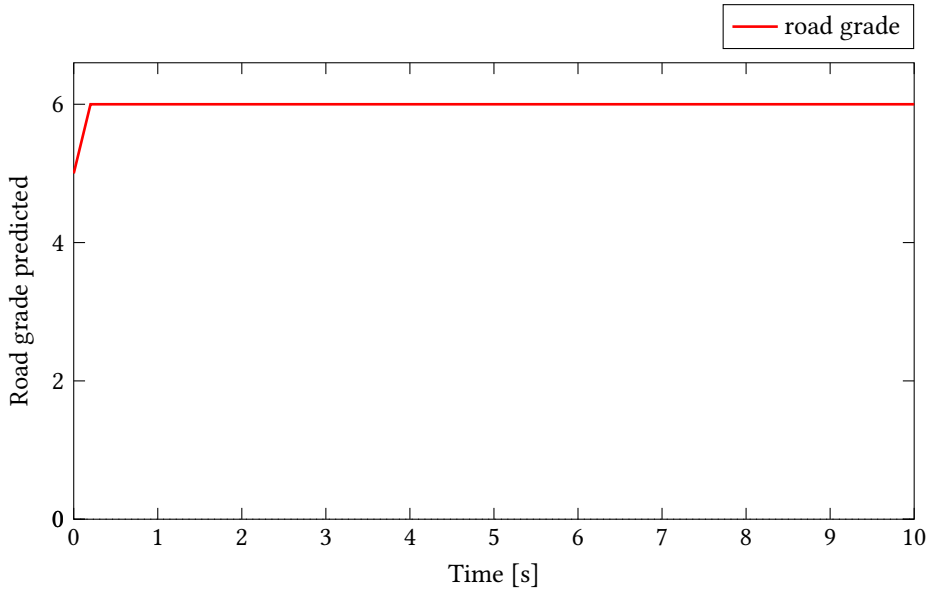

Figure 6.4: Classification performance for random road with grade 6 roughness

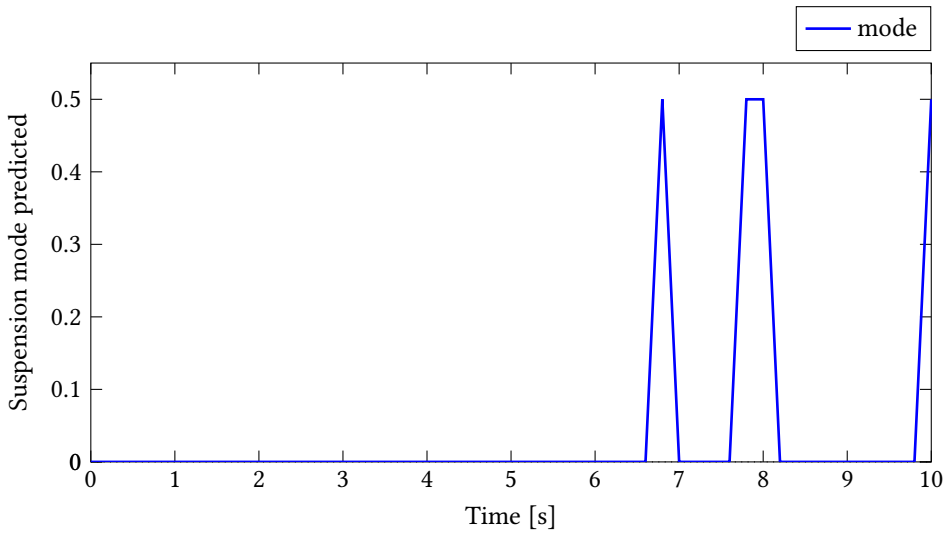

Figure 6.5: Prediction mode for Random road grade 3

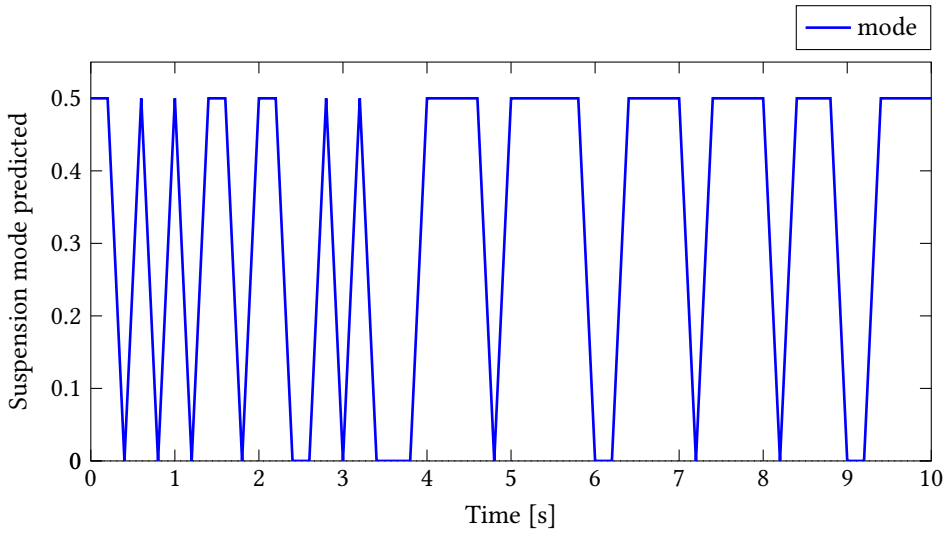

Figure 6.6: Prediction mode for Random road grade 4

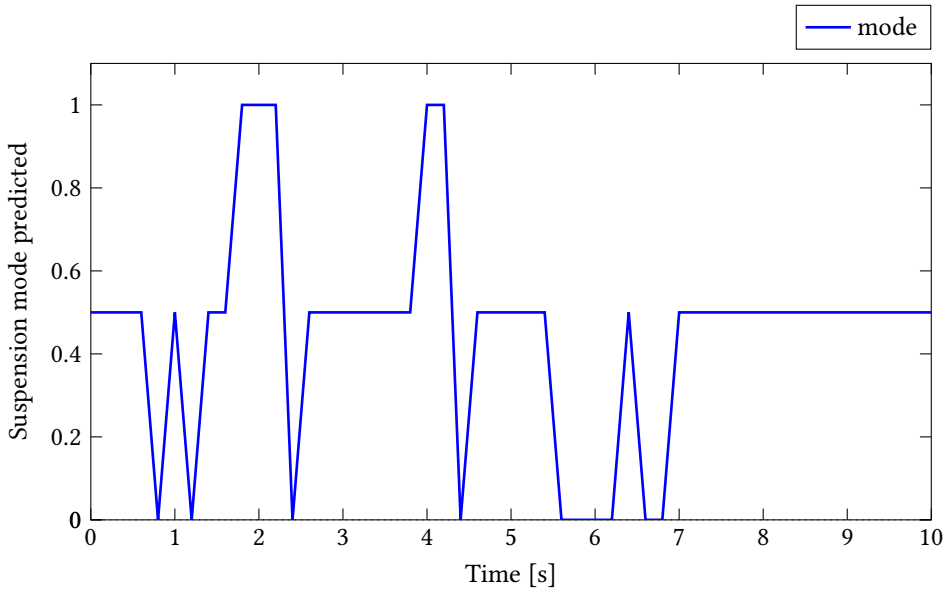

Figure 6.7: Prediction mode for Random road grade 5

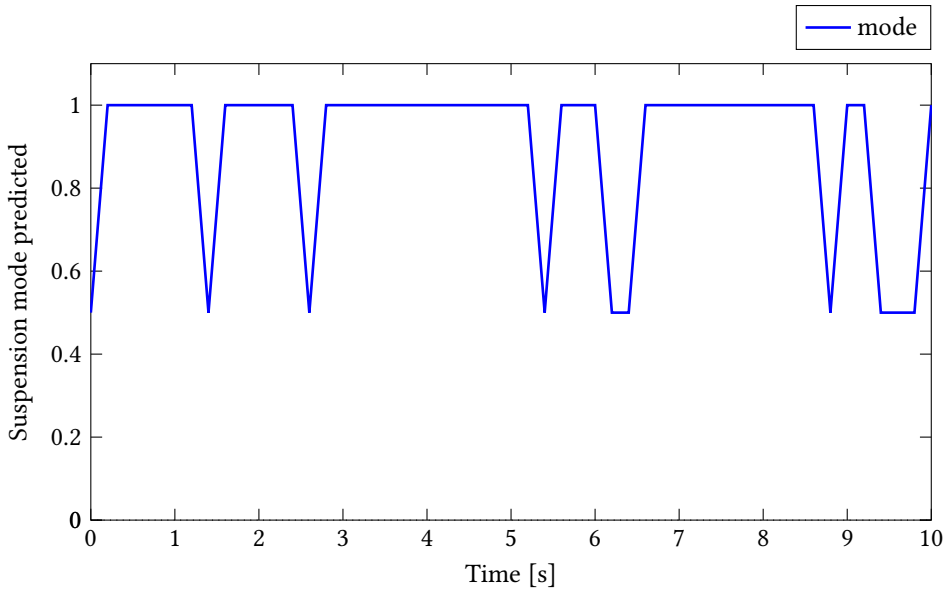

Figure 6.8: Prediction mode for Random road grade 6

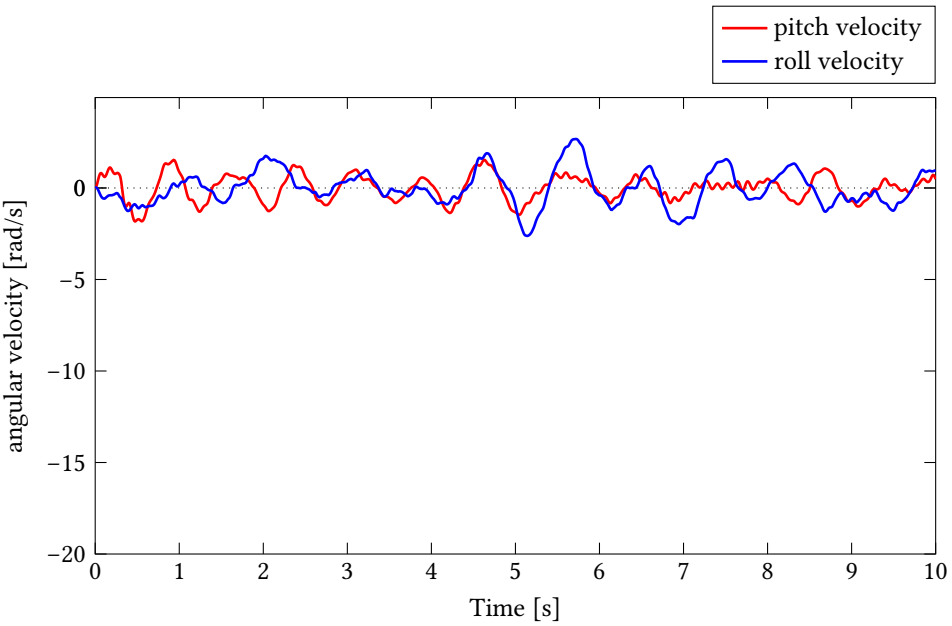

Figure 6.9: Suspension performance on road grade 3 before training

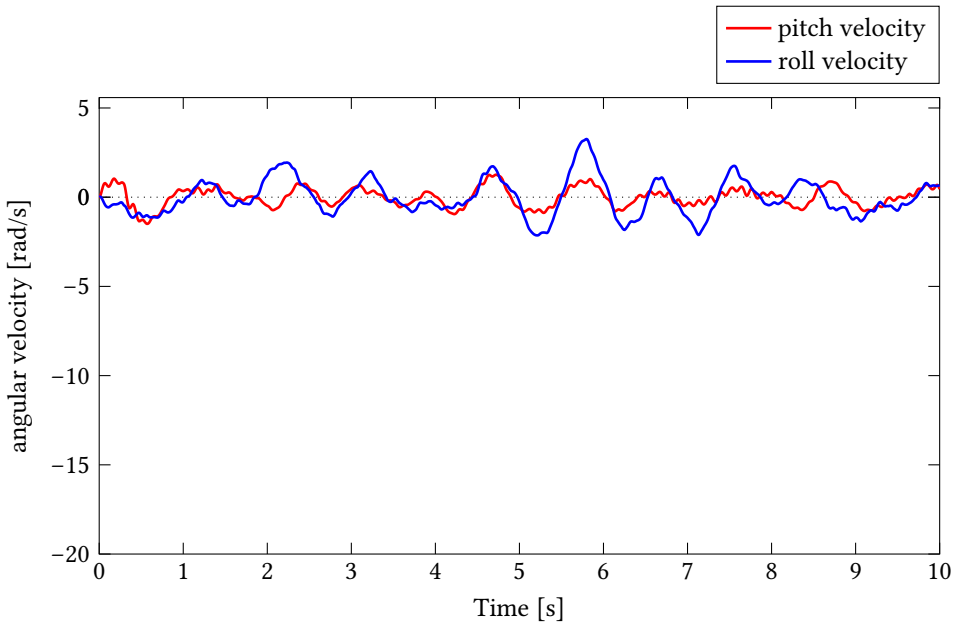

Figure 6.10: Suspension performance on road grade 3 after training

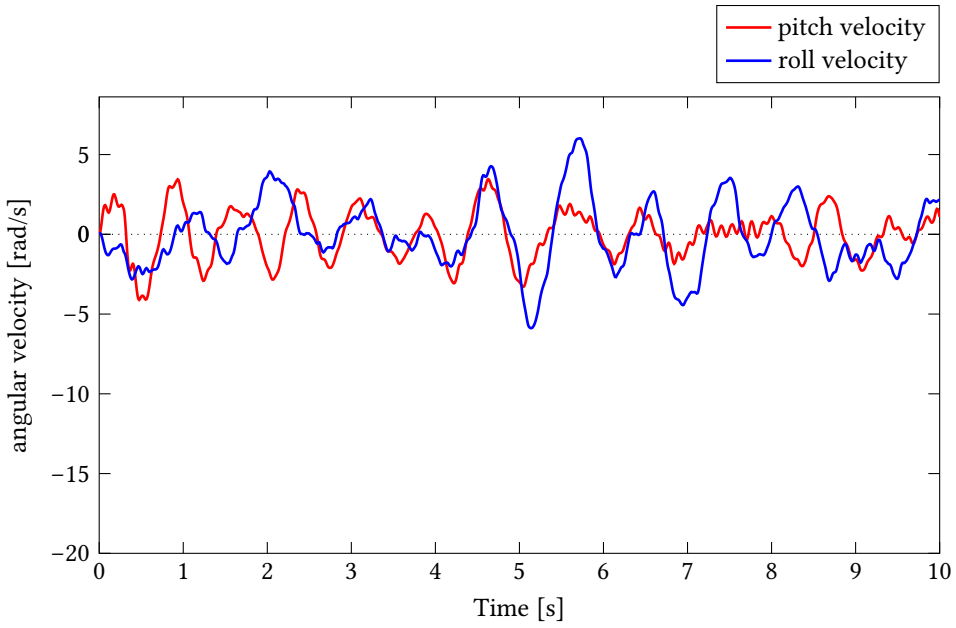

Figure 6.11: Suspension performance on road grade 4 before training

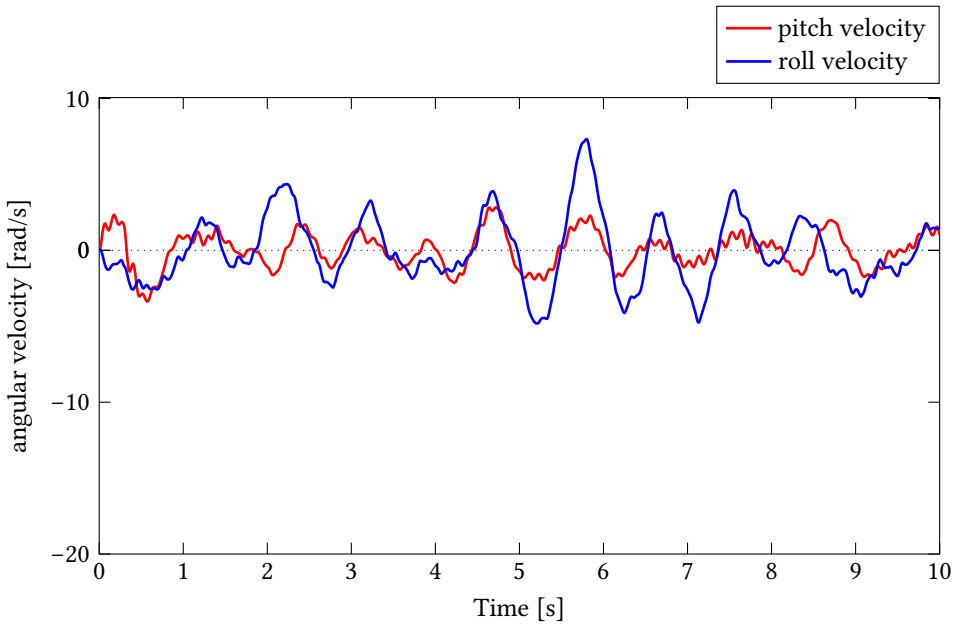

Figure 6.12: Suspension performance on road grade 4 after training

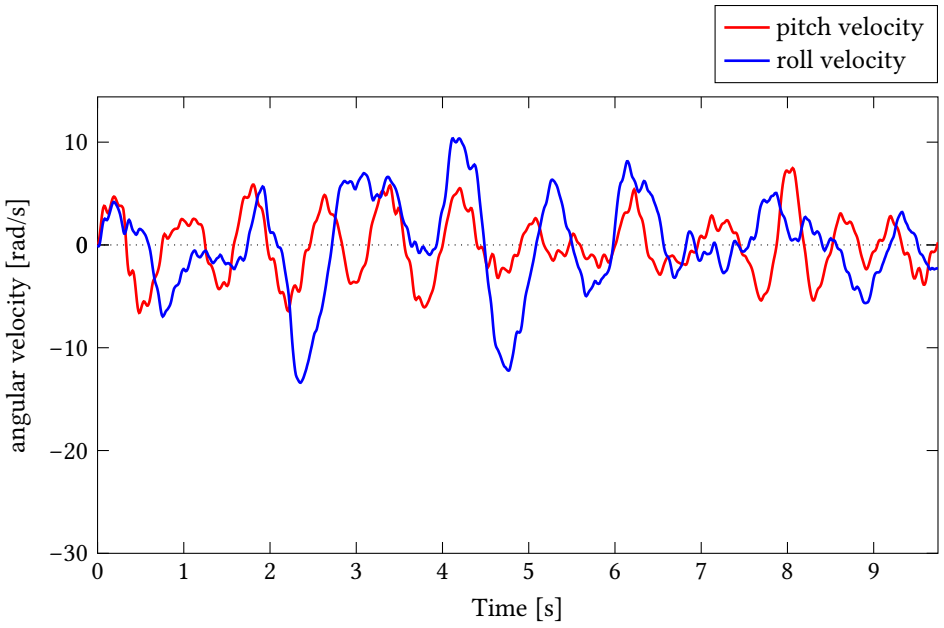

Figure 6.13: Suspension performance on road grade 5 before training

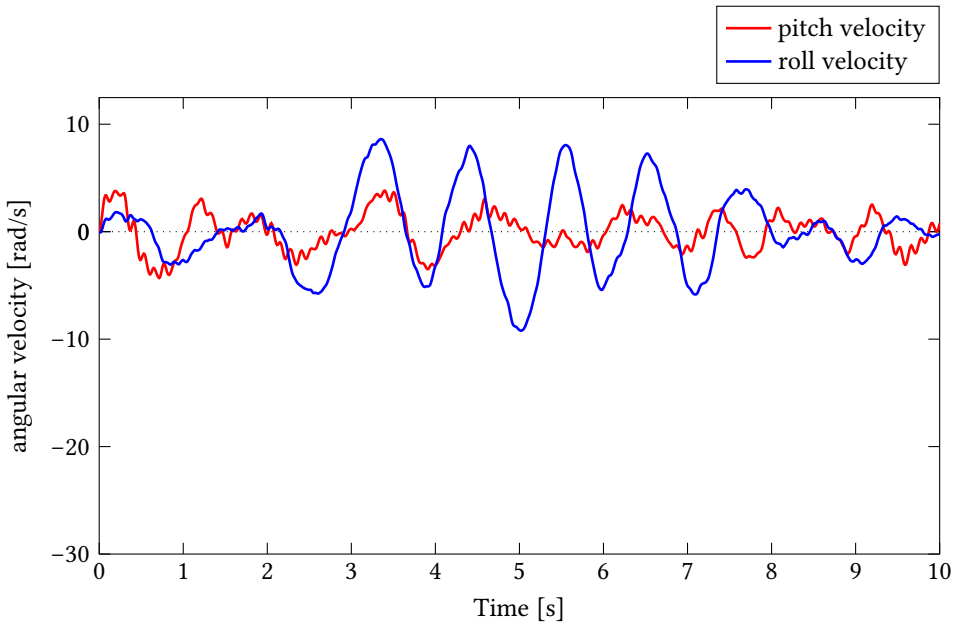

Figure 6.14: Suspension performance on road grade 5 after training

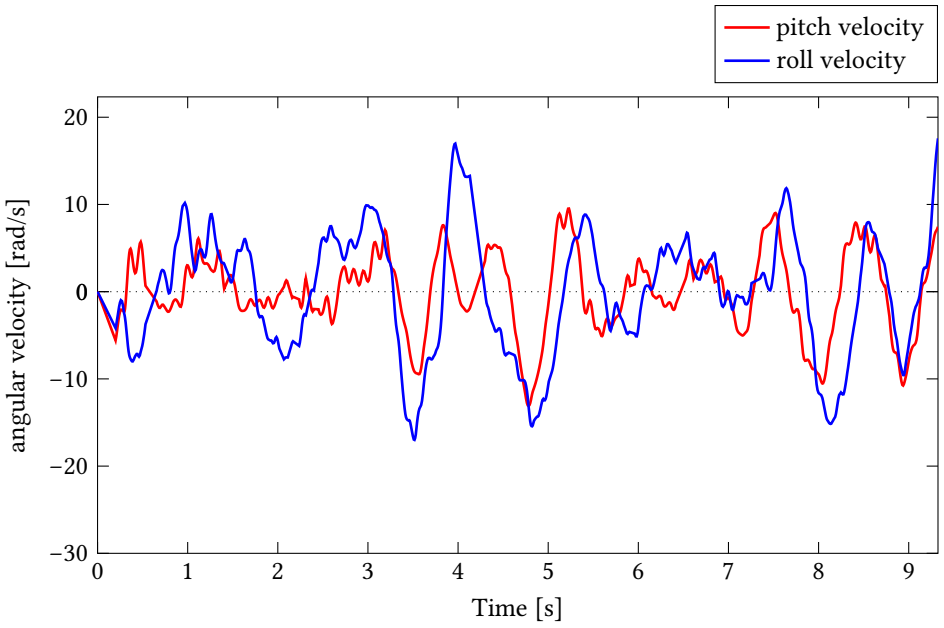

Figure 6.15: Suspension performance on road grade 6 before training

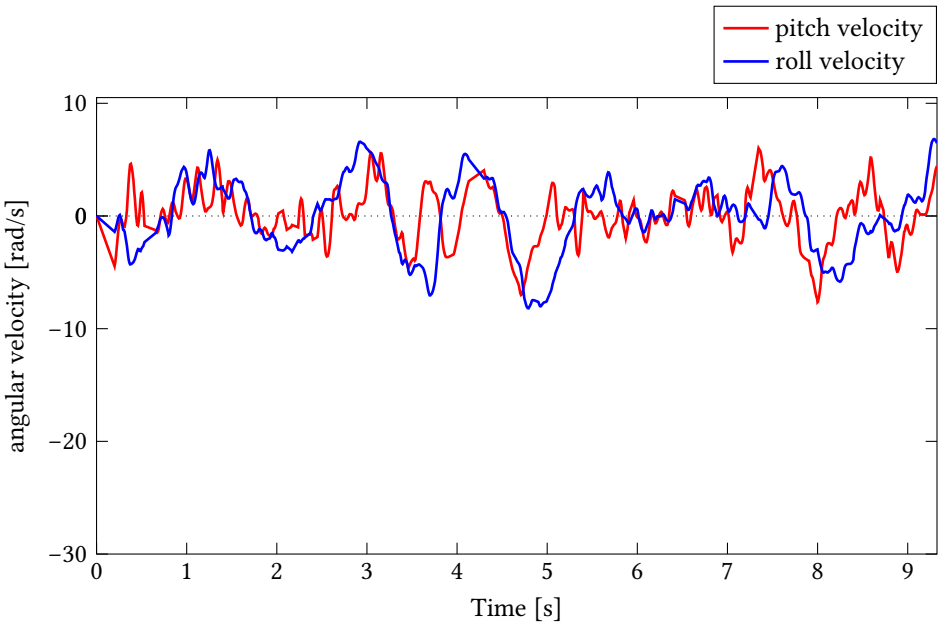

Figure 6.16: Suspension performance on road grade 6 after training

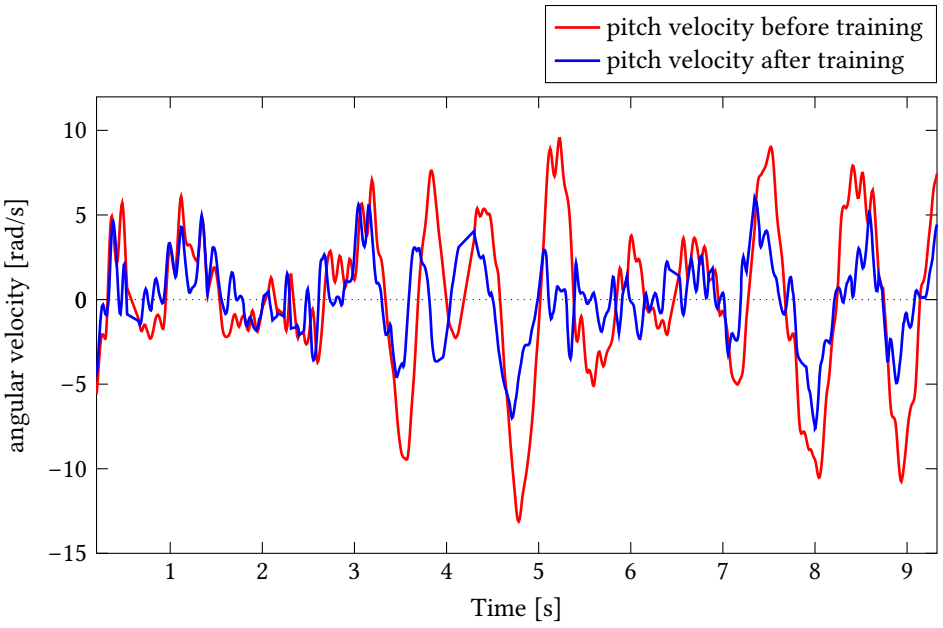

Figure 6.17: Pitch velocities comparison Before and After for road grade 6

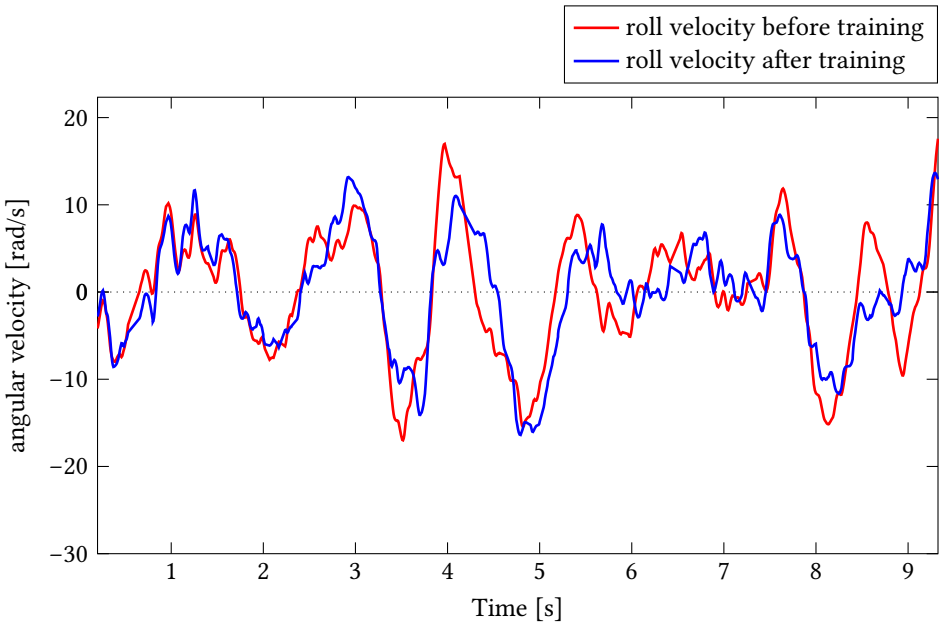

Figure 6.18: Roll velocities comparison Before and After for road grade 6

CHAPTER 7

FURTHER DISCUSSION AND RECOMMENDATIONS

