Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Applying Residual Policy Reinforcement Learning To Optimise PID-Control In Suspension Control

Andrew Hynes
[redacted]

Supervisor: Dr. Ivana Dusparic

April 30, 2020

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: 

Date: 30/04/2020

Abstract

Suspension control systems in cars are a vital component in modern vehicles tasked with enhancing ride comfort for passengers and protecting occupants from the impact of road disturbances. In order to maximise the performance of these systems, modern cars make use of a variety of different types of suspension. One popular type used in many cars is known as semi-active suspension and allows for the suspension systems to dynamically change to match road disturbances. Specifically, these systems dynamically alter a value known as the damping rate, which is responsible for controlling the dampers in a car. By doing this the vehicle can dissipate energy stored in the suspension system's spring, which in turn improves ride comfort.

However, these systems rely on system controllers to monitor and dictate the damping rate. One of the most common types of controller for this purpose is known as a Proportional Integral Derivative (PID) controller. Whilst these controllers have a number of benefits associated with them there are also a number of drawbacks, including the linear nature of the controllers, the difficulties with tuning the controller's parameters, and the sensitivity of the controllers to changes in the environment.

On the other hand, many of these weaknesses are the strengths of reinforcement learning techniques which are capable of dealing with non-linearities and are self-training and highly adaptable. In recent years, a lot of research has gone into combining reinforcement learning with PID-controllers in order to benefit from each system's strengths.

In this research, a novel approach is taken to do this, by directly combining a PID-controller with a Deep Deterministic Policy Gradient (DDPG) reinforcement learning agent, through a technique known as Residual Policy Reinforcement Learning. This approach involves summing both policies together in order to create a final policy which exhibits the benefits of both systems.

This approach is evaluated on a quarter car suspension system model provided by ZF Friedrichshafen AG. The reinforcement learning agent is trained using the DDPG algorithm and is deployed, with a PID-controller as a base policy, on a variety of road disturbances. The results show that the algorithm is capable of improving the suspension control of a car, enhancing the performance of a PID-controller, and adapting to environmental changes. However, it is also shown that the algorithm's performance is highly reliant on the performance of the base policy.

Acknowledgements

My deep gratitude goes to my fellow students and professors from Trinity College Dublin. In particular, I would like to thank my supervisor, Dr. Ivana Dusparic for her guidance and advice throughout the year. In addition, I would like to thank Dr. Elena Sapozhnikova, Dr. Tobias Pobandt and the entire team at ZF Friedrichshafen AG for their efforts in supporting me throughout this year. Finally, I would also like to extend my sincerest appreciation to my friends and family who have encouraged me throughout my education and without whom I would not be here.

Contents

List of Figures

List of Tables

1 Introduction

Suspension control systems are an integral part of modern car systems, with ride comfort and passenger satisfaction being directly linked to the performance of these systems. As such, there has been a lot of research focused on improving and maximising their performance. In order to do this, systems need to be developed that are capable of reducing the chassis acceleration, the chassis velocity and the chassis position when a vehicle impacts a road disturbance i.e. a bump. The chassis acceleration in particular plays a key role in ride comfort, with chassis velocity and position being directly related to it. As such, a variety of different approaches have been proposed to achieve this with one of the most popular systems being known as semi-active suspension systems.

These semi-active suspension systems work by dynamically altering the characteristics of the suspension system in order to maximise its performance. Specifically, semi-active suspension systems work by altering a parameter known as the damping rate. This parameter dictates the level of damping the system's damper experiences. These dampers are responsible for dissipating the energy absorbed in the system's spring following road disturbances and as such play a key role in maximising ride comfort for passengers.

However, in order for these systems to be effective, a system controller is needed to provide continuous input to the damper to dictate how it should behave. A very popular controller for this purpose is a Proportional, Integral, Derivative (PID) controller. These controllers operate by reducing the error between a target value and current value and are incredibly popular due to their cost, simplicity, reliability and ability to output continuous actions. As such, they are deployed across a wide variety of systems including temperature control and, of course, suspension control.

However, whilst these controllers are incredibly popular, they do have their drawbacks. The first problem associated with the controller relates to its linear nature which can make it perform poorly in complex non-linear systems. As well as this, the controller has a number of parameters associated with it, known as gains, and these parameters are known to be incredibly tedious to tune. Whilst there exists a wide variety of tuning techniques including the Ziegler and Nichols method [23] and Chien–Hrones–Reswick method [24], these methods rarely agree on the gain values, usually outputting different values causing the

controllers to behave slightly differently. This indicates that finding the set of parameters with the optimum performance for a PID is not trivial and as a result PID-controllers rarely operate at their maximum. As well as this, these parameters are highly specific to environments and as such even small deviations in the characteristics of an environment can result in a significant drop in performance. These changes in environmental characteristics can relate to incredibly minor fluctuations in temperature or even wear and tear. As such, even PID-controllers tuned in simulations can exhibit dramatically different performances once they are deployed to the real world. As a result, a lot of effort has been spent trying to identify online or self-tuning methods to help PID-controllers to adapt during their life cycle.

In recent years, a lot of research has also focused on replacing or combining PID-controllers with reinforcement learning in an attempt to overcome and compensate for the problems associated with the PID-controller. This focus is due in part to the growth of reinforcement learning techniques. More recently, reinforcement learning has been applied to a variety of different applications ranging from playing Atari games [25] to the control of Unmanned Aerial Vehicles (UAVs) [26]. This has been possible due to the unique set of characteristics associated with reinforcement learning algorithms which include the ability to deal with non-linearities, the ability to self-train and the ability to adapt to changes in the environment. This is particularly true for deep reinforcement learning approaches. These characteristics directly contrast the PID-controller's weaknesses and as such a number of efforts have been made to replace or enhance PID-control with reinforcement learning techniques.

One such attempt was shown in [1] where attempts were made to replace a PID-controller in a suspension control system completely with the implementation of a reinforcement learning algorithm. However, it was found during this work that the algorithm proved too unstable and struggled to fully grasp the task. This was believed to be due to the inconsistencies experienced during the learning phase and highlights the fact that this is not a trivial task.

In an attempt to remove these inconsistencies, and to allow reinforcement learning to be applied to PID-control, this work proposes the use of the Residual Policy Reinforcement Learning algorithm [9] to combine a PID-controller together with a deep reinforcement learning agent trained using the Deep Deterministic Policy Gradient algorithm [27]. This algorithm works by summing the output of both the PID-controller and the reinforcement learning agent together to form a final policy, which it is hoped can outperform either approach individually. This policy or strategy essentially describes the behaviour of the control system and it is believed that by combining these two individual policies together, through Residual Policy Reinforcement Learning, the final system can benefit from the consistency provided by the PID-controller, which will help the reinforcement learning agent to learn and exhibit the adaptability, non-linearity and self-training ability that is a hallmark of reinforcement learning techniques.

This is, to the best of the author's knowledge, the first time Residual Policy Reinforcement Learning will be applied to enhance the performance of a PID-controller, and it is hoped that by doing so the performance of PID-controllers and suspension control systems can be improved.

It is also hoped that by successfully improving the PID-controller's performance in suspension control, the techniques used can be applied to other applications involving PID-controllers in the future.

1.1 Thesis Aims and Objectives

The main aim of this thesis is to improve the overall performance of suspension control in vehicles by enhancing a PID-controller's ability to reduce the chassis acceleration, speed and movement when a road disturbance is encountered and hence improve ride comfort. It is believed that this is possible by combining PID-control with reinforcement learning, through Residual Policy Reinforcement Learning, in order to maximise the benefits of both systems. This thesis argues that the strengths and weaknesses of both these systems independently complement each other, and that by combining them through Residual Policy Reinforcement Learning a more effective control system can be developed. This thesis analyses whether this is indeed the case, and presents the design and implementation for such a system. It also provides an evaluation of such a system on a quarter car suspension system model provided by ZF Friedrichshafen AG.

1.2 Thesis Assumptions

In designing and evaluating the Residual Policy Reinforcement Learning implementation, a number of assumptions are made which simplifies the task and limits the scope of the thesis.

The main assumption involves the role of suspension control in vehicles. This thesis limits this role to only providing passenger comfort, however, in truth, these systems also play key roles in road handling, traction and steering stability. In this thesis these roles are ignored and this is reflected in the simulation used. This simulation only monitors vertical chassis acceleration, velocity and movement along a 2-D plane. As such, no consideration is given to the system's ability to deal with other tasks.

Another assumption is in relation to the factors that affect the suspension system of cars. The simulation used provides a simplified version of the impact of road disturbances and damping rate on a vehicle, and as such neglects a number of aspects which may affect this. This includes vehicle driving speed, which is not considered when calculating the chassis acceleration. However, intuitively, this is an important factor in determining ride comfort.

In addition to this, the simulation only allows an agent to input a damping value every 35ms, which is done to match the mechanical constraints of a real system, which only allows damping rate changes approximately every 35ms. However, in the simulation used this is done by only allowing the algorithm to input a damping value every 35ms, even if this input is the same. This means that the simulation has a harsher constraint than a real world system as even actions that are the same can only be taken once every 35ms. As a result, it is possible for a disturbance to occur during one of these periods and the algorithm will not have the ability to change the damping rate until the next 35ms period. This is the case even if the policy has not changed its output over the previous 35ms.

Finally, this thesis also assumes that road surfaces are completely smooth and ignores the small bumps and rough surfaces of a typical road. As a result, the agent's ability to deal with the typical random noise which may be found whilst driving on a normal road surface is not tested.

1.3 Thesis Contribution

This thesis identifies and motivates the need for combining PID-controllers with deep reinforcement learning algorithms through Residual Policy Reinforcement Learning. It attempts to illustrate the applicability of such a control system in the context of suspension control. It presents the challenges associated with using PID-controllers in system control and proposes a solution aimed at overcoming these problems with the use of deep reinforcement learning techniques. The main contribution of this thesis is the design, implementation, application and evaluation of a Residual Policy Reinforcement Learning algorithm aimed at improving suspension control in a car system. This technique allows for the development of a complementary system which combines the strengths of both PID-controllers and deep reinforcement learning to create a highly adaptive, consistent and effective control system. This control system is capable of learning strategies to enhance the performance of the PID-controller on a variety of different road disturbances and is capable of adapting to a variety of environmental changes. This thesis evaluates this model on a quarter car suspension system simulation provided by ZF Friedrichshafen AG and shows the potential of such an algorithm to enhance PID-control in suspension control.

1.4 Document Structure

This document is structured as follows. Chapter 2 provides the background to this research, providing an insight into reinforcement learning, PID-control and suspension systems. Chapter 3 describes the literature including some online self-tuning PID methods, reinforcement learning tuning methods and also highlights attempts to replace

PID-controllers with reinforcement learning. This section also describes previous work whereby Residual Policy Reinforcement Learning was used to optimize the performance of a Proportional (P) controller, which is a controller very closely linked with the PID-controller used in this study. Chapter 4 outlines the design used in this particular thesis, as well as providing information on the design of the task as a RL problem, the design of the algorithm itself and a number of additional design features used. Chapter 5 describes how these designs were implemented, including the tools used to do so. Chapter 6 presents the results and evaluations of this thesis on a variety of different road disturbances using a quarter car suspension system. Finally, chapter 7 provides a summary and conclusion of the work done, and provides direction for any future work.

2 Background

This chapter provides an insight into the techniques and technologies used in this thesis. Section 2.1 describes the basics of reinforcement learning. Section 2.2 introduces deep learning and describes the different components used to make it effective. Section 2.3 describes deep reinforcement learning which combines deep learning and reinforcement learning. Section 2.4 explains control systems, whilst section 2.5 describes suspension systems and portrays it as a control system. Finally, section 2.6 explains PID-controllers and how they operate.

2.1 Reinforcement Learning

Reinforcement learning (RL) represents a family of artificially intelligent techniques where an agent attempts to maximise its reward in an environment by learning how to act optimally in this environment through trial and error [28]. In order to do this the agent interacts with an environment through actions, which it makes based on the current position of the environment. This position is described to the agent via a state vector. The agent then receives a reward signal which describes how good a particular action was in the particular state, and the process begins again. This interaction can be seen in Figure 2.1.

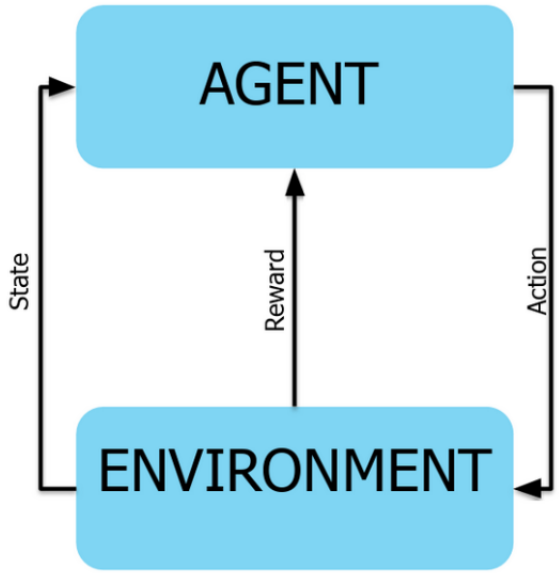

Figure 2.1: RL agent's interaction with environment [1]

Reinforcement learning is formalized as a type of Markov Decision process [29], and therefore is defined by;

- T – timestep
- S – set of states such that $s \in S$
- A – set of actions such that $a \in A$
- $R(s_t, a_t, s_{t+1})$ – reward function for taking action a_t, in state s_t and transitioning to the new state s_{t+1}
- $T(s_t, a_t, s_{t+1})$ – transition function which described the probability of transitioning to state s_{t+1} from state s_t having taken action a_t

In truth, RL is an incompletely-known Markov Decision Process and therefore does not know the transition function or reward function. Instead, reinforcement learning algorithms attempt to learn how best to behave in an environment in order to maximise their returns from that environment. This behaviour is defined by what is known as a policy in reinforcement learning. The policy essentially describes the strategy of an agent.

There are two main groups of reinforcement learning techniques. The first group is known as model based techniques and essentially involves the algorithm developing a model of the environment by taking actions and observing the feedback which includes the next state and the reward. However, these algorithms are of limited use due to the difficulty in modelling complex environments.

The alternative techniques are known as model free algorithms and do not require the

development of a model. Instead these algorithms try to directly learn the correct behaviour required to maximise reward. This thesis focuses on model free algorithms only.

Within model free algorithms, there are two subgroups; value based and policy based algorithms.

Value based algorithms work by trying to optimise value function approximations. These value functions usually try to learn Q-values which describe how good a particular action is in a particular state. These Q-value approximations can then be used to optimise the performance of an agent's policy. For example, if the agent's policy is to always pick the action with the highest predicted Q-value, then by optimising the accuracy of the Q-value predictions, the policy's performance can also be optimised.

In contrast, policy based algorithms work by trying to optimise a policy itself. In other words, this group of algorithms attempt to optimise the agents strategy as opposed to a value function which can be used to aid the strategy.