This research was not only a progress of building a full vehicle model, applying Newton's equations of motion on a multibody system at steady state, but also a learning progress of Neural Networks. Nowadays, the development of artificial intelligence technology is unstoppable. Software developers around the world have developed a great number of methodologies to avoid manual thinking that is limited by human bodies. Using different machine learning technologies to solve problems efficiently is necessary as many calculations can be extremely time consuming in real life. It is no doubt that all the researchers in automotive industry are eager to borrow the power of AI in order to get closer to cutting-edge technologies. Although a fully intelligent suspension system has not yet appeared and applied in practical applications, the simulation data that can be collected from various visual analysis gives the manufactures enormous useful references to make decisions. The results of this research show that road quality detection through neural networks is very feasible, and that this information should be available to allow improvements in ride quality. However, because the neural network is trained based on vehicle properties, making active changes in vehicle behaviour can confuse the neural network.

7.1 Simulation limitations

The simulations were conducted on a 2D road to simplify the model and the progress of the simulation. However, road has 3 dimensions in real world, where the lateral accelerations are quite influenced such that weight transfer happens a lot. Meanwhile, the simulations in this research neglected the time required to switch mode and to make predictions of the following road input. To be realistic, it would be much more practical if the time delay was taken into account and use neural networks in different frequency domains.