Regardless of which model is used the agent's ability to learn is reliant on the presence of a reward signal. The reward signal is returned to the agent from the environment and describes how good a particular action is within a particular state. Based on this signal the agent can learn what actions are good in what states.

2.1.1 Q-Learning

One of the most popular reinforcement learning algorithms, known as Q-learning, was created in [30]. This algorithm was first used in tabular form whereby a table was used to describe the Q-value associated with particular states and actions, as can be seen in Figure 2.2

	Action 1	Action 2
State 1	0	5
State 2	0	5
State 3	0	5
State 4	20	0

Figure 2.2: Tabular Q-learning [2]

The algorithm works by assigning Q-values for each state-action pair. In order to do this the algorithm randomly chooses actions in a particular state. This stage of learning is known as the exploration stage and is where the agent explores the environment, by taking random

actions, and learns about the Q-values of each state-action pair. In order to do this, the algorithm tries to apply temporal difference learning, which was first proposed in [31] and represents an attempt to incorporate long term rewards into the reward system. This results in the Q-learning update of:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[R_t + \gamma max_{a\epsilon A}Q(s_{t+1}, a_{t+1})] \tag{2.1}$$

where:
$Q(s_t, a_t)$ = the current prediction for the current state s and action a
γ = discount factor
α = learning rate **Note:** usually in the range (0,1)
R_t = actual reward returned from the environment
$max_{a\epsilon A}Q(s_{t+1}, a_{t+1})$ = the maximum estimated reward from the taking the best action in the next state

This equation states that the current reward is based on a combination of the immediate reward from the environment and the discounted future reward associated with the best action in the next state. The discount factor here reduces the importance of the future reward slightly, which conveys the priority of the immediate reward to the agent.

Following the exploration stage the algorithm enters into what is known as the exploitation stage, where the agent then tries to take the optimal actions based on what it has learnt during exploration. In the case of the Q-learning algorithm this involves choosing the action with the highest Q-value, considering the state.

As this algorithm works by optimising a value prediction, it is a value based method.

This algorithm can be very effective in discrete-state, discrete-action environments i.e environments where there are a limited number of actions and states, and it has been shown to have guaranteed convergence to the optimal policy in [32]. However, for this to happen, all actions must be repeatedly sampled in all states and the Q-values must be represented discretely.

2.2 Deep Learning

Deep learning is a subset of machine learning which sits in the wider paradigm of artificial intelligence. Deep learning makes use of highly complex computational models known as neural networks which draw heavily from our understanding of human biology, statistics and applied maths. Deep learning, in particular, refers to those neural network models with many

layers of neurons. The history of this technology goes back to the 1940's and can be seen in section A1.1.

Deep learning uses these neural networks to achieve great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones [33]. In other words, deep learning works by breaking down complex tasks into simpler tasks. For example, in image classification, one layer of the network may be responsible for identifying lines in an image, whilst another may be responsible for identifying contours. The key aspect of deep learning is that these layers aren't designed by humans, but instead are learnt during training through the use of learning algorithms such as the Backpropagation algorithm which will be discussed in Section 2.2.2.

This allows deep learning models to deal with highly complex tasks such as image classification and audio recognition. This field has grown considerably in recent years due to more powerful computers, larger data sets and better techniques [33].

2.2.1 Deep Neural Networks

Neural networks are a group of algorithms designed to emulate the human brain. In particular, neural networks attempt to mimic the behaviour and structures of the neurons in the brain. As a result, neural networks are composed of a number of layers of neurons, which are essentially processing elements, with synaptic input connections and a single output. These neurons are connected to each other via a series of weighted connections, which are updated during learning. Most neurons also have a bias associated with them, which can also be updated during training. This results in an input equation like so;

$$\Sigma = w_1 x_1 + w_2 x_2 ..... w_n x_n + b \tag{2.2}$$

where:
w_i = weight i
x_i = input i
b = bias
n = the number of weights passing into the neuron

An activation function is then applied to this weighted sum of inputs in order to get the output, as can be seen in Figure 2.3.

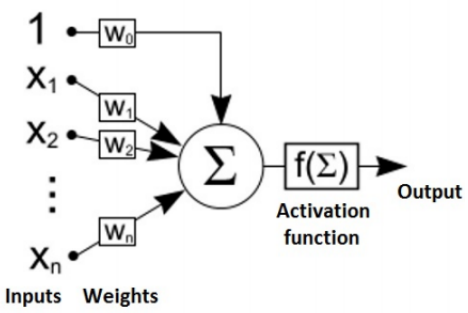

Figure 2.3: Neurons [3]

Models are generally composed of three types of layers; the input layers, the hidden layers and the output layers. As can be seen in Figure 2.4, the input layer is tasked with taking in the data, and the output layer is tasked with providing the final product. Each layer is used to compute a different feature of the data, for example, in an image classification network, a layer may be used to determine the presence of a contour. Due to this the number of hidden layers may vary as the task complexity increases and more layers are needed to analyse all features. There are no exact rules for calculating the number of hidden layers, however this is an area of on-going research and at present there are a number of design heuristics used [34].

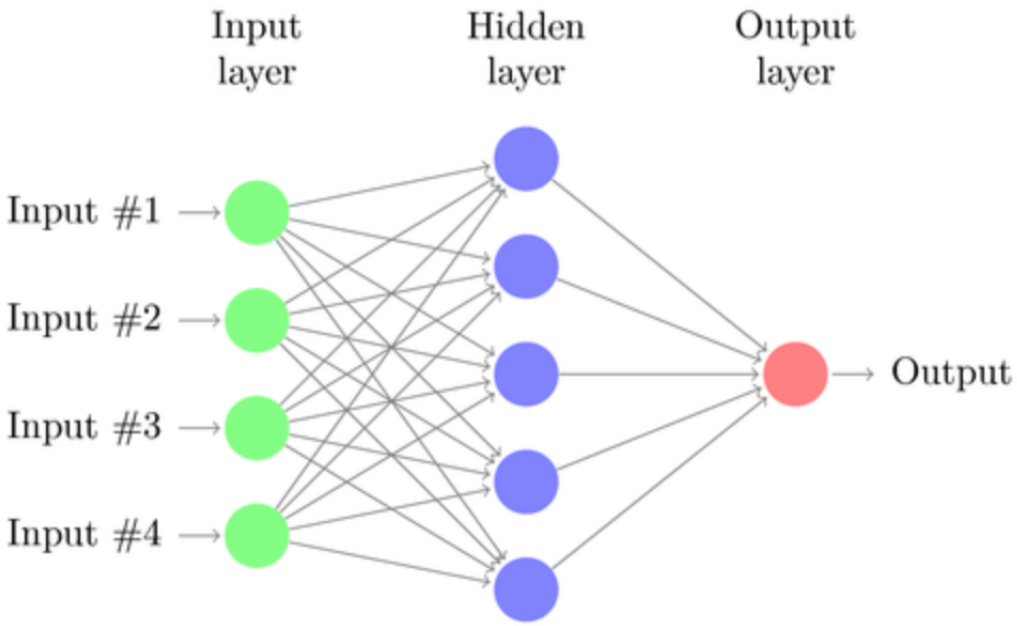

Figure 2.4: Neural Network Architecture [4]

These networks are usually trained to minimise error by using the Backpropagation algorithm which will be explained in Section 2.2.2. They also combine a number of features such as non-linear activation functions and optimisation algorithms to maximise performance. These features will be discussed in Sections 2.3.3 and 2.2.4 respectively.

2.2.2 Backpropagation

In order to train these models for function approximation an algorithm known as the Backpropagation algorithm was proposed in [35] to calculate the gradient of the loss with respect to each weight. It uses the chain rule to calculate each weight's contribution to the loss i.e. the error between the desired output and the actual output. In other words, it is essentially used to calculate the gradient of the loss function for each weight. From here gradient descent can be deployed to minimise the loss function as so;

$$w_{t+1}^{i} = w_{t}^{i} - \alpha \frac{\partial E}{\partial w^{i}} \tag{2.3}$$

where:
w = weight i
t = timestep
α = learning rate
$\frac{\partial E}{\partial w}$ = the gradient of the loss with respect to the weight i, this is calculated by the backpropagation algorithm

2.2.3 Activation Functions

According to [5], activation functions are functions used in neural networks to compute the weighted sum of income weights and biases and to decide whether a neuron can fire or not. They are also referred to as a threshold function as certain activation functions require the weighted sum of the inputs and biases to surpass a particular value for the neuron to fire. Activation functions are vital to neural network's success as these functions are what allow neural networks to cope with non-linearities. There are a wide variety of activation functions, some of which will be discussed here.

Linear Activation Function

The linear activation function is one of the simplest activation functions, simply outputting the weighted sum of the inputs. It is of the form;

$$output = weights \times inputs + bias \tag{2.4}$$

It can return any value in the range (-∞, ∞).

Hyperbolic Tangent Activation Function (TanH)

The TanH function is a zero centred function which squashes the weighted sum of inputs into the range (-1,1), as can be seen in Figure 2.5. It allows the neural network to deal with non-linearities and is of the form;

$$output = \frac{e^{\chi} - e^{-\chi}}{e^{\chi} + e^{-\chi}} \tag{2.5}$$

where:

$$\chi = inputs \times weights + bias \tag{2.6}$$

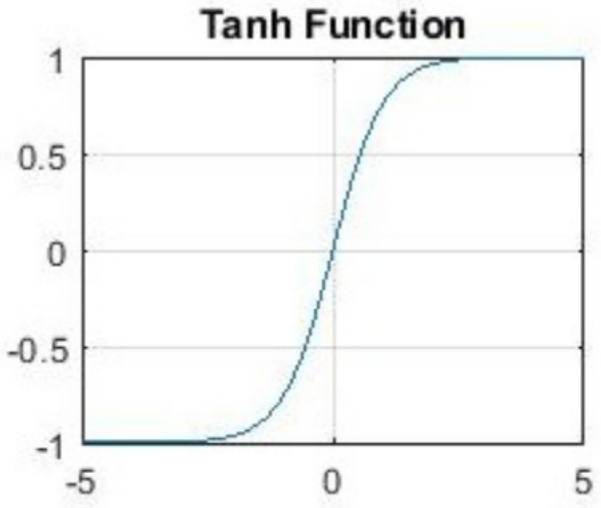

Figure 2.5: TanH Activation Function [5]

However, this activation function suffers from a problem known as the vanishing gradient problem. This problem occurs when applying the backpropagation algorithm to large neural networks with many layers. As the algorithm moves back through the network the gradients tend to get smaller and smaller, until they eventually vanish. As a result, the neurons in the early layers tend to learn much slower than those in the later layers, resulting in reduced performance.

Rectified Linear Unit Function (ReLU)

In order to overcome the vanishing gradient problem, [36] proposed the Rectified Linear Unit function, which has the form;

$$output = max(0, \chi) \tag{2.7}$$

where,

$$\chi = inputs \times weights + bias \tag{2.8}$$

As can be seen in Figure 2.6, the ReLU activation function has a slope of either 0 or 1. As a result, the gradient cannot progressively decrease which prevents the vanishing gradient problem. However, it is possible for neurons to suffer from the dying ReLU problem, which results in the neuron constantly outputting a zero value. This is caused by the zero slope produced by the ReLU function for negative numbers.

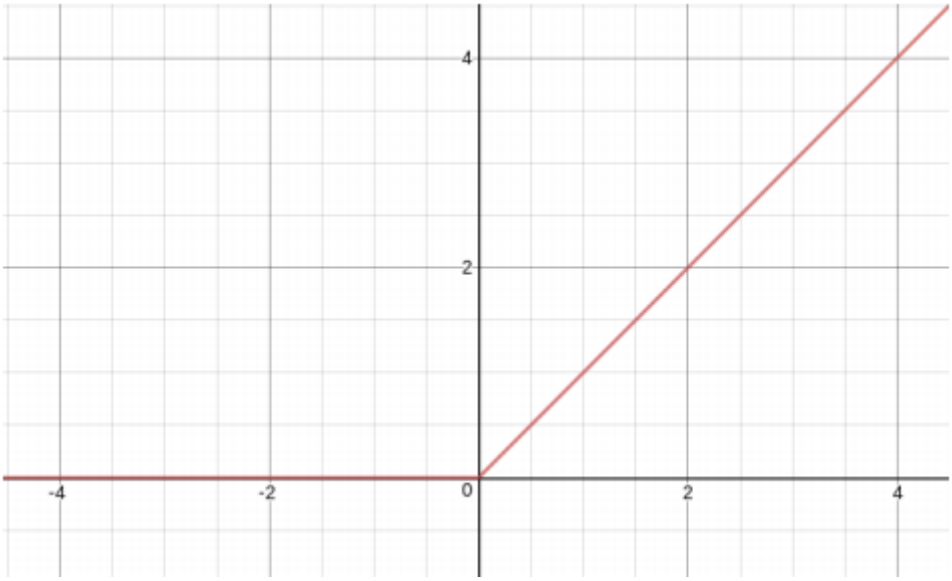

Figure 2.6: Rectified Linear Unit Function [6] **Note:** the red line indicates the ReLU function

2.2.4 Optimisation Algorithms

Optimisation algorithms are a group of algorithms which attempt to reduce the loss of a neural network by updating the neural network's parameters. Arguably, the most well-known optimisation algorithm is the standard gradient descent algorithm which works by iteratively taking steps towards the local minimum of a differentiable equation. The size of these steps is dictated by the gradient of the loss and the learning rate, which is a hyperparameter. Gradient descent is of the form;

$$\theta_{t+1} = \theta_t - \alpha.\nabla_\theta J(\theta) \tag{2.9}$$

where:

θ = parameters

α = learning rate
$\nabla_\theta J(\theta)$ = gradient of the loss with respect to the parameters

However, gradient descent does suffer from a number of drawbacks, one of these being the fact that the learning rate is fixed for all features in a neural network. This is a problem if features occur at a different frequency and therefore may need to be updated by different amounts e.g. features which occur half as often as other features should have slightly larger updates to ensure they are trained as efficiently. This also means choosing the correct learning rate can be very tedious and is the sole responsibility of the user. Therefore a number of other algorithms have been proposed to overcome this issue and to make use of adaptive learning rates, some of these are outlined now.

AdaGrad

[37] proposed an algorithm called AdaGrad. Intuitively this algorithm works by making smaller updates for frequently occurring parameters and larger updates for infrequent parameters. AdaGrad does this by taking the sum of squares of the gradient, with respect to each parameter, into account. It is of the form;

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t \tag{2.10}$$

where:
θ = parameters
α = learning rate
G_t = matrix of the sum of squares of the gradients
ϵ = smoothing term to avoid division by zero
g_t = matrix of the gradients of the loss

[38] found that this optimiser not only allowed for adaptive learning rates, but also proved to enhance the robustness compared to stochastic gradient descent. However, as the sum of squares of the gradient is in the denominator, overtime this grows infinitesimally large causing the learning rate to become infinitesimally small.