In related to the neural networks in the Simulink control loop, the study of how to select input parameters for neural network was far away from enough. Although the neural networks showed its ability of recognizing road grade and selecting modes for the suspension system, the weaknesses were obvious. The poor accuracy of prediction for road grades in the middle(4 and 5) had multiple negative effects on the performances. At the same time, the mode value could only be 0, 0.5 and1 for four road classes, and there were no strategies of choosing the values of extra spring stiffness and damping. The numbers were chosen only by trying different values and running simulations to see how much extra force were needed so as to make observable improvements.

7.2 Recommendations

The simulations can be done using a more practical maneuver by:

- Using 3D road profile for road input.
- Taking the response time for the system to make adjustments into account.
- Digging deeper to find better ways of picking the input parameters.
- Having more accommodations for the mode selection and the amount of extra suspension forces.

REFERENCES

[1] Marco Di Vittorio. *Regenerative damping-Analysis of expected benefits on CO 2 emissions and ride-comfort improvement.* 2016.

[2] Ankita R Bhise et al. "Comparison between passive and semi-active suspension system using matlab/simulink". In: *IOSR Journal of Mechanical and Civil Engineering* 13.4 (2016), pp. 1–6.

[3] Mahmoud Omar, MM El-Kassaby, and Walid Abdelghaffar. "A universal suspension test rig for electrohydraulic active and passive automotive suspension system". In: *Alexandria Engineering Journal* 56.4 (2017), pp. 359–370.

[4] Bruce P Minaker. *Fundamentals of vehicle dynamics and modelling: A textbook for engineers with illustrations and examples.* John Wiley & Sons, 2019.