RMSprop

In order to overcome the decreasing learning rate problem associated with the AdaGrad algorithm, [39] proposed an algorithm known as RMSProp which also adapts the learning rate during training. This algorithm replaces the sum of squares of the gradients with the

exponentially decaying average, meaning the denominator should not get infinitesimally large. It is of the form;

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E(g^2)_t + \epsilon}}.g_t \tag{2.11}$$

where:
θ = parameters
α = learning rate
g = is the sum of squares of the gradients
ϵ = smoothing term to avoid division by zero
g_t = the gradient of the loss function

Adam

[40] proposed an algorithm known as Adam. As in RMSprop, Adam uses the exponentially decaying average of squared gradients. However, it also uses the exponentially decaying average of gradients, which is similar to how a technique known as momentum works. These two are calculated as follows;

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.13}$$

where:
m_t = the first moment(the mean)
v_t = the second moment(the uncentered variance)
g_t = past gradients
β_1 = hyperparameter for m_t
β_2 = hyperparameter for v_t

However, as these moments are initialised as vectors of zero's, they are biased towards zero. As such, these biases need to be corrected by using the following bias correction estimates;

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{2.14}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{2.15}$$

From here the update can then calculated as;

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.16}$$

The fact that Adam incorporates the bias correction estimates outlined above has allowed it to outperform RMSProp, as it can deal with the sparser gradients near the end of training according to [41].

2.2.5 Universal Approximation Theorem

One of the most common and basic neural networks is known as a feed forward network, which is essentially a non-cyclic neural network i.e. everything moves forward. One of the reasons for its popularity is its ability to approximate functions. In fact, [42] proved that these networks, with a linear output layer and at least one hidden layer, are capable of approximating any function, in a theorem known as the Universal Approximation Theorem. As such feed forward networks make great function approximators.

However, as highlighted in [43], whilst feed forward networks are capable of approximating any function in theory, the layers may need to be infeasibly large and may fail to generalize completely. Therefore, even though the Universal Approximation Theorem holds up in theory, it may not be always be achievable in practice.

2.3 Deep Reinforcement Learning

Over the last number of years there has been a lot of research aimed at combining the power of deep learning with the paradigm of reinforcement learning. To this end, there have been many new reinforcement learning algorithms created which harvest the function approximation power of neural networks to extend the use of reinforcement learning, some of these will be looked at in this section.

2.3.1 Deep Q-learning

The Q-learning algorithm described in Section 2.1.1 is a very effective algorithm, with guaranteed convergence to an optimal policy when deployed correctly. However, it is limited to discrete-state, discrete-action environments due to its tabular form. In order to overcome this issue, [7] proposed the use of deep neural networks to act as function approximators to provide predictions of the Q-values, as opposed to the use of a table to record the exact Q-values in an algorithm known as Deep Q-Networks (DQN). By doing this, DQN can operate in continuous-state, discrete-action environments i.e. environments where there is a large, potentially, infinite number of states, and a finite number of actions.

This was applied to Atari games as can be seen in Figure 2.7. The states, which are the screenshots of the Atari game, act as an input into a neural network, which then assigns a Q-value to each action. From here the action with the highest predicted Q-value is chosen.

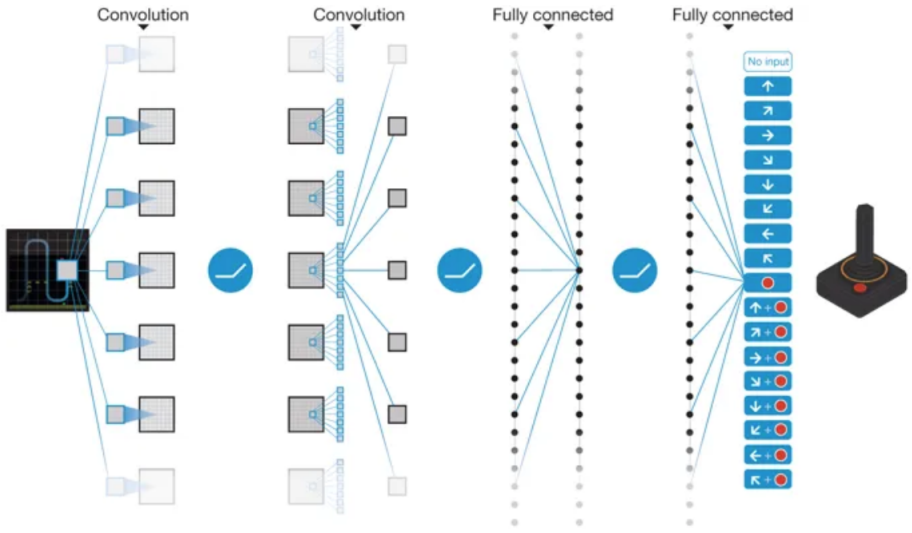

Figure 2.7: Deep Q-learning [7] **Note:** the use of convolutional layers to analyse the image of the screen

However, this algorithm was not trivial to create as the use of neural networks in reinforcement learning is known to be unstable and therefore the authors implemented two new features to help overcome this: experience replay and target networks.

Experience replay is the idea of storing the agents experience and reusing these experiences throughout training to continuously learn. An algorithm that uses experience replay is said to be an off-policy algorithm i.e. it is an algorithm that learns about its current policy or strategy, by using a different policy. In this case it is off-policy because it learns from past experiences which most likely came from an old strategy. This feature has three main benefits. Firstly, it is sample efficient i.e. the agent can learn multiple times from the same experience. Secondly, it improves learning as learning from consecutive samples can be inefficient due to the correlation between them. Therefore, randomly selecting samples from past experiences can help overcome these problems. Finally, due to the fact it is off-policy it improves the stability of learning in certain circumstances, as on-policy learning methods can experience large deviations and divergence in parameters as the next set of sample data is determined by the current set of parameters. However, it should be noted that this is a point of contention with certain on-policy algorithms showing improved stability over off-policy algorithms. As such, the use of experience replay can reduce divergence.

The implementation of experience replay results in two additional hyperparameters. The first hyperparameter relates to the size of the experience replay buffer i.e. the amount of

experiences to store. The second is the batch size hyperparameter, which outlines the number of experiences to sample from the replay buffer at every iteration.

The other feature, known as a target network, also contributes greatly to improving the stability of learning. This is because it helps reduce the fluctuations experienced when using temporal difference learning. As seen in section 2.1.1, temporal difference learning creates a target function based on the current reward and a prediction of the future reward. This future prediction is calculated using a function approximator and results in a technique known as bootstrapping i.e. using one function approximator to train another. This results in a target value of the form;

$$target\ value = R_t + \gamma max_{a \epsilon A} \hat{Q}(s_{t+1}, a_t, w_t) \tag{2.17}$$

where:
R_t = current reward
γ = discount factor
$\hat{Q}(s_{t+1}, a_t, w_t)$ = the Q-value approximation given the new state, s_{t+1}, the current action, a_t, and the current neural network parameters at time t, w_t

Using this function an agent can learn to consider the long term implications of taking a particular action. However, when used with neural networks, temporal difference learning has proved to be very unstable, mainly because the function uses bootstrapping. Bootstrapping is so problematic because the neural network being used to predict the future rewards is oftentimes updated at the same rate as the main function approximator that is being trained. In fact, in some implementations the neural network being used for predicting the future reward, is the same one that is being updated. This results in a lot of fluctuations and noise during training.

Target networks overcome this by using a far smaller update for the future reward function approximator than is used for the main Q-value function approximator. They are essentially identical copies of the main network, however, during learning these networks update by taking small updates from the main network. Usually, this is done by summing a large percentage of the target network's weights (usually approximately 99%) with a low percentage of the main network's weights (usually approximately 1%), resulting in an update equation for a target network as so;

$$\theta_{target\ network} = \tau \theta_{target\ network} + (1 - \tau)\theta_{main\ network} \tag{2.18}$$

where:
$\theta_{target\ network}$ = target network's parameters

τ = percentage of parameters the target network maintains
$\theta_{main\ network}$ = main network's parameters

This has the effect of updating the target network at about 1% the frequency of the main network. By doing this, a smoother learning curve can be created. Figure 2.8 shows the difference in noise between a normal network and a target network with reduced updates.

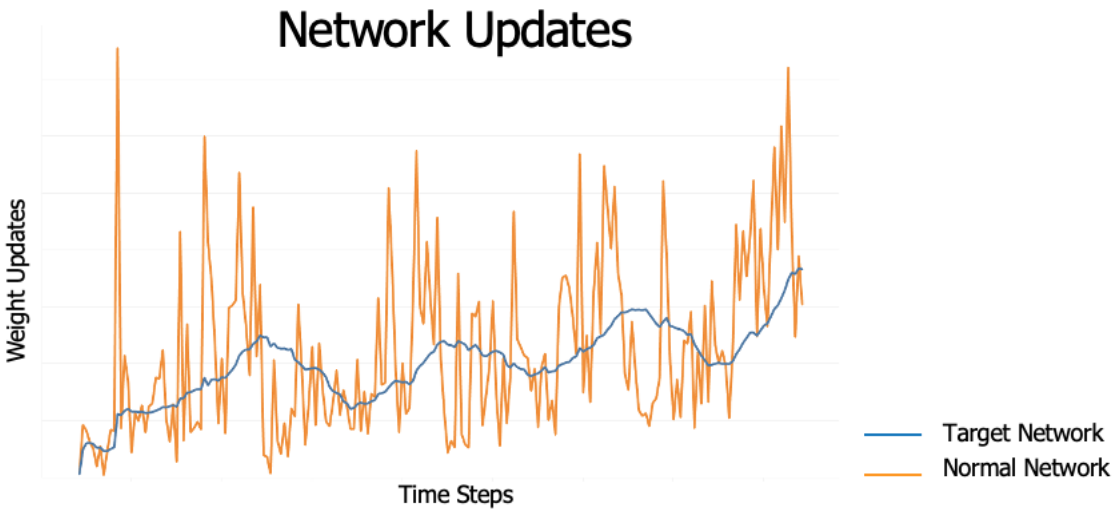

Figure 2.8: Deep Q-learning Target Networks [8] **Note:** the difference in noise between the learning of the normal network versus the target network with reduced updates

Again, target networks introduce another hyperparameter known as the target network parameter, τ. This outlines the ratio target networks should be updated by in comparison to the main networks. For example, a τ value of 0.01 corresponds to updating the target networks at a rate of approximately 1% of the main network.

By combining these two features the authors were able to create a stable and efficient learning algorithm capable of dealing with a variety of continuous-state, discrete-action environments.

2.3.2 Policy Gradients

Whist Deep Q-learning, described in Section 2.3.1, is a very powerful algorithm, it is incapable of dealing with highly complex continuous-action spaces, as it relies on the idea of assigning a value to every possible action. Instead, in order to overcome this, policy gradient methods were proposed in [44].

Policy gradient methods operate differently to the other algorithms we have seen, mainly because they directly attempt to optimise a policy by updating the policy's parameters, θ. As such they are policy based methods, as opposed to the value based methods we have

already seen, like DQN which works by optimising a value approximation. Essentially, the goal of policy gradient methods is to directly find the parameters, θ, which will maximise the reward. When using policy gradient methods alongside deep learning, θ is used to describe the parameters of the neural networks i.e. the weights and biases. Therefore updating the parameters means updating the weights and biases of the network, which can be done using gradient ascent which is of the form;

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.19}$$

where:
θ = policy's parameters
α = learning rate
$\nabla J(\theta_t)$ = gradient of the objective function at time t

Note: the use of the + as opposed to the - in this equation. This is due to the use of gradient ascent as opposed to gradient descent, as reinforcement learning is focused on maximising the reward, as opposed to minimising the loss.

The gradient of the objective function is equal to the gradient of the expected reward function and can be written as;

$$\nabla J(\theta) \ = \nabla \, \mathbb{E}[r(\tau)] \tag{2.20}$$

Therefore the gradient ascent equation can be rewritten as;

$$\theta_{t+1} = \theta_t + \alpha \nabla \, \mathbb{E}[r(\tau)] \tag{2.21}$$

In order to solve this function we can make use of the Policy Gradient Theorem presented in [44], which states that the derivative of the expected reward is the expectation of the product of the reward and the gradient of the log policy. This can be written as;

$$\nabla \, \mathbb{E}[r(\tau)] = \mathbb{E}_{\pi_\theta}[r(\tau) \nabla log \pi_\theta(\tau)] \tag{2.22}$$

where:

$$\pi_\theta(\tau) = \sum_{t=1}^{T} \pi_\theta(a_t|s_t) \tag{2.23}$$

This results in a final gradient update of the form;

$$\theta_{t+1} = \theta_t + \alpha \, \mathbb{E}_{\pi_\theta}[r(\tau) \sum_{t=1}^{T} \nabla log \pi_\theta(a_t|s_t)] \tag{2.24}$$

It is important to note here that the policy is a probabilistic distribution, and as such this must be accounted for. Therefore $\pi_\theta(a_t|s_t)$ is used to describe the probability of choosing action a_t given state s_t.

2.3.3 Actor-Critic Models

In order to apply the policy gradient methods described in section 2.3.2, it is necessary to make use of a family of algorithms known as actor-critic methods, which were first proposed in [45].

As can be seen from Figure 2.9 these models are composed of two sections, an actor section and a critic section. The actor section is responsible for outputting a probabilistic distribution of actions, from which the action with the highest probability will occur most often. Whilst the critic section is responsible for providing feedback on that action.

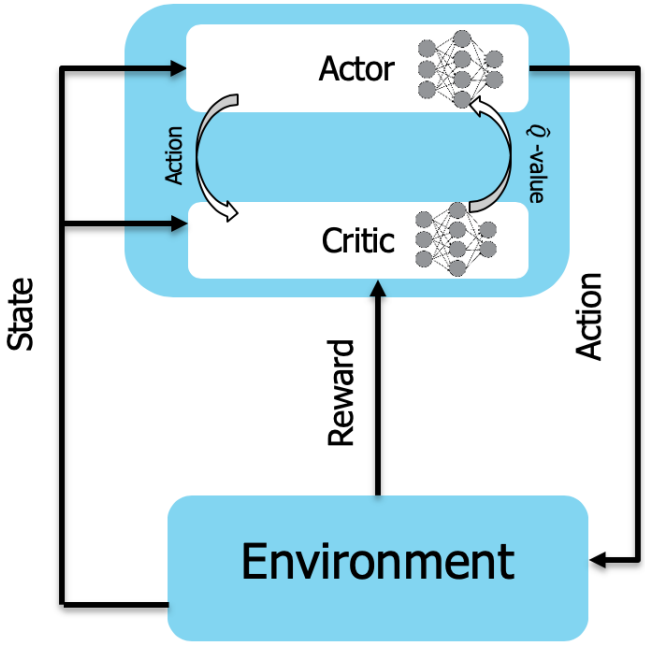

Figure 2.9: Actor Critic Architecture [1]

As can been seen in Figure 2.9 the critic receives the state and the action as inputs. The critic's task is to try to learn a value approximation for a state action pair i.e. the Q-value. This is done by using Backpropagation on the following loss function;

$$Loss = prediction - actual \tag{2.25}$$

This gives a critic update of the form;

$$J(\theta) = \frac{1}{2}(R_t + \gamma \hat{Q}(s_{t+1}, a_t) - \hat{Q}(s_t, a_t))^2 \tag{2.26}$$

where:
$J(\theta)$ = the objective function for the critic
R_t = the actual reward at time t
γ = the discount factor
$\hat{Q}(s_{t+1}, a_t)$ = the Q-value approximation for the next state s_{t+1} and current action a_t
$\hat{Q}(s_t, a_t)$ = the Q-value approximation for the current state s_t and action a_t

This is very similar to the update function used for the DQN algorithm in section 2.3.1, as the critic's role is to essentially learn an approximation of the Q-value for a state-action pair, just like in the DQN algorithm.

The actor network can now be updated using the Policy Gradient Theorem described in section 2.3.2. However, the difference here is the use of $\hat{Q}(s_t, a_t)$ instead of $r(\tau)$ as the goal is to update the actor based on the feedback from the critic, which acts as an approximation of the reward. This gives an equation of the form;

$$\theta_{t+1} = \theta_t + \alpha \, \mathbb{E}_{\pi_\theta}[\hat{Q}(s_t, a_t) \sum_{t=1}^{T} \nabla log \pi_\theta(a_t|s_t)] \tag{2.27}$$

This can then be used to train actor-critic models for use in continuous-state, continuous-action environments. An important point when using actor-critic models is understanding that the critic is only needed during training, and therefore can be discarded once training is completed.

2.3.4 Deep Deterministic Policy Gradient

However, actor-critic models alone can often struggle to converge. In order to overcome these difficulties, [21] proposed combining the Policy Gradient Theorem and actor-critic models, with the added features proposed in the DQN algorithm: experience replay and target networks, in order to create the Deep Deterministic Policy Gradient algorithm (DDPG).

The target networks in this case were implemented slightly differently, mainly because actor-critic models make use of two neural networks. As such both networks used a target network to help calculate the temporal difference error. This results in a slightly different target value update for the critic which takes the form:

$$\textit{target value} = R_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}, w_t) \tag{2.28}$$

where:
R_t = current reward
γ = discount factor
$\hat{Q}(s_{t+1}, a_{t+1}, w_t)$ = the Q-value approximation given the new state, s_{t+1}, the target network's new action, a_{t+1}, and the neural network parameters, w_t

As can be seen from this equation, the future prediction is based on a new action $\mathbf{a_{t+1}}$ as opposed to the current action, $\mathbf{a_t}$, used for updating the critic in section 2.3.3. This is because the DDPG algorithm makes use of the target networks to choose a new action based on the new state. However, just like in the DQN algorithm the critic's loss is the difference between the predicted Q-value and the actual Q-value, which is of the form:

$$J(\theta) = \frac{1}{2}(R_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t))^2 \tag{2.29}$$

The actor's update function, however, is slightly different. This is due to the fact that the update function described in equation 2.27 is based around stochastic policies, and therefore needs to incorporate probabilistic distributions. However, the DDPG algorithm is deterministic and therefore its updates are done using the equation;

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{Q}(s_t, a_t) \nabla \mu(s_t) \tag{2.30}$$

where:
$\nabla_\theta J$ = Distribution of actions
$\nabla \hat{Q}(s_t, a_t)$ = gradient of the critic
$\nabla \mu(s_t)$ = gradient of the deterministic policy

Note: the policy is now deterministic and this justifies the use of $\mu(s)$ instead of $\pi(a|s)$ and the removal of the expectation.

[27] proved this was the policy gradient of the deterministic policy.

However, the fact that this algorithm is deterministic creates another challenge in terms of exploration. Usually, when using a stochastic policy gradient method, the agent is able to explore the environment itself by taking a range of actions, as described by the stochastic policy. However, this is no longer possible with a deterministic policy and therefore a new exploration strategy needs to be developed. This is done by adding Gaussian noise to actions

at a probability of epsilon, where epsilon is a small probabilistic number e.g. 0.3. This means that 30% of the time the action taken will be random due to noise being added on to the model's output. This epsilon parameter is also known as the exploration rate, and is reduced at each time step by a value known as the exploration decay rate. This reduction takes the form;

$$\epsilon_{new} = \epsilon_{old}.K \tag{2.31}$$

where:
ϵ = exploration rate
K = decay rate

The decay rate is usually another small value in the range [0.9,1). This results in a very small and gradual reduction in exploration rate. It also creates two further hyperparameters: exploration rate and exploration decay rate.

2.3.5 Residual Policy Reinforcement Learning

However, whilst the DDPG is a very capable algorithm, it does not always converge, instead, in certain complex environments it suffers greatly from divergent problems. This is often attributed to the critic being unable to learn accurate predictions for the Q-value.

One algorithm that appears capable of reducing the likelihood of divergence is the Residual Policy Reinforcement Learning algorithm proposed in [9]. This algorithm works by combining a reinforcement learning policy with another policy. The reinforcement learning policy is known as the residual policy, whilst the other policy is known as the base policy. It takes the form of:

$$\pi_\theta(s) = \pi(s) + f_\theta(s) \tag{2.32}$$

where:
π_θ(s) = final policy
$\pi(s)$ = base policy
f_θ = residual policy

Depending on how well the base policy performs, the residual policy learning algorithm can be seen as either a corrective term used to enhance performance of the base policy, or it may become a substantial component in the overall policy and may just use the base policy for hints to guide its exploration.

Using this equation it can be seen that the gradient of the policy, with respect to the

parameters θ, depends on the residual policy as opposed to the base policy;

$$\nabla_\theta \pi_\theta(s) = \nabla_\theta f_\theta(s) \tag{2.33}$$

Therefore, it is possible to use policy gradient methods alongside residual policy reinforcement learning.

In [9], the authors used the DDPG algorithm as their residual reinforcement learning policy alongside a base policy known as the reactive hook and were able to show that it was far more stable than alternative RL methods. They were also able to show that the Residual Policy Reinforcement Learning algorithm could converge in environments were the DDPG algorithm, which was deployed with a sparse reward algorithm known as Hindsight Experience Replay (HER) [46], could not, as can be seen in Figure 2.10.

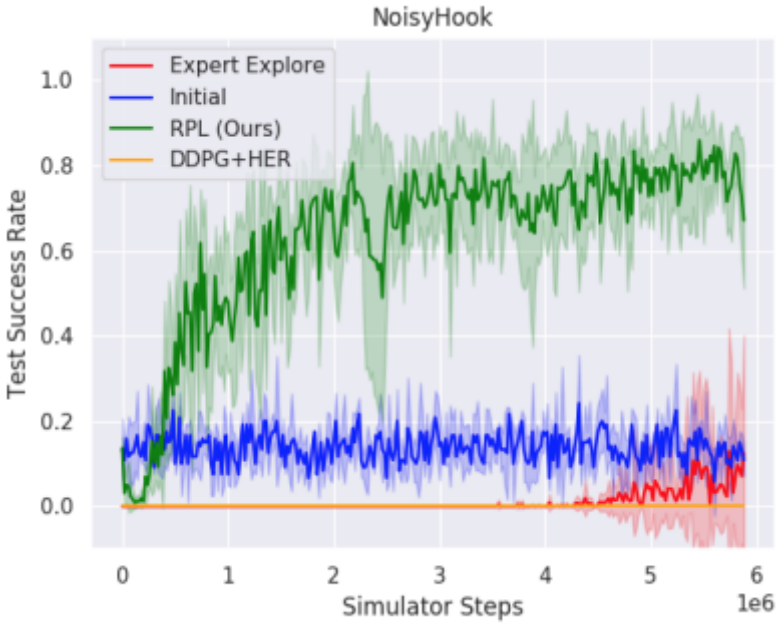

Figure 2.10: Performance of Residual Policy Reinforcement Learning Algorithm [9] **Note:** The yellow line indicating the DDPG+HER implementation failing

This suggests that the Residual Policy Reinforcement Learning algorithm was able to overcome the instability which led to the DDPG+HER implementation failing. The authors believe that this was due to the Residual Policy Reinforcement Learning algorithm's enhanced ability to deal with long horizons and sparse rewards. In reinforcement learning, a horizon refers to a future point from the current timestep, beyond which the reward is irrelevant. For long horizons i.e. horizons which incorporate many future timesteps, neural networks can struggle to learn accurate Q-value approximations as shown in [47]. However, by using a base policy the trajectory can stay relatively consistent, meaning that the target Q-values are more stable during training and hence easier to learn. Essentially the base

policies provide a level of consistency which aids the critic in learning the value approximation. This in turn helps reduce the likelihood of divergence.

2.4 Control Systems

Control systems are systems which provide a desired response from a process or plant by controlling an output signal, commonly known as a manipulated variable, which interacts with the process or plant to attempt to reach a target output value. There are two main families of control systems: open loop control systems and closed loop control systems.

Open loop systems are systems which do not feedback the output into the input, as can be seen in Figure 2.11.

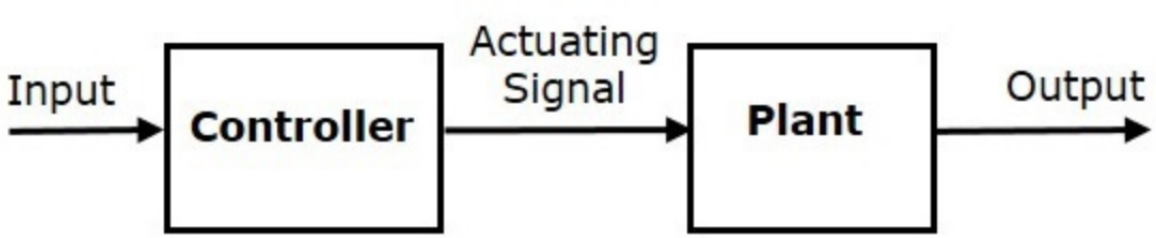

Figure 2.11: Open Loop system [10]

Closed loop systems on the other hand work by feeding the output back into the control unit, as can be seen in Figure 2.12. Closed loop systems will be the systems focused on in this thesis.

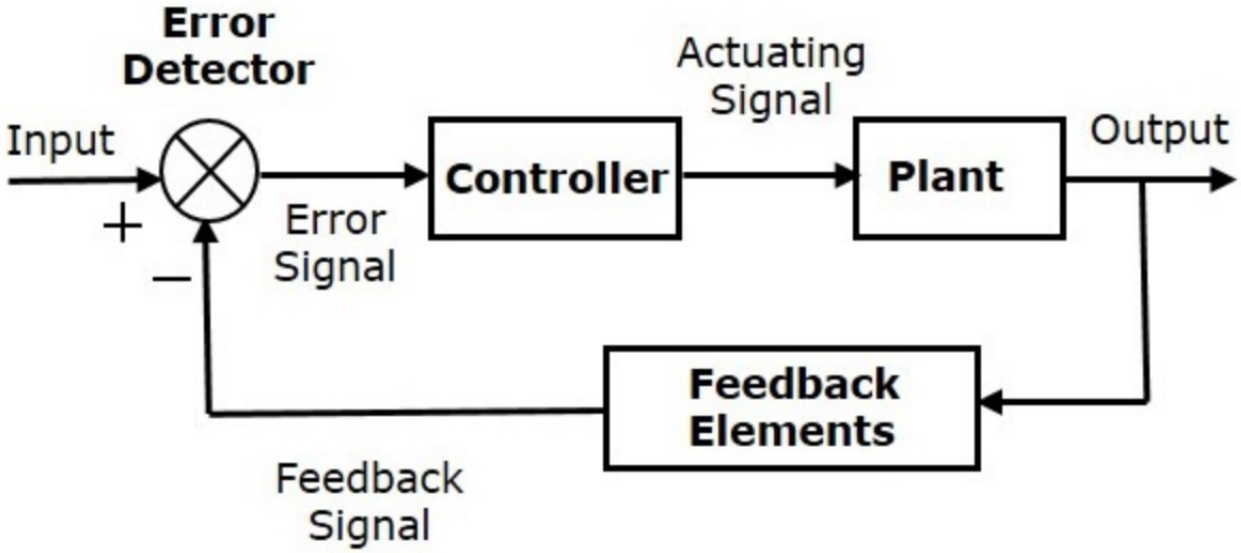

Figure 2.12: Closed Loop system [10]

The job of the controller therefore is to output a signal i.e. the manipulated variable, to interact with the system, in Figure 2.12 this is the plant, in order to achieve the optimum output.

2.5 Suspension Control

In this thesis we will focus on suspension control systems for cars. These suspension systems are composed of a tyre, a chassis, springs and dampers as can be seen in Figure 2.13. The suspension system's main goal is to reduce the impact of road disturbances felt by passengers in the chassis.

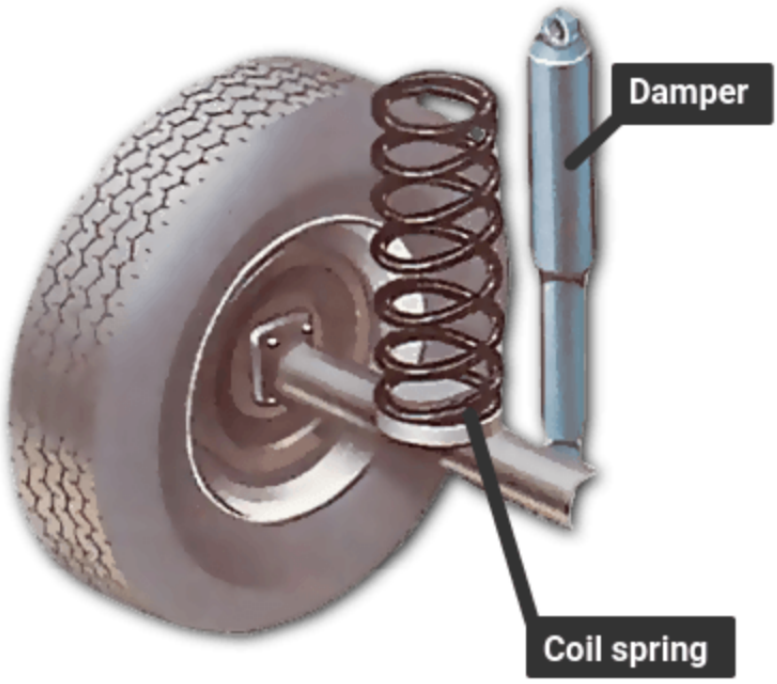

Figure 2.13: Suspension system [11]

When a car encounters a bump the spring is compressed and stores the energy from the road. In order to release this energy the spring oscillates, however, this can create discomfort for the passengers and therefore it is the role of the damper to control this oscillation. In order to do this the damper, which is connected in parallel to the spring, converts the kinetic energy of a spring into heat energy, which can be dissipated by hydraulic fluid.