[5] José L Escalona and Antonio M Recuero. "A bicycle model for education in multibody dynamics and real-time interactive simulation". In: *Multibody System Dynamics* 27.3 (2012), pp. 383–402.

[6] J.P. Meijaard, AR Savkoor, and G Lodewijks. "Potential for vehicle ride improvement using both suspension and aerodynamic actuators". In: *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.* Vol. 1. IEEE. 2005, pp. 385–390.

[7] Dean Karnopp. "Active damping in road vehicle suspension systems". In: *Vehicle System Dynamics* 12.6 (1983), pp. 291–311.

[8] Pieter Schalk Els et al. "The ride comfort vs. handling compromise for off-road vehicles". In: *Journal of Terramechanics* 44.4 (2007), pp. 303–317.

[9] Michal Makowski and Lech Knap. "Investigation of an off-road vehicle equipped with magnetorheological dampers". In: *Advances in Mechanical Engineering* 10.5 (2018), p. 1687814018778222.

[10] J McCarthy et al. "The Dartmouth summer research project on artificial intelligence". In: *Artificial intelligence: past, present, and future* (1956).

[11] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

[12] A Sinkov et al. "Neural networks in data mining". In: *2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*. IEEE. 2016, pp. 1–5.

[13] Feng Jiang, Xue Yang, and Shuyu Li. "Comparison of forecasting India's energy demand using an MGM, ARIMA model, MGM-ARIMA model, and BP neural network model". In: *Sustainability* 10.7 (2018), p. 2225.

[14] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *towards data science* 6.12 (2017), pp. 310–316.

[15] Peter Múčka. "Simulated road profiles according to ISO 8608 in vibration analysis". In: *Journal of Testing and Evaluation* 46.1 (2017), pp. 405–418.

APPENDIX A
JULIA MODEL DETAILS

A.1 Code

This section presents the source code used to generate the vehicle model equations.