Dampers are often designed with a chamber filled with hydraulic fluid, and a piston with orifices in it, as can be seen in Figure 2.14. When a bump is encountered on the road the spring will start to oscillate, which in turn causes the damper to oscillate. However, as the damper oscillates a huge amount of pressure is needed to push the hydraulic fluid through the small orifices in the piston. This slows the piston down, and as a result also reduces the

frequency of spring oscillations, hence, improving comfort. The size of these orifices dictate the force required to push the piston down, and hence dictate the amount of damping, this is known as the damping rate.

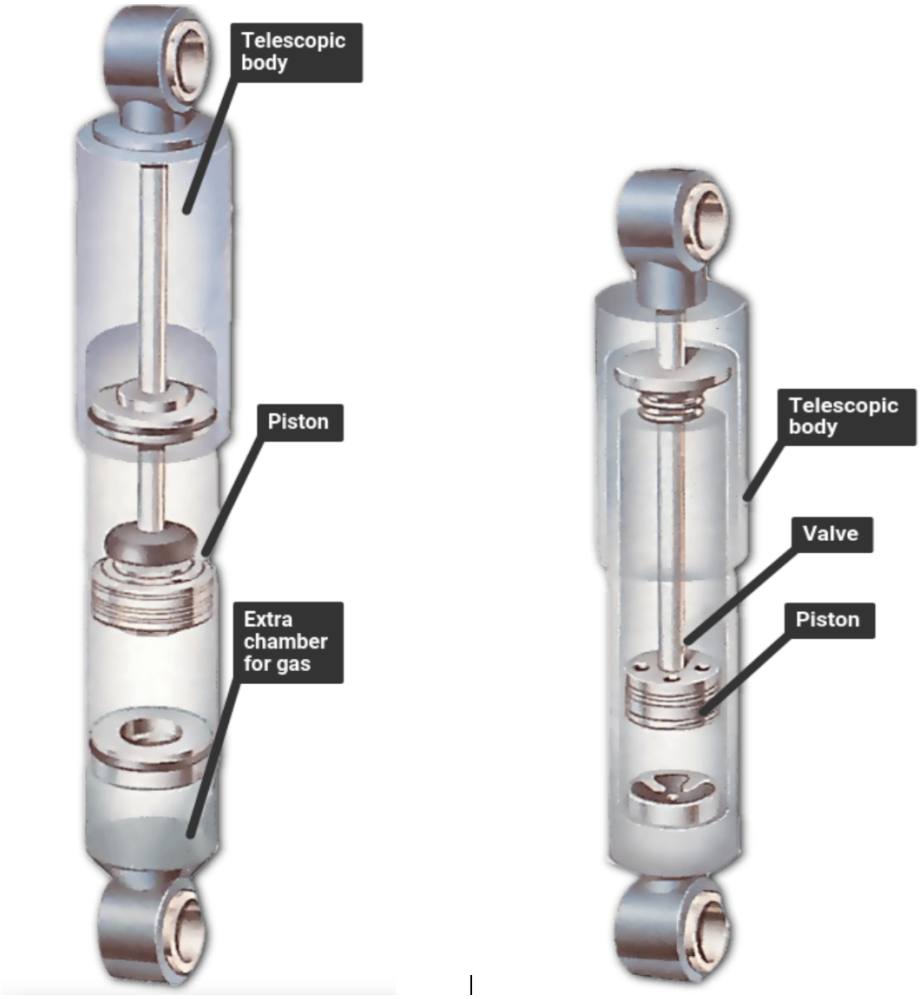

Figure 2.14: Dampers [11] **Note:** the difference as the piston is compressed

However, it is sometimes desirable to change the orifice size in order to vary the damping rate. For example, when a sharp bump is encountered the damper should have a very low damping rate and therefore a very large orifice, but it is then desirable to reduce the orifice size to dissipate the energy in the spring. In order to do this, [48] proposed a damper known as a semi-active damper, which allows the orifices to change size.

In order to control the size of these orifices a control system is used, which can monitor the acceleration and velocity of oscillation in the chassis and wheel, and manipulate the size of the orifice to increase or decrease damping. This creates a closed loop control system and as such a controller is needed to govern it. There has been a wide variety of control techniques proposed to optimise this control including: fuzzy logic controllers [49], H_∞ controllers [50] and LQR controllers [51]. However, the most popular choice at present is the Proportional

Integral Derivative Controller (PID-Controller).

2.6 PID-controller

As mentioned in section 2.5 the PID-controller is one of the most popular forms of control for suspension control. This is due to the controller's simplicity and ability to output accurate and continuous control.

These controllers are composed of three parts, a proportional part, an integral part and a derivative part, as can be seen in Figure 2.15.

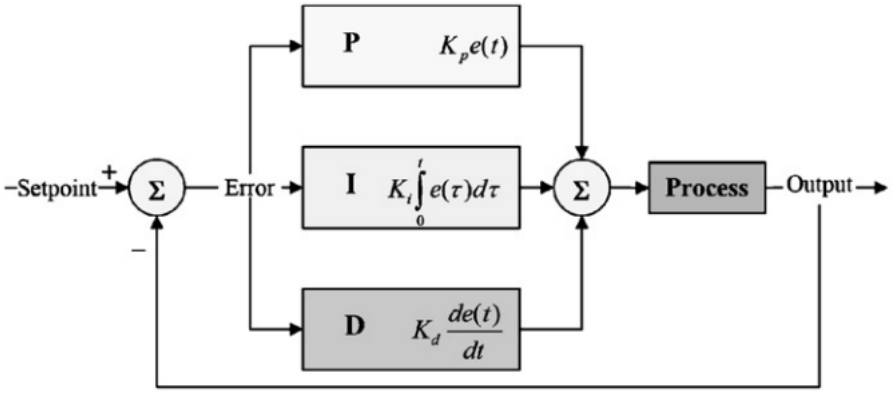

Figure 2.15: PID-controller [12]

Each part has a gain associated with it. The proportional section calculates its output by calculating the product of the error and the proportional gain, K_p, as so:

$$Proportional\ Output = K_p e(t) \tag{2.34}$$

where:
K_p = the proportional gain
$e(t)$ = the error between the actual process output and target process output

The proportional gain dictates the system's response speed. However, if the proportional gain is too large it can result in large oscillations around the set point i.e. the target value.

The next section is the integral section which calculates its output as the product of the integral gain, K_i, and the integral of the error over time, it is of the form:

$$Integral\ Output = K_i \int_0^t e(\tau)d\tau \tag{2.35}$$

where:

K_i = the integral gain
$e(\tau)$ = error

The integral section is responsible for reducing the steady state error, which is the difference between the target and actual value of a system as time goes to infinity.

Finally, the derivative section produces its output by calculating the product of the derivative gain, K_d, and the rate of change of the error. It is of the form:

$$\textit{Derivative Output} = K_d \frac{de(t)}{dt} \tag{2.36}$$

where:
K_d = the derivative gain
$e(t)$ = the error

By monitoring the rate of change of the system the derivative section acts as brake for the system by reducing the speed of the response if it is too rapid.

Once all the individual sections have been calculated they are summed to create the manipulated variable which is then passed to the process.

PID-controllers are deployed in numerous different closed loop feedback systems, where, when given the error between the current output and target output, they can adjust the manipulated variable to reduce this error.

However, whilst PID-controllers are very popular, they do have a number of drawbacks. The first problem is the linearity of PID-controllers. PID-controllers are linear systems and as such they have been known to struggle with complex non-linear systems. As well as this, PID-controllers are highly sensitive to changes in their gain values, as can be seen in Figure 2.16. These gains are chosen and tuned for deployment in particular environments, but even minor changes in these environments can reduce the performance of the PID-controller. These changes may be in the form of temperature fluctuations or wear and tear on the system. Due to this many PID-controllers that are deployed in real world systems often undergo routine tuning where possible. However, in certain systems, such as suspension control systems, this is not possible and therefore the PID-controller's performance can reduce over time. In addition, current tuning methods are very tedious and often struggle to agree on PID gain values. As a result, a lot of research has focused on the development of self-tuning and online tuning PID methods.

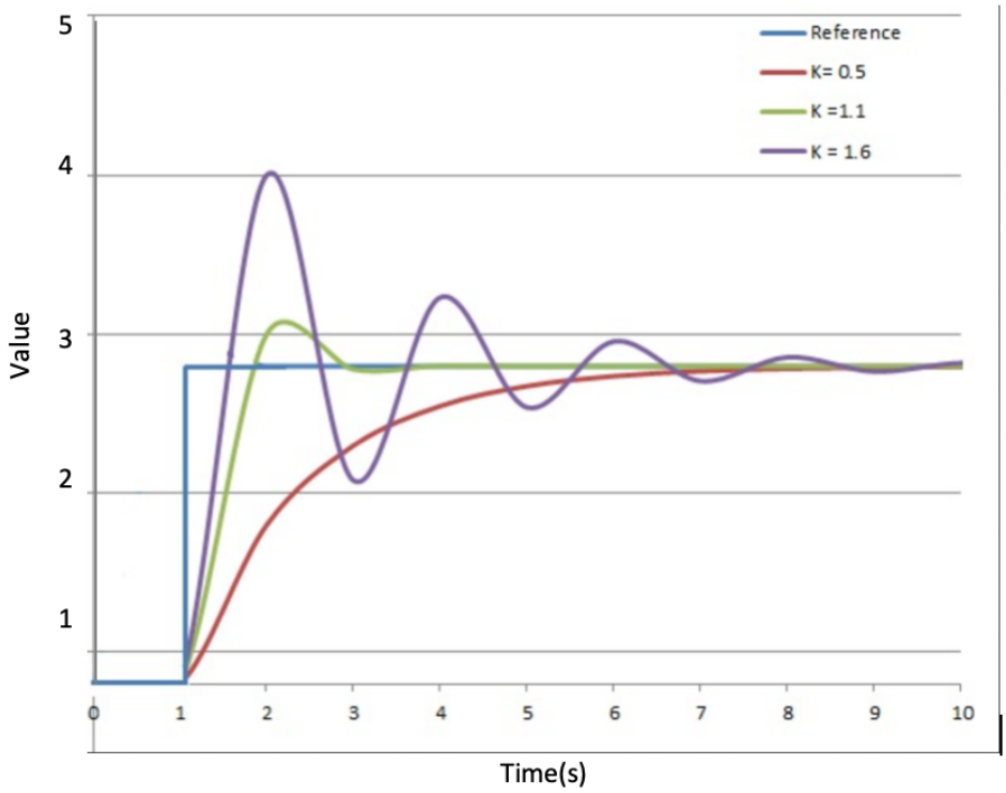

Figure 2.16: PID-controller with K_p being changed [1] **Note:** that this diagram only highlights the deviations in performance when K_p is changed. Similar performance variations occur when changing either of the other two gains.

2.7 Summary

In section 2.1 Reinforcement Learning was introduced alongside Q-learning. Section 2.2 described the paradigm of deep learning, including Backpropagation, activation functions and optimisation algorithms. The Universal Approximation Theorem was also outlined here. Section 2.3 then introduced deep reinforcement learning, which combined reinforcement learning techniques with deep learning methods. The DDPG algorithm and the Residual Policy Reinforcement Learning algorithm were also outlined here. Section 2.4 described control systems whilst section 2.5 explained how suspension systems work and why they can be considered a control system. Finally, section 2.6 introduced PID-controllers and provided insight into how they work and the pros and cons of using them.

3 Related Work

This section recaps previous work from literature which has attempted to overcome some of the difficulties associated with PID-control by either developing online tuning methods for PID-controllers to improve their adaptability, or by attempting to completely replace PID-control with artificially intelligent methods. Both families of techniques are applicable to this research as Residual Policy Reinforcement Learning acts as a hybrid of both by trying to tune the PID-controller and optimise its performance by learning a secondary policy which can compensate in areas where the PID-controller may struggle.

Section 3.1 touches on self-tuning methods, encompassing both traditional and reinforcement learning approaches, whilst section 3.2 focuses on work aimed at completely replacing PID-controllers through reinforcement learning. Section 3.3 focuses on work where Residual Policy Reinforcement Learning has been applied successfully to a Proportional (P) controller. Finally, Section 3.4 provides a brief summary of all this research and a conclusion.

3.1 Online PID Tuning Methods

PID-controllers are commonly used in a variety of control systems due to their ease of implementation and ability to provide a continuous, accurate output. However, as mentioned in section 2.6, a PID-controller's performance is highly dependent on the values of its parameters. These parameters are optimised for particular environments, and as such they begin to become sub optimal as environmental characteristics change. These changes can be the result of a number of factors, including wear and tear and temperature deviations.

In order to combat these environmental changes, some systems can be taken offline to undergo routine PID tuning, periodically, in order to re-optimise the PID's performance. However, this is not always possible and as such a lot of research has focused on self-tuning and online tuning methods. These methods allow PID-controllers to re-optimise in real time and independently, in order to counteract any deviations in environmental properties.

This research has created a variety of online tuning methods, some of which are related to reinforcement learning and others which have adopted other, more traditional approaches.

These are of particular interest to this work, as one of the aims of this thesis is to show that Residual Policy Reinforcement Learning has the ability to compensate for poorly tuned PID-controllers and act as an online tuning method of sorts. This section looks at some of the other attempts to do this and apply online tuning methods using both traditional methods and RL techniques. Subsection 3.1.1 discusses the traditional, non-RL related methods used to tune these controllers, whilst subsection 3.1.2 introduces some of the RL related methods.

3.1.1 Traditional Tuning Methods

The following approaches outline some of the non-RL related techniques seen in the literature.

[13] presents an Adaptive PID (APID) controller for use in suspension control. The algorithm makes use of gradient descent to optimise the PID gains online, which guarantees convergence in a closed loop system. In order to do this the authors identified an objective function of the form;

$$J = \frac{1}{2}(y - r)^2 \tag{3.1}$$

Equation 3.1 APID Objective Function [13]

where:
r = desired force
y = actual force

The gradient descent algorithm was then used to create a set of update parameters as seen in Fig 3.1

$$\dot{K}_p = -\eta \frac{\partial J}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial K_p},$$

$$\dot{K}_i = -\eta \frac{\partial J}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial K_i},$$

$$\dot{K}_i = -\eta \frac{\partial J}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial K_i},$$

Figure 3.1: APID Update Parameters [13] **Note:** the negative sign, as gradient descent is attempting to minimise the cost

These values were then used to calculate the actual PID gain value, as seen in Figure 3.2

$$\dot{K}_p(t+1) = K_p(t) + \dot{K}_p(t),$$

$$\dot{K}_i(t+1) = K_i(t) + \dot{K}_i(t),$$

$$\dot{K}_d(t+1) = K_i(t) + \dot{K}_d(t).$$

Figure 3.2: APID PID Gain Updates [13]

This was then used to create the algorithm seen in Figure 3.3.

Step 1: Set the reference r and output y values of the closed-loop system.
Step 2: Update the parameters of the APID control, i.e. K_p, K_i, and K_d using Equations (25)–(27).
Step 3: Calculate the output of APID controller.
Step 4: Finally, output of the controller with disturbances added is applied to the system.
Step 5: Repeat steps (2–4) until the required error tolerance is achieved.

Figure 3.3: APID Algorithm [13]

[14] proposes a fuzzy logic self-tuning PID-controller for use in suspension control. Here a fuzzy logic controller is used to decide the gains for the PID-controller, based on the chassis velocity error and the change in this error, as can be seen in Figure 3.4.

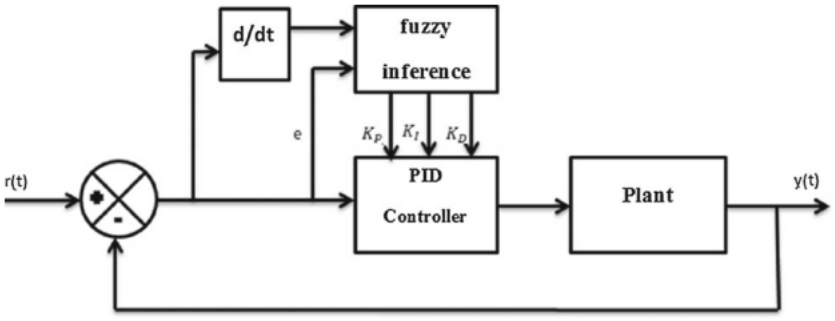

Figure 3.4: Fuzzy Logic Self Tuning PID Implementation [14]

The authors devised a set of fuzzy rules based on a set of classifications for both the input and outputs. The inputs had 5 different classifications (Negative Large (NL), Negative Small

(NS), Zero (ZE), Positive Small (PS), Positive Large (PL)) and the outputs had 7 different classifications (Positive Very Small (PVS), Positive Small (PS), Positive Medium Small (PMS), Positive Medium (PM), Positive Medium Large (PML), Positive Large (PL) and Positive Very Large (PVL)). As can be seen in Figure 3.5 the input classifications determine the output classifications.

Table 2 Fuzzy rules for computing K_p

ve/ce	NL	NS	ZE	PS	PL
NL	PVL	PVL	PVL	PVL	PVL
NS	PML	PML	PML	PML	PML
ZE	PVS	PVS	PS	PMS	PMS
PS	PML	PML	PML	PM	PM
PL	PVL	PVL	PVL	PVL	PVL

Table 3 Fuzzy rules for computing K_i

ve/ce	NL	NS	ZE	PS	PL
NL	PM	PM	PM	PM	PM
NS	PMS	PMS	PMS	PMS	PMS
ZE	PS	PS	PVS	PS	PS
PS	PMS	PMS	PMS	PMS	PMS
PL	PM	PM	PM	PM	PM

Table 4 Fuzzy rules for computing K_d

ve/ce	NL	NS	ZE	PS	PL
NL	PVL	PVL	PVL	PVL	PVL
NS	PMS	PMS	PMS	PMS	PMS
ZE	PS	PS	PVS	PS	PS
PS	PMS	PMS	PMS	PMS	PMS
PL	PM	PM	PM	PM	PM

Figure 3.5: Fuzzy Logic Class Tables [14] **Note:** that the outputs are classed based on the combination of input classes e.g. in Table 2 in the image NL and NL gives and output class of PVL

Membership function plots are of triangular form, as can be seen in Fig 3.6. In this case, these plots determine the value of K_p based on the fuzzy logic rules.

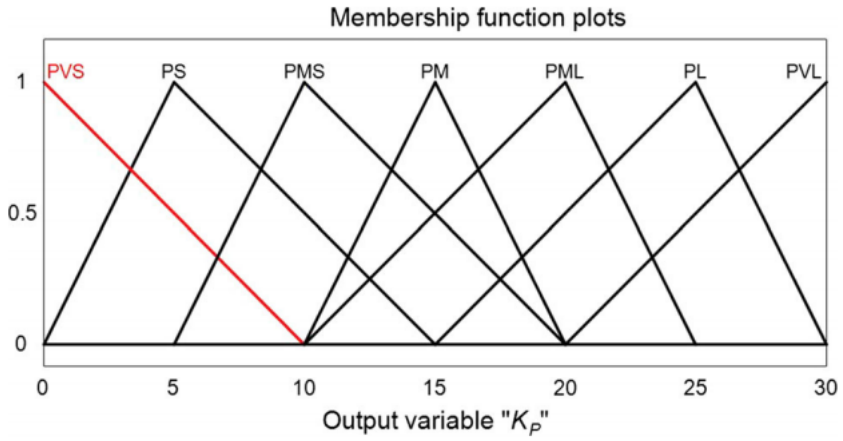

Figure 3.6: Fuzzy Logic Membership Plots for K_p [14] **Note:** The triangular shape of the membership plots

[15] proposes a neural network for tuning PID-controllers for use in Underwater Remotely Operated Vehicles (ROVs). Neural Networks are chosen here due to their high fault tolerance, adaptability, parallelism and ability to deal with non-linearities. As can be seen in Figure 3.7 the architecture is composed of a single hidden layer connecting an input and output layer. The neural network predicts the PID gains as outputs, and training is done using the Backpropagation algorithm.

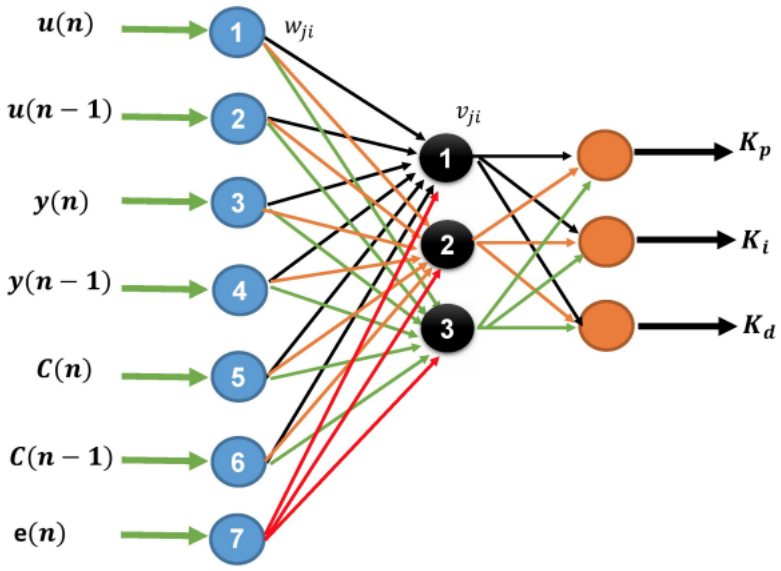

Figure 3.7: Neural Network PID Tuner Architecture [15]

The Backpropagation algorithm makes use of the chain rule to update the network's parameters, as described in section 2.2.2. This is based on the feedback of the error which, as can be seen in Figure 3.8, is based on the difference between the actual trajectory and the

desired trajectory. It can also be seen here that the network receives the desired trajectory, actual trajectory and PID output as inputs.

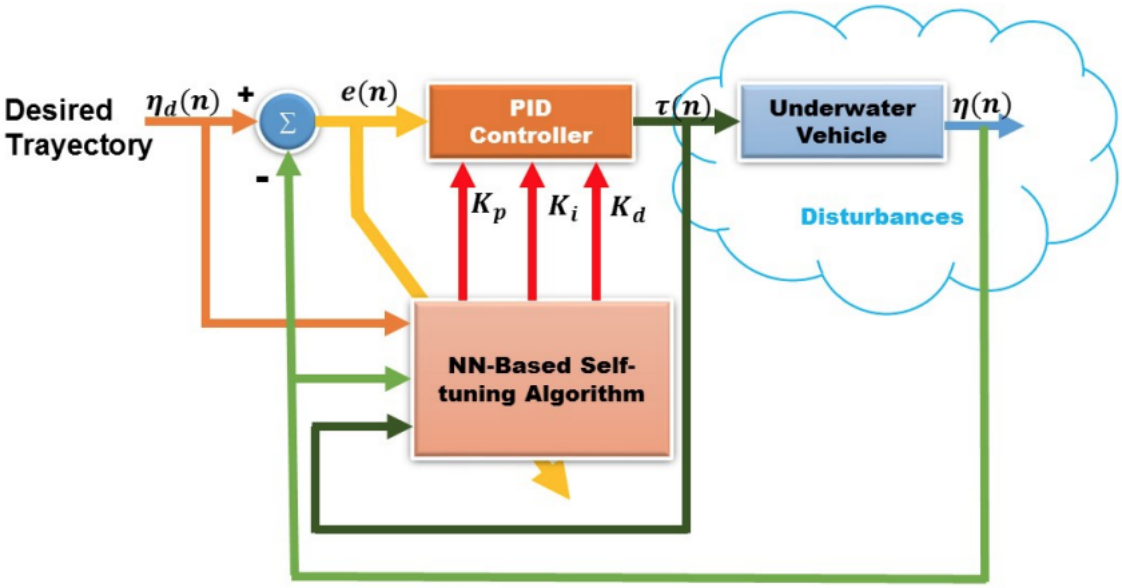

Figure 3.8: Neural Network PID Tuner Implementation [15]

Whilst all these methods were designed to tune PID-controllers, they were tested in a variety of different environments. [13] built a full car suspension model i.e. a model composed of all 4 wheels, with 8 degrees of freedom to test the APID algorithm. This suspension model was then used to test the algorithm on a variety of different road profiles, including potholes and sine wave disturbances. The algorithm proved to outperform both passive and semi-active skyhook systems in terms of heave, heave response, pitch and maximum displacement. Similarly, [14] tested the fuzzy control self-tuning PID-controller on a half car suspension system i.e. a system consisting of only two wheels. Again the system was tested on a variety of road inputs, including sinusoidal bumps and potholes. The Fuzzy Logic Self-Tuning PID-controller was then compared against a fuzzy logic controller, a conventional PID-controller and a H∞ controller. In every case the Fuzzy Logic self-tuning PID-controller outperformed the other controllers in terms of mean square error (MSE). It also outperformed the other controllers in terms of max displacement on 2 out of the 3 different road profiles. Finally, [15] evaluated the neural network based self-tuning algorithm on a simulation representing an underactuated 6 degree of freedom underwater ROV. The neural network self-tuning algorithm was compared to the conventional PID-controller, and showed an improvement over the conventional PID-controller in terms of trajectory, with a MSE of approximately 2.2cm versus 3.1cm recorded by the conventional PID-controller.

As can be seen from the work above, self-tuning and online tuning PID techniques have the ability to improve the performance of PID-controllers and as such have become a prominent

and important aspect of research. Whilst only two of the methods were tested on suspension control systems, all methods showed the performance gains of implementing online tuning methods for PID-controllers.

Table 3.1 provides a brief summary of these methods.

Reference	Algorithm	Analysis
[13]	Adaptive PID Using Gradient Descent	Full Car Suspension Model
[14]	Fuzzy Logic Self-Tuning PID	Half Car Suspension Model
[15]	Neural Network Self-Tuning PID	Underwater ROV Model

Table 3.1: Non-RL Online Tuning Techniques

3.1.2 Reinforcement Learning Tuning Methods

There has also been a number of attempts to apply reinforcement learning techniques to tune PID-controllers. This is in part due to the characteristics of reinforcement learning methods, in particular, deep reinforcement learning methods which display many of the positive attributes of neural networks, which are outlined in Section 3.1.1. As well as these attributes, reinforcement learning algorithms also benefit from the fact that they need very little prior information or data about a system before being trained. As such reinforcement learning algorithms have been the subject of a lot of research lately, some of which is outlined below.

[16] presents an incremental Q-learning strategy for adaptive PID-control. The algorithm is based on the popular tabular Q-learning algorithm described in section 2.1.1. Here the algorithm is used to create a table describing the performance of a particular action in a particular state space, where the action represents the optimal PID gains for the controller.

In the traditional algorithm the number of states and actions are fixed, meaning there is a table of fixed size. However, in this implementation the number of states and actions can grow dynamically.

The dynamic state space allocations are based on incremental state aggregation, whereby new states are analysed to determine if they are in the same state space as a known state. In other words the algorithm tries to identify how similar two states are. This is done by measuring the Euclidean distance between two states to determine if states are within a

certain threshold of each other. Any new state that is not within the Euclidean distance threshold of a previous state in the table will then be listed itself in the table, allowing the table to grow dynamically.

This is done incrementally, as can be seen in Figure 3.9, hence the name Incremental Q-learning Learning Strategy.

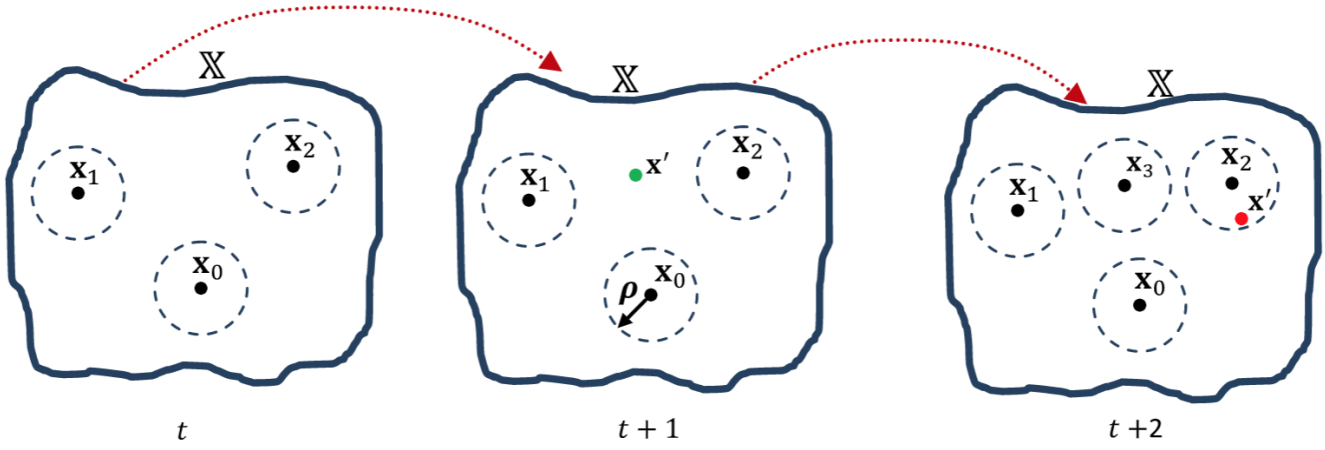

Figure 3.9: State Aggregation for Incremental Q-Learning [16]

The action space discretization is done differently, however, the goal is still to allow it to grow dynamically. In this case new actions are added to the table, and the discretization level is augmented based on the invariance of the system. In other words, the action space grows if actions taken have little to no effect on the system. If there is complete invariance then the number of potential actions will increase within the "neighbourhood" of the original action as can be seen in Figure 3.10

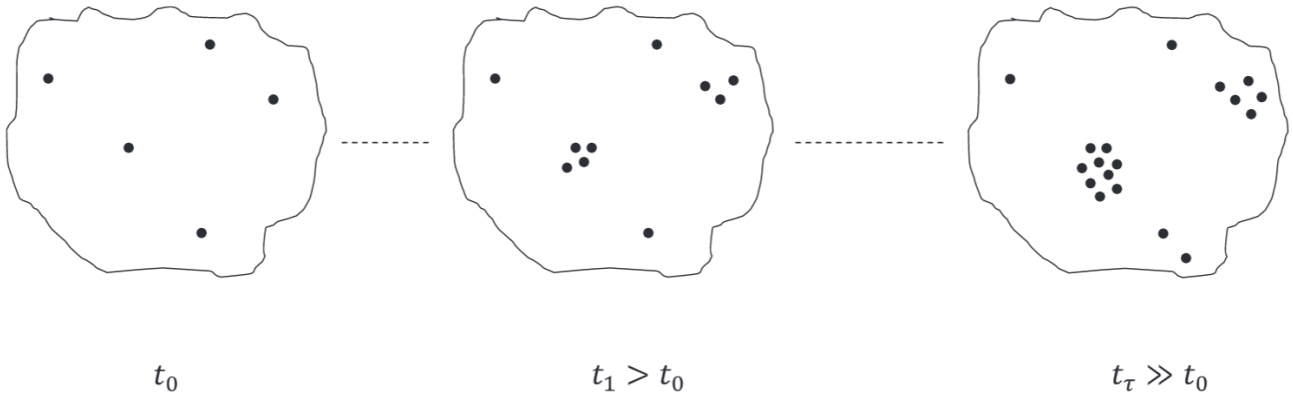

Figure 3.10: Action Aggregation for Incremental Q-Learning [16] **Note:** the increase in points in particular regions, this represents the increase in actions in those areas that showed invariance

These two aggregation techniques are then combined to create a dynamically growing Q-learning table where values can be assigned to particular actions in particular states. This then allows an agent to choose the action which is deemed to have the highest reward, as is the case in the original tabular Q-learning algorithm.