A.1.1 Input file for jeep model

```
function input_jeep(;u=10,a=2.946*0.47,b=2.946*0.53,tf=1.572,tr=1.572,kf=25000,
kr=25000,cf=1000,cr=1200,m=2000,
Ix=800,Iy=3200,Iz=3200,kt=210000,
muf=80,mur=80,r=0.419,kfr=0.25*kf*(tf-0.4)^2,krr=0.25*kr*(tr-0.4)^2)

the_system=mbd_system("Jeep Model")

# add one body representing the chassis
item=body("chassis")
item.mass=m
item.moments_of_inertia=[Ix,Iy,Iz]  ## Only the Iy term matters here
item.products_of_inertia=[0,0,0]
item.location=[0,0,0.8]  ## Put cg at origin, but offset vertically to make animation more clear
item.velocity=[u,0,0]
push!(the_system.item,item)
```

```
push!(the_system.item,weight(item))

item=body("front axle")
item.mass=muf
item.moments_of_inertia=[1/12*muf*tf^2,0,1/12*muf*tf^2]
item.location=[a,0,r]
item.velocity=[u,0,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))

item=body("rear axle")
item.mass=mur
item.moments_of_inertia=[1/12*mur*tr^2,0,1/12*mur*tr^2]
item.location=[-b,0,r]
item.velocity=[u,0,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))


item=body("LF wheel+hub")
item.mass=40
item.moments_of_inertia=[2,4,2]
item.location=[a,tf/2,r]
item.velocity=[u,0,0]
item.angular_velocity=[0,u/r,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))


item=body("RF wheel+hub")
item.mass=40
item.moments_of_inertia=[2,4,2]
item.location=[a,-tf/2,r]
item.velocity=[u,0,0]
item.angular_velocity=[0,u/r,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))
```

```
item=body("LR wheel+hub")
item.mass=40
item.moments_of_inertia=[2,4,2]
item.location=[-b,tr/2,r]
item.velocity=[u,0,0]
item.angular_velocity=[0,u/r,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))


item=body("RR wheel+hub")
item.mass=40
item.moments_of_inertia=[2,4,2]
item.location=[-b,-tr/2,r]
item.velocity=[u,0,0]
item.angular_velocity=[0,u/r,0]
push!(the_system.item,item)
push!(the_system.item,weight(item))

# front suspension
item=flex_point("front anti-roll")
item.body[1]="front axle"
item.body[2]="chassis"
item.location=[a,0,r]
item.stiffness=[0,kfr]
item.forces=0
item.moments=1
item.axis=[1,0,0]
push!(the_system.item,item)

# rear suspension
item=flex_point("rear anti-roll")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location=[-b,0,r]
item.stiffness=[0,krr]
item.forces=0
item.moments=1
item.axis=[1,0,0]
```

```
push!(the_system.item,item)

#bearings
item=rigid_point("wheel bearing")
item.body[1]="LF wheel+hub"
item.body[2]="front axle"
item.location=[a,tf/2,r]
item.forces=3
item.moments=2
item.axis=[0,1,0]
push!(the_system.item,item)

item=rigid_point("wheel bearing")
item.body[1]="RF wheel+hub"
item.body[2]="front axle"
item.location=[a,-tf/2,r]
item.forces=3
item.moments=2
item.axis=[0,1,0]
push!(the_system.item,item)


item=rigid_point("wheel bearing")
item.body[1]="LR wheel+hub"
item.body[2]="rear axle"
item.location=[-b,tr/2,r]
item.forces=3
item.moments=2
item.axis=[0,1,0]
push!(the_system.item,item)

item=rigid_point("wheel bearing")
item.body[1]="RR wheel+hub"
item.body[2]="rear axle"
item.location=[-b,-tr/2,r]
item.forces=3
item.moments=2
item.axis=[0,1,0]
push!(the_system.item,item)
```

```
# tires
item=flex_point("left front tire, vertical")
item.body[1]="LF wheel+hub"
item.body[2]="ground"
item.stiffness=[kt,0]
item.location=[a,tf/2,0]
item.forces=1
item.moments=0
item.axis=[0,0,1]
item.rolling_axis=[0,1,0]
push!(the_system.item,item)

item=flex_point("right front tire, vertical")
item.body[1]="RF wheel+hub"
item.body[2]="ground"
item.stiffness=[kt,0]
item.location=[a,-tf/2,0]
item.forces=1
item.moments=0
item.axis=[0,0,1]
item.rolling_axis=[0,1,0]
push!(the_system.item,item)

item=flex_point("left rear tire, vertical")
item.body[1]="LR wheel+hub"
item.body[2]="ground"
item.stiffness=[kt,0]
item.location=[-b,tr/2,0]
item.forces=1
item.moments=0
item.axis=[0,0,1]
item.rolling_axis=[0,1,0]
push!(the_system.item,item)

item=flex_point("right rear tire, vertical")
item.body[1]="RR wheel+hub"
item.body[2]="ground"
```

```
item.stiffness=[kt,0]
item.location=[-b,-tr/2,0]
item.forces=1
item.moments=0
item.axis=[0,0,1]
item.rolling_axis=[0,1,0]
push!(the_system.item,item)


item=flex_point("left front tire, horizontal")
item.body[1]="LF wheel+hub"
item.body[2]="ground"
item.location=[a,tf/2,0]
item.damping=[30000/u,0]
item.forces=2
item.moments=0
item.axis=[0,0,1]
push!(the_system.item,item)

item=flex_point("right front tire, horizontal")
item.body[1]="RF wheel+hub"
item.body[2]="ground"
item.location=[a,-tf/2,0]
item.damping=[30000/u,0]
item.forces=2
item.moments=0
item.axis=[0,0,1]
push!(the_system.item,item)


item=flex_point("LR tire, horizontal")
item.body[1]="LR wheel+hub"
item.body[2]="ground"
item.location=[-b,tr/2,0]
item.damping=[30000/u,0]
item.forces=2
item.moments=0
item.axis=[0,0,1]
push!(the_system.item,item)
```

```
item=flex_point("RR tire, horizontal")
item.body[1]="RR wheel+hub"
item.body[2]="ground"
item.location=[-b,-tr/2,0]
item.damping=[30000/u,0]
item.forces=2
item.moments=0
item.axis=[0,0,1]
push!(the_system.item,item)


# suspension constraints
item=link("susp link")
item.body[1]="chassis"
item.body[2]="front axle"
item.location[1]=[a-0.4,tf/2-0.2,r-0.1]
item.location[2]=[a,tf/2-0.2,r-0.1]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="front axle"
item.location[1]=[a-0.4,-(tf/2-0.2),r-0.1]
item.location[2]=[a,-(tf/2-0.2),r-0.1]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="front axle"
item.location[1]=[a-0.4,tf/2-0.2,r+0.2]
item.location[2]=[a,tf/2-0.4,r+0.2]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="front axle"
item.location[1]=[a-0.4,-(tf/2-0.2),r+0.2]
item.location[2]=[a,-(tf/2-0.4),r+0.2]
```

```
push!(the_system.item,item)


item=link("susp link")
item.body[1]="chassis"
item.body[2]="rear axle"
item.location[1]=[-b+0.4,tr/2-0.2,r-0.1]
item.location[2]=[-b,tr/2-0.2,r-0.1]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="rear axle"
item.location[1]=[-b+0.4,-(tr/2-0.2),r-0.1]
item.location[2]=[-b,-(tr/2-0.2),r-0.1]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="rear axle"
item.location[1]=[-b+0.4,tr/2-0.2,r+0.2]
item.location[2]=[-b,tr/2-0.4,r+0.2]
push!(the_system.item,item)

item=link("susp link")
item.body[1]="chassis"
item.body[2]="rear axle"
item.location[1]=[-b+0.4,-(tr/2-0.2),r+0.2]
item.location[2]=[-b,-(tr/2-0.4),r+0.2]
push!(the_system.item,item)


# front suspension
item=spring("left front spring")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,tf/2-0.2,r]
item.location[2]=[a,tf/2-0.2,r+0.4]
item.stiffness=kf
```

```
item.damping=cf
push!(the_system.item,item)

# front suspension
item=spring("right front spring")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,-(tf/2-0.2),r]
item.location[2]=[a,-(tf/2-0.2),r+0.4]
item.stiffness=kf
item.damping=cf
push!(the_system.item,item)

# rear suspension
item=spring("left rear spring")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,tr/2-0.2,r]
item.location[2]=[-b,tr/2-0.2,r+0.4]
item.stiffness=kr
item.damping=cr
push!(the_system.item,item)

# rear suspension
item=spring("right rear spring")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,-(tr/2-0.2),r]
item.location[2]=[-b,-(tr/2-0.2),r+0.4]
item.stiffness=kr
item.damping=cr
push!(the_system.item,item)


# front suspension force
item=actuator("left front actuator")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,tf/2-0.2,r]
```

```
item.location[2]=[a,tf/2-0.2,r+0.4]
push!(the_system.item,item)

item=actuator("left rear actuator")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,tr/2-0.2,r]
item.location[2]=[-b,tr/2-0.2,r+0.4]
push!(the_system.item,item)

item=actuator("right front actuator")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,-(tf/2-0.2),r]
item.location[2]=[a,-(tf/2-0.2),r+0.4]
push!(the_system.item,item)

item=actuator("right rear actuator")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,-(tr/2-0.2),r]
item.location[2]=[-b,-(tr/2-0.2),r+0.4]
push!(the_system.item,item)

# force motion
item=actuator("z0_LF")
item.body[1]="LF wheel+hub"
item.body[2]="ground"
item.location[1]=[a,tf/2,0]
item.location[2]=[a,tf/2,-0.1]
item.gain=kt
push!(the_system.item,item)

item=actuator("z0_LR")
item.body[1]="LR wheel+hub"
item.body[2]="ground"
item.location[1]=[-b,tr/2,0]
item.location[2]=[-b,tr/2,-0.1]
item.gain=kt
```

```
push!(the_system.item,item)

item=actuator("z0_RF")
item.body[1]="RF wheel+hub"
item.body[2]="ground"
item.location[1]=[a,-tf/2,0]
item.location[2]=[a,-tf/2,-0.1]
item.gain=kt
push!(the_system.item,item)

item=actuator("z0_RR")
item.body[1]="RR wheel+hub"
item.body[2]="ground"
item.location[1]=[-b,-tr/2,0]
item.location[2]=[-b,-tr/2,-0.1]
item.gain=kt
push!(the_system.item,item)

# measure spring displacements
item=sensor("left front spring")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,tf/2-0.2,r]
item.location[2]=[a,tf/2-0.2,r+0.4]
push!(the_system.item,item)

item=sensor("left rear spring")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,tr/2-0.2,r]
item.location[2]=[-b,tr/2-0.2,r+0.4]
push!(the_system.item,item)

item=sensor("right front spring")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,-(tf/2-0.2),r]
item.location[2]=[a,-(tf/2-0.2),r+0.4]
push!(the_system.item,item)
```

```
item=sensor("right rear spring")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,-(tr/2-0.2),r]
item.location[2]=[-b,-(tr/2-0.2),r+0.4]
push!(the_system.item,item)


# measure damper velocities
item=sensor("left front damper")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,tf/2-0.2,r]
item.location[2]=[a,tf/2-0.2,r+0.4]
item.order=2
push!(the_system.item,item)

item=sensor("left rear damper")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,tr/2-0.2,r]
item.location[2]=[-b,tr/2-0.2,r+0.4]
item.order=2
push!(the_system.item,item)

item=sensor("right front damper")
item.body[1]="front axle"
item.body[2]="chassis"
item.location[1]=[a,-(tf/2-0.2),r]
item.location[2]=[a,-(tf/2-0.2),r+0.4]
item.order=2
push!(the_system.item,item)

item=sensor("right rear damper")
item.body[1]="rear axle"
item.body[2]="chassis"
item.location[1]=[-b,-(tr/2-0.2),r]
item.location[2]=[-b,-(tr/2-0.2),r+0.4]
```

```
item.order=2
push!(the_system.item,item)


# measure body motions
item=sensor("pitch velocity")
item.body[1]="chassis"
item.body[2]="ground"
item.location[1]=[0,0,0.8]
item.location[2]=[0,0.1,0.8]
item.gain=180/pi
item.order=2
item.twist=1
push!(the_system.item,item)

item=sensor("roll velocity")
item.body[1]="chassis"
item.body[2]="ground"
item.location[1]=[0,0,0.8]
item.location[2]=[0.1,0,0.8]
item.gain=180/pi
item.order=2
item.twist=1
push!(the_system.item,item)

item=sensor("ddot z_LR")
item.body[1]="chassis"
item.body[2]="ground"
item.location[1]=[-b,tr/2,r+0.1]
item.location[2]=[-b,tr/2,0]
item.order=3
item.gain=1/9.81
push!(the_system.item,item)

item=sensor("ddot y_LR")
item.body[1]="chassis"
item.body[2]="ground"
item.location[1]=[-b,tr/2,r+0.1]
item.location[2]=[-b,tr/2-0.1,r+0.1]
```

```
item.order=3
item.gain=1/9.81
push!(the_system.item,item)

the_system

end
```