[52] also proposes a reinforcement learning tuning technique, however, this technique uses a continuous action space for selecting the PID gains, as opposed to the discrete-action space seen in [16]. In order to achieve this, the technique uses the Continuous Action Reinforcement Learning Automata (CARLA) algorithm. CARLA works by assigning a probability distribution to a range of actions. The action with the highest probability will then be selected most often. This distribution can then be updated based on a Gaussian neighbourhood function, which takes into account the performance of a particular action i.e. if an action performs well in a particular state the neighbourhood function updates the distribution to give this action a higher probability. The authors here deploy three separate CARLA agents, with each agent responsible for a different PID gain i.e. one for the proportional gain, one for the integral gain etc. The agents operate independently of each other, with the only interconnection being the environment and a shared reward function.

[17] also proposes a continuous-action reinforcement learning tuning technique. However, in this case the authors opt to use an actor-critic model, as opposed to the CARLA method seen in [52]. As described in section 2.3.3, these models are usually composed of two separate parts; an actor and a critic. However, the authors here have designed a single network as can be seen in Figure 3.11. This is possible as they have chosen a Radial Basis Function (RBF) network which has a simple structure and can be trained efficiently.

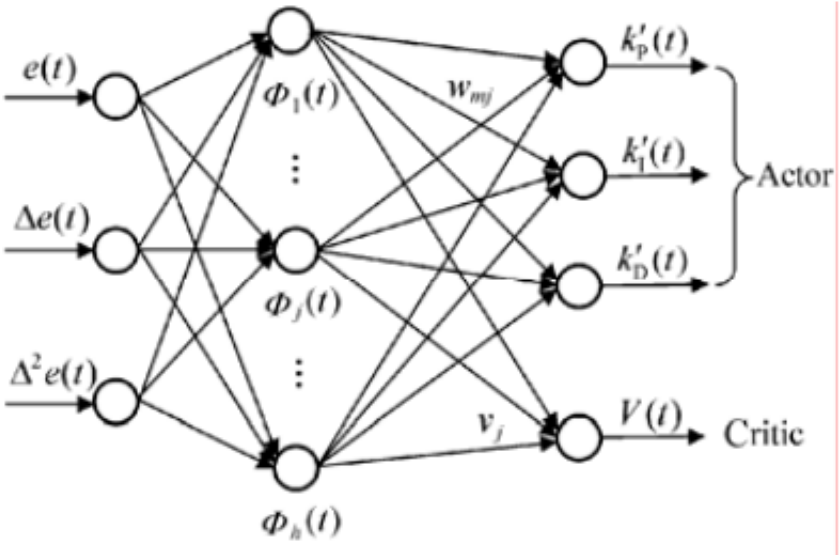

Figure 3.11: Actor Critic Network for PID Tuning [17] **Note:** the last output is the critics valuation

As there is only one network only one update is needed for both the actor and critic.

[17] has also chosen to add a Stochastic Action Modifier (SAM) as can be seen in Figure 3.12 which stochastically selects the PID gains based on the actor's suggestion and the valuation from the critic.

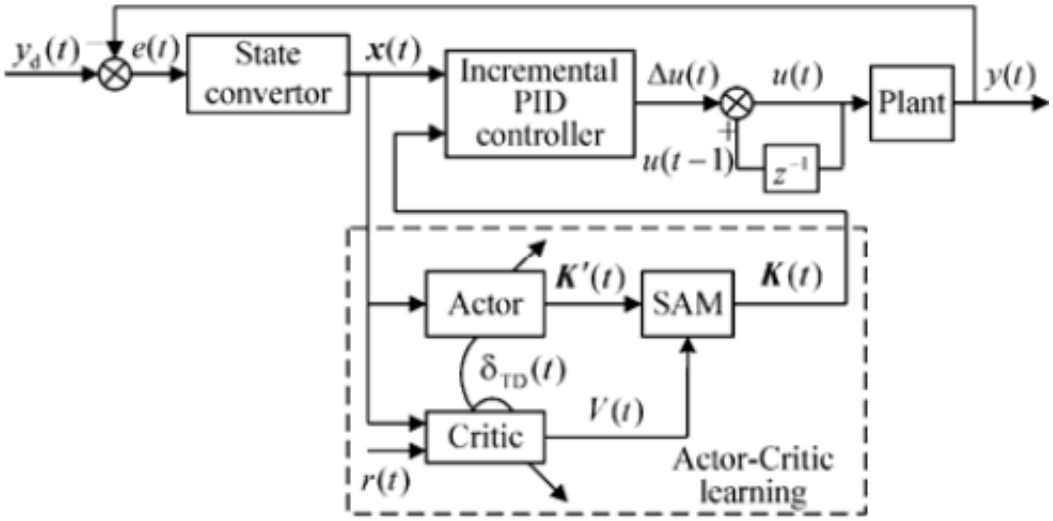

Figure 3.12: Actor Critic Model for PID Tuning [17] **Note:** the addition of the SAM unit

Whilst these algorithms were all designed to tune PID-controllers online, each were analysed in different applications. [16] uses the Pioneer 3AT® terrestrial mobile robot as the experimental platform. A variety of different test cases were established, each varying in difficulty, with the algorithm showing itself capable of maintaining control in a variety of environments. More impressively, it also demonstrated the algorithm's ability to alter the PID gains in order to compensate for changes in the system. [52] applied the CARLA tuning algorithm to engine idle-speed control. Initially, the algorithm was tested on a simulation based on the Cook and Powell structure described in [53]. Here the model was compared with the Ziegler-Nichols tuning method and proved to outperform it in terms of settling time and overshoot. It was then applied to a real engine system and again was able to outperform the Ziegler-Nichols method. Finally, [17] tested their adaptive PID algorithm in a simple sine wave tracking task and compared the results against a conventional PID-controller. Testing showed that the Adaptive PID-controller was capable of significantly minimizing the error experienced during tracking. The authors also noted that their algorithm was strongly robust to system disturbances, which allowed it to outperform the conventional PID-controller. In addition, they concluded that their adaptive PID-controller was capable of realizing stable tracking of complex non-linear systems, something which conventional PID-controllers can struggle with.

These results highlight the potential role of reinforcement learning in optimising and tuning PID-controllers online. It also highlights the ability of reinforcement learning algorithms to help PID-controllers deal with non-linear systems. As a result, there is strong evidence here that when reinforcement learning algorithms are used to aid PID-controllers it can often result in superior performance.

Table 3.2 provides a brief summary of these methods.

Reference	Algorithm	Action Space	State Space	Analysis
[16]	Incremental DQN	Discrete (dynamic)	Discrete (dynamic)	Mobile Robots
[52]	CARLA	Continuous	Continuous	Engine idle-speed
[17]	RBF Actor Critic	Continuous	Continuous	Sine Wave Tracking

Table 3.2: Reinforcement Learning Online Tuning Techniques

3.2 Replacing PID Control with Reinforcement Learning

This section looks at previous work focused around replacing PID-controllers with reinforcement learning alternatives. As mentioned previously, the Residual Policy Reinforcement Learning algorithm can be viewed as a hybrid between an online PID tuning method and a RL agent attempting to replace a PID-controller. As such it is important to consider work where researchers have successfully used reinforcement learning to improve and replace PID-controllers.

[54] proposes an unmanned surface vehicle path following control method based on the continuous-state, discrete-action reinforcement learning algorithm known as SARSA. SARSA was originally proposed in [55] and is an acronym for State Action Reward State Action. It operates in a similar way to the DQN algorithm described in Section 2.3.1. However, it is an on-policy variation, which means it learns how to improve based on its current policy, as opposed to a different policy. This is achieved by using the current policy to determine the next action based on the next state, and storing these in the memory buffer i.e. storing the current state, current action, reward, next state and next action (SARSA) in the memory buffer. The fact that the next action is stored is different to off-policy methods such as

DQN which only stores the current state, current action, reward and next state (SARS).

In this case, the SARSA algorithm was used to determine the rudder angle needed to allow an unmanned surface vehicle to follow a path, a task usually done by PID-controllers. In order to do this the agent was provided with information to do with heading deviation, water current and wind. This was then used to calculate the correct rudder angle, which was allowed vary between -25°and 25°. As the SARSA algorithm is a discrete-action algorithm i.e. it can only pick from a limited number of actions, the action space was broken down into 51 different actions, each one representing a 1°increment i.e. the action space was {-25°, -24°, -23°..... 23°, 24°, 25°}. The agent was rewarded if actions reduced the heading deviation between the current position and the desired position.

[18] also proposes a deep reinforcement learning method, this time for an intelligent control strategy for transient response of a variable geometry turbocharger system. In this case, the RL algorithm chosen was the Deep Deterministic Policy Gradient (DDPG) algorithm discussed in section 2.3.4. As mentioned previously, this is an actor-critic model capable of operating in continuous-state, continuous-action spaces. In this task, the algorithm was designed to control a variable geometry turbocharger and as such had a state vector describing the engine speed, actual boost pressure, desired boost pressure and the vane position. The action space was the vane position which was controlled by a membrane vacuum actuator.

The actor's architecture can be seen in Figure 3.13, whilst the critic's architecture can be seen in Figure 3.14.

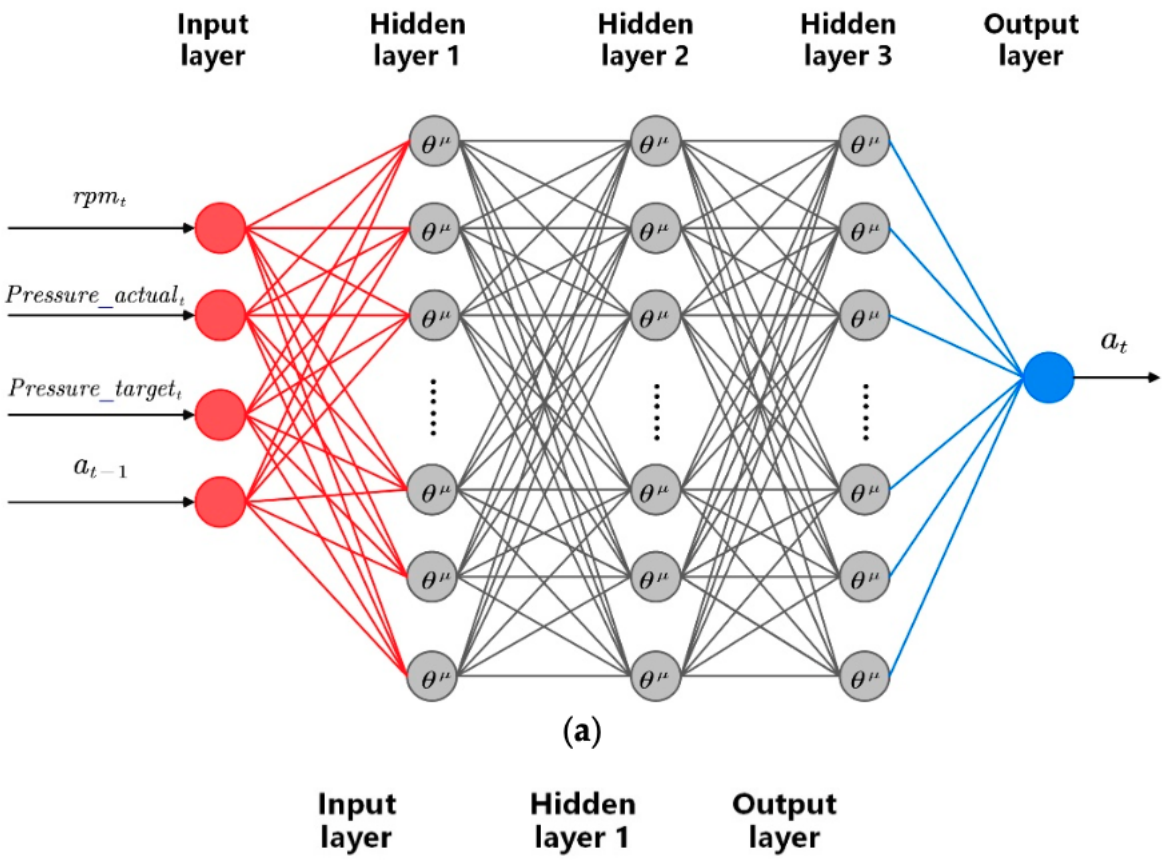

Figure 3.13: Actor's Architecture for Turbo-Charger control [18]

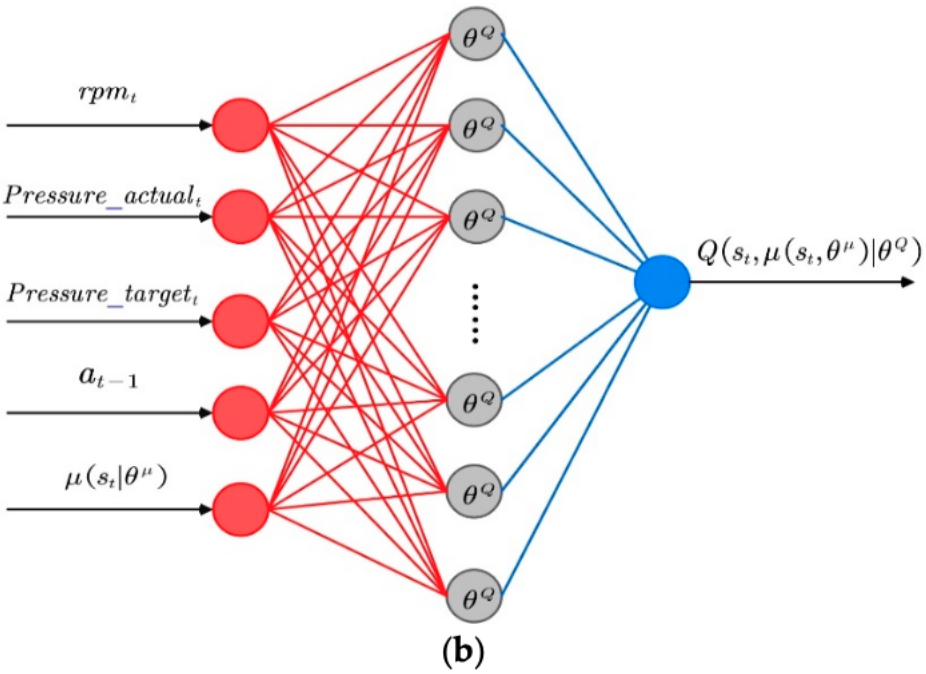

Figure 3.14: Critic's Architecture for Turbo-Charger control [18] **Note:** the critic receives one extra input, which is the action taken by the actor

Similarly, [56] also uses the DDPG algorithm, this time to control the lane keeping assistance (LKA) and the adaptive cruise control (ACC) systems for vehicle control. In this case, the agent was trained to maintain a safe driving distance from a leading car whilst maintaining its position in a lane. In order to do this the agent received information about the lateral deviation and the relative heading, and the rate of change of each of these. It was also provided with the velocity of itself and the leading car, the distance between the two, and the current angle of steering. From this information the agent was tasked with choosing the acceleration and the steering angle.