A.1.2 Jeep model in Julia

```
using EoM, DelimitedFiles
include("input_jeep.jl")
my_sys, my_eqns = run_eom(input_jeep, :verbose)
#my_result = analyze(my_eqns, :verbose , decomp=false)

writedlm(joinpath("matlab", "A.out"), my_eqns[end].A)
writedlm(joinpath("matlab", "B.out"), my_eqns[end].B)
writedlm(joinpath("matlab", "C.out"), my_eqns[end].C)
writedlm(joinpath("matlab", "D.out"), my_eqns[end].D)
writedlm(joinpath("matlab", "E.out"), my_eqns[end].E)
```

APPENDIX B

MATLAB MODEL DETAILS

B.1 Matlab Script

```
wb=2.946;
u =10;
t_end = 10;
simin_kd = 0;

[x,z,z2]=random_road(6, u*t_end,0.5);
t=x/u;
simin_z=timeseries([z;z2]',t);

A=importdata('A.out');
B=importdata('B.out');
C=importdata('C.out');
D=importdata('D.out');
E=importdata('E.out');

jeep=dss(A,B,C,D,E);  %% Convert to descriptor state space form, and store a variable readable by

sim('jeep_simulink_4');  %% simulate
```

```
figure(1)
plot(simout.Time,simout.Data(:,1))
ylabel('Pitch velocity [degrees/s]')
figure(2)
plot(simout.Time,simout.Data(:,2))
ylabel('Roll velocity [degrees/s]')
figure(3)
plot(simout.Time,simout.Data(:,3))
ylabel('Vertical accln [g]')
figure(4)
plot(t,[z;z2]')
ylabel('Road input [m]')
legend('L','R')
```

B.2 Road grade identification Neural Network

```
load G3 n3
load G4 n4
load G5 n5
load G6 n6

data = [n3;n4;n5;n6];
data = data(:,[2,3,4,5,1]);

%% split data
[m,~] = size(data);
p = 0.5; %% percentage of total data becomes train data
idx=randperm(m);
train_data = data(idx(1:round(p*m)),:);
rest_data = data(idx(round(p*m)+1:end),:);
[m,~] = size(rest_data);
idx=randperm(m);
val_data = rest_data(idx(1:round(p*m)),:);
test_data = rest_data(idx(round(p*m)+1:end),:);

%% convert train data and validation data into datastore
label_name = 'G_type';
train_table = array2table(train_data,'VariableNames',{'f1','f2','f3','f4',label_name});
```

```
train_table = convertvars(train_table,label_name,'categorical');
train_datastore = arrayDatastore(train_table,'OutputType','same');

%%train_datastore_trans = transform(train_datastore,@(x)
%%[cellfun(@transpose,mat2cell(x(:,1:4),ones(1,1000)),'UniformOutput',false) ,
%%mat2cell(categorical(x(:,5)),ones(1,m))]);

val_table = array2table(val_data,'VariableNames',{'f1','f2','f3','f4',label_name});
val_table = convertvars(val_table,label_name,'categorical');
val_datastore = arrayDatastore(val_table,'OutputType','same');

test_table = array2table(test_data,'VariableNames',{'f1','f2','f3','f4',label_name});
test_table = convertvars(test_table,label_name,'categorical');
test_datastore = arrayDatastore(test_table,'OutputType','same');

head(train_table)

classNames = categories(train_table{:,label_name})
%% define training option parameters
numFeatures = size(train_table,2) - 1;
numClasses = numel(classNames);
miniBatchSize = 50; %% iteration per epoch = number of training samples (1000 in my case)
%%/ miniBatchSize = 20
maxEpochs = 200; %% iterations = iteration per epoch * maxEpoc

%% define all layers in the neural network
layers = [
featureInputLayer(numFeatures,'Normalization',"zscore")
fullyConnectedLayer(32) % 1st stack of layers
batchNormalizationLayer
reluLayer
fullyConnectedLayer(64) % 2nd stack of layers
batchNormalizationLayer
reluLayer
fullyConnectedLayer(128) % 3rd stack of layers
batchNormalizationLayer
reluLayer
fullyConnectedLayer(64) % 4th stack of layers
batchNormalizationLayer
```

```
        reluLayer
        fullyConnectedLayer(32) % 5th stack of layers
        batchNormalizationLayer
        reluLayer
        fullyConnectedLayer(numClasses) % output layers
        softmaxLayer
        classificationLayer]

        %% define training options
        options = trainingOptions('adam','MiniBatchSize',miniBatchSize,'MaxEpochs',maxEpochs,
        'Shuffle','every-epoch','ValidationData',val_table,'Plots','training-progress','Verbose',false)

        %% construct neural network and start to train
        road_net = trainNetwork(train_table,label_name,layers,options);

        %% Test the network
        y_pred = classify(road_net,test_table(:,1:end-1),'MiniBatchSize',miniBatchSize);
        y_true = test_table{:,label_name};
        acc = sum(y_pred == y_true) / numel(y_true)

        %% plot a confusion matrix to visualize the results
        figure
        confusionchart(y_true,y_pred)
        %% save the trained network for further use
        save('road_net.mat',"road_net");
\end{lstlisting}

\section{Neural Network for choosing the suspension mode}
\label{B:Neural Network for choosing the suspension mode}
\begin{lstlisting}
        %% === Data Preparation ===
        %% load data
        load("G3_new.mat");
        load("G4_new.mat");
        load("G5_new.mat");
        load("G6_new.mat");

        %% remove root mean square values
        G3(:,5) = [];
```

```
G4(:,5) = [];
G5(:,5) = [];
G6(:,5) = [];

%% combine all grades data
data = [G3;G4;G5;G6];

%% split train data
[m,~] = size(data);
p = 0.75; % percentage of total data becomes train data
idx=randperm(m);
train_data = data(idx(1:round(p*m)),:);
rest_data = data(idx(round(p*m)+1:end),:);

%% split validation data and test data
[m,~] = size(rest_data);
p = 0.5; % percentage of total data becomes test data
idx=randperm(m);
val_data = rest_data(idx(1:round(p*m)),:);
test_data = rest_data(idx(round(p*m)+1:end),:);

%% convert train data and validation data into table
label_name = 'G_type';
train_table = array2table(train_data,'VariableNames',{'f1','f2','f3','f4',label_name});
train_table = convertvars(train_table,label_name,'categorical'); % label categorical col

%% convert validation data into table
val_table = array2table(val_data,'VariableNames',{'f1','f2','f3','f4',label_name});
val_table = convertvars(val_table,label_name,'categorical'); % label categorical col

%% conver test data into table
test_table = array2table(test_data,'VariableNames',{'f1','f2','f3','f4',label_name});
test_table = convertvars(test_table,label_name,'categorical'); % label categorical col

head(train_table)

classNames = categories(train_table{:,label_name})
%% define training option parameters
numFeatures = size(train_table,2) - 1;
```

```
numClasses = numel(classNames);
miniBatchSize = 50; % iteration per epoch = number of training samples
%% (1000 in my case) / miniBatchSize = 20
maxEpochs = 300; % iterations = iteration per epoch * maxEpochs

%% define all layers in the neural network
layers = [
featureInputLayer(numFeatures,'Normalization',"zscore")
fullyConnectedLayer(32) % 1st stack of layers
batchNormalizationLayer
reluLayer % activiation layer
fullyConnectedLayer(64) % 2nd stack of layers
batchNormalizationLayer
reluLayer % activiation layer
fullyConnectedLayer(128) % 3rd stack of layers
batchNormalizationLayer
reluLayer % activiation layer
fullyConnectedLayer(64) % 4th stack of layers
batchNormalizationLayer
reluLayer % activiation layer
fullyConnectedLayer(32) % 5th stack of layers
batchNormalizationLayer
reluLayer % activiation layer
fullyConnectedLayer(numClasses) % output layers
softmaxLayer
classificationLayer]

%% define training options
options = trainingOptions('adam','MiniBatchSize',miniBatchSize,'InitialLearnRate',
0.001,'MaxEpochs',maxEpochs,'Shuffle','every-epoch','ValidationData',
val_table,'Plots','training-progress','Verbose',false)

%% construct neural network and start to train
road_net = trainNetwork(train_table,label_name,layers,options);

%% Test the network
y_pred = classify(road_net,test_table(:,1:end-1),'MiniBatchSize',miniBatchSize);
y_true = test_table{:,label_name};
acc = sum(y_pred == y_true) / numel(y_true)
```

```
%% plot a confusion matrix to visualize the results
figure
confusionchart(y_true,y_pred)
%% save the trained network for further use
save('road_net_mode.mat',"road_net");
```

VITA AUCTORIS

Zhixian Yin was born in 1998 in Yangzhou, Jiangsu, China. She graduated from Taizhou Secondary High School in 2013. From there she pursued her undergraduate study with a B.Sc. in Mechanical Engineering with Automotive Option at the University of Windsor. She obtained her degree With Distinction and decided to push forward her education to a higher level. She is presently a candidate for the Master's degree in Applied Science at University of Windsor and is expected to graduate in August, 2022.

Education– Master of Applied Science-Mechanical student at University of Windsor, September 2018- present. Thesis title: Effects on Vehicle Ride Comfort of an Adaptive Suspension System Using Neural Networks Bachelor of Science (August 2018) in Mechanical Engineering, University of Windsor, Ontario, Canada.

Academic Employment– Graduate Teaching Assistant, Department of Mechanical, Automotive, & Materials Engineering, University of Windsor, 2018-2020. Responsibilities include: assisting professors with undergraduate courses, grading examinations or papers, and tutoring.

Employment– Vehicle Dynamics Engineer, Stellantis, April 2022 - present.