All these algorithms were then tested on different environments. [54] tested the SARSA USV control algorithm in a simulation USV path finding environment based on the parameters of the CybershipII simulation in [57]. The SARSA control algorithm was compared against a PID-controller in a number of different tests, including path following tests in ideal conditions i.e. no wind or currents and in tests with interference. In the ideal tests it showed similar performance to the PID-controller but was able to converge quicker, whilst in the tests with interference it was able to outperform the PID-controller. [18] tested the DDPG variable geometry turbocharger control algorithm on an advanced co-simulation platform. Again, this was compared to a fine tuned PID-controller using Integral Absolute Error (IAE). Again, the DDPG algorithm was able to outperform the PID-controller. The authors also deployed an agent, pre-trained in a different environment, into a new simulation and were able to show its ability to re-adapt to the new environment. This highlighted the DDPG algorithm's ability of adapting to small changes in the environment, similar to those experienced when deploying control methods optimised in simulation to the real world. Finally, [56] tested the DDPG path tracking method using a simulation environment. Two tests were devised, one using a circular path and another using a clothoid path. Results showed that the DDPG algorithm was able to substantially outperform the PID-controller on the more difficult clothoid path, however, the improvement on the simpler circular road was a lot smaller with both control methods performing very well.

As can be seen from the research above RL algorithms are very capable of replacing and outperforming their PID counterparts whilst also having an increased degree of adaptability. As such, for complex control systems RL methods represent a very promising alternative to traditional control methods and this research suggests that they have the potential to target areas of system control that PID-controllers struggle with.

Table 3.3 provides a brief summary of the above methods.

Reference	Algorithm	Action Space	State Space	Analysis
[54]	SARSA	Discrete	Continuous	Path Following USV
[18]	DDPG	Continuous	Continuous	Variable Geometry Turbocharger
[56]	DDPG	Continuous	Continuous	Path Following LKA and ACC

Table 3.3: Reinforcement Learning Alternatives to PID-control Methods

3.3 Residual Policy Reinforcement Learning

This section discusses previous work in applying Residual Policy Reinforcement Learning to optimise the performance of a Proportional (P) controller.

[19] presents attempts to use Residual Policy Reinforcement Learning to optimise the performance of a P-controller in electric connector assembly tasks. As described in section 3.3, Residual Policy Reinforcement Learning works by combining a base policy with a reinforcement learning residual policy. Here the authors have chosen to use a P-controller as the base policy, but use two different reinforcement learning algorithms as their residual policies. The first reinforcement learning algorithm used is the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm first proposed in [58], which is essentially an enhanced version of the DDPG algorithm described in section 2.3.4. It attempts to improve on the DDPG algorithm by using two Q-function approximators to reduce value overestimation as well as delayed policy updates to stabilize training. The second algorithm is known as Soft Actor Critic (SAC) and was proposed in [59]. It is an off-policy value based reinforcement learning technique which uses a stochastic policy and aims to maximise the entropy i.e. the unpredictability of picking a particular action. This is aimed at preventing agents from getting stuck picking the same actions repeatedly.

These implementations were tested on three tasks; USB insertion tasks, D-sub connector tasks and Model-E connector tasks. For each task, 3 different tests were used. The first test was a vision based test, whereby the algorithms were tested on their ability to complete the tasks using only raw image observations and a distance measure. The second test used only sparse rewards i.e. a singular reward for success or failure. In the sparse rewards case, the algorithms were only tested on the USB connection task. Finally, the techniques were also tested using perfect state information and a dense reward signal. Both these algorithms were

tested and compared against stand-alone RL algorithms i.e. algorithms that did not use the P-controller, and RL algorithms which learnt with the aid of demonstrations from expert policies. In the perfect state information test the algorithms were only compared against the standalone reinforcement learning algorithms.

As can be seen from Figures 3.15 and 3.16 the Residual Policy Reinforcement Learning algorithm successfully managed to outperform the other methods in the vision-based test and the perfect state information test. In both cases the figures show the distance from the goal.

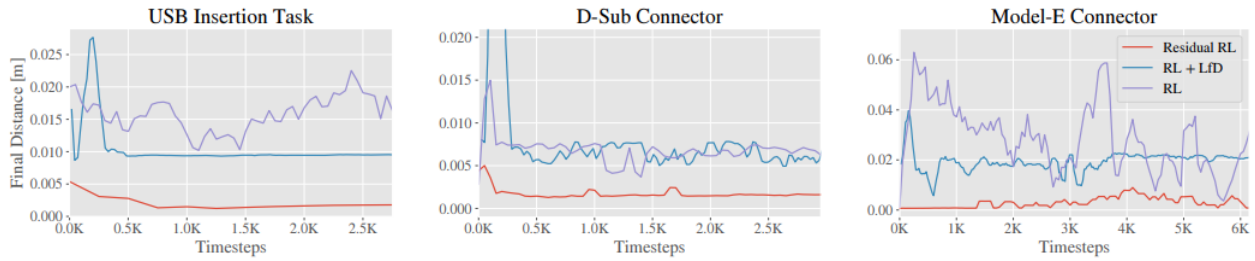

Figure 3.15: Residual Policy RL results for the Vision Based tests [19]

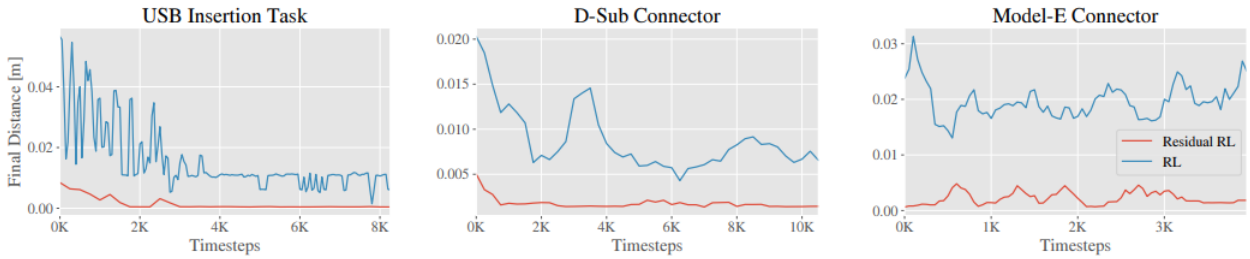

Figure 3.16: Residual Policy RL results for the Full State Information tests [19] **Note:** that the residual policy is only compared against the standalone Reinforcement Learning Techniques

The Residual Policy Reinforcement Learning algorithm also performed well in the sparse reward test, with results matching those of the other algorithms. However, as can be seen from Figure 3.17 the Residual Policy Reinforcement Learning algorithms took longer to converge than the algorithms making use of demonstration learning. As mentioned above this test was only conducted on the USB insertion environment.

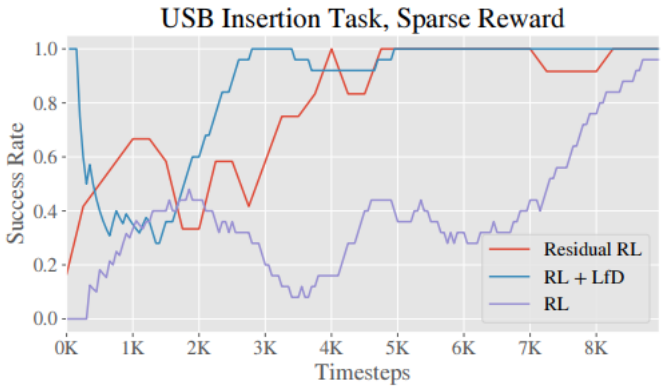

Figure 3.17: Residual Policy RL results for the Sparse Rewards tests [19] **Note:** that this test in only conducted on the USB insertion environment

Table 3.4 provides a quick summary of this chapter. The Performance columns refer to how the residual policy algorithm compared to the other techniques.

Reference	**Algorithm**	**Performance Vision Based**	**Performance Full State Info**	**Performance Sparse Reward**
[19]	Residual Policy RL	Better	Better	Equal

Table 3.4: Residual Policy Reinforcement Learning with P-controller base policy Summary Table

3.4 Summary

In Section 3.1.1 the potential of adaptive PID-controllers, without RL methods, was outlined. In Section 3.1.2 the paradigm of online tuning methods was extended to incorporate reinforcement learning techniques. In Section 3.2 the potential for reinforcement learning techniques to operate independently and to outperform PID-controllers was shown. It was also shown here that reinforcement learning techniques have the added benefit of being very adaptable to environmental changes and being capable of dealing with non-linear environments, something PID-controllers struggle with.

In addition to this, the research in Section 3.3, shows that a P-controller's performance can be optimised when deployed alongside a residual policy. In this case, the TD3 and SAC reinforcement learning algorithms were used to improve the performance of the P-controller. Therefore, there is strong evidence here to suggest that Residual Policy Reinforcement Learning algorithm can also be used alongside a PID-controller to optimise performance.

As such there is clear evidence that the combination of both PID-controllers and reinforcement learning algorithms alongside each other through the application of Residual Policy Reinforcement Learning is a completely reasonable idea due to the success of each method, both independently and in combination, as well as the success of Residual Policy Reinforcement Learning with other, similar, control methods.

It is clear therefore, that there are potential performance gains from tuning PID-controllers online, and from replacing them altogether with reinforcement learning techniques capable of dealing with certain areas of system control that PID-controllers struggle with. Due to this there is a strong case for the application of the Residual Policy Reinforcement Learning algorithm here as it acts as a hybrid of these two families of techniques. On one hand, Residual Policy Reinforcement Learning attempts to learn how to control the system by targeting areas where the PID-controller struggles. Whilst on the other, it tries to optimise the performance of the PID-controller in an online fashion by adapting to the environment and complimenting the PID-controller's policy, similar to how self-tuning methods work. As such, the evidence from the literature would suggest that Residual Policy Reinforcement Learning may be a very effective way of optimising the performance of the PID-controllers used in suspension control systems.

4 Design

This chapter outlines the approach used to design the suspension control problem as a RL task. It describes the design of the suspension system simulation, as well as the state and action spaces and the reward functions. It also describes the reasoning for choosing the Residual Policy Reinforcement Learning algorithm, as well as the additional features used, the training and the network architecture.

4.1 Suspension Control As a RL Problem

In order to simulate the behaviour of a suspension system, a simulation of a quarter car model was provided by ZF Friedrichshafen AG. The quarter model car describes the suspension system of a single wheel in car i.e. a quarter of the car. Similar simulations were used in [60, 61, 62].

In order to apply the Residual Policy Reinforcement Learning algorithm, the suspension control system needs to be framed as an RL problem. This can be done by allowing an agent to interact with a damper and provide this damper with a continuous damping rate, which describes the amount of damping the damper should apply. Specifically, the problem depends on an agent viewing state $s_t \epsilon S$, taking action $a_t \epsilon A$, to transition to state $s_{t+1} \epsilon S$ and receive reward R_t. These components will now be defined.

4.1.1 State Space

The state space is a vector describing the velocity and position of the system's wheel. These numbers are zero centred as can be seen in Figure 4.1 and Figure 4.2. A zero reading for the position means the wheel has not moved i.e. it is still in the resting position, whilst positive and negative values correspond to the wheel's movement in position upwards and downwards respectively. Similarly, a zero reading for velocity means the wheel is stationary, whilst positive and negative values indicate the wheel is moving upward and downwards respectively.

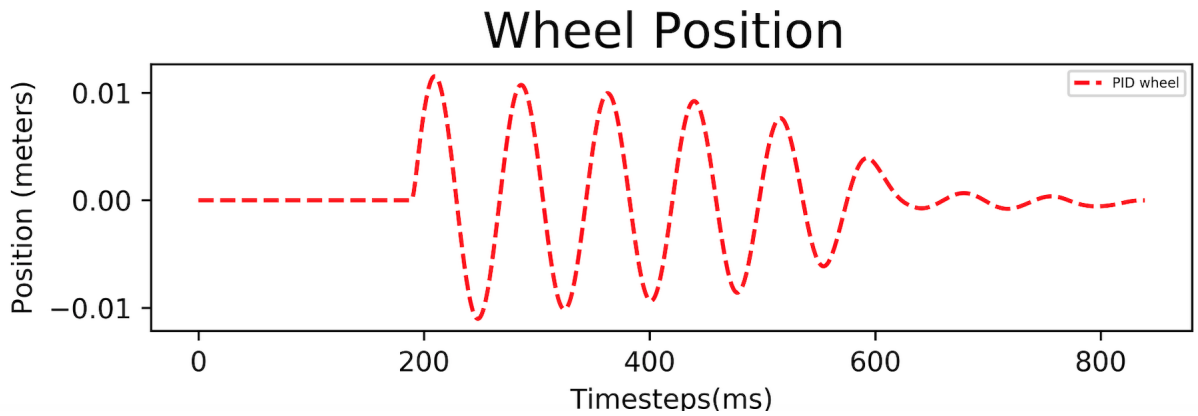

Figure 4.1: Wheel Position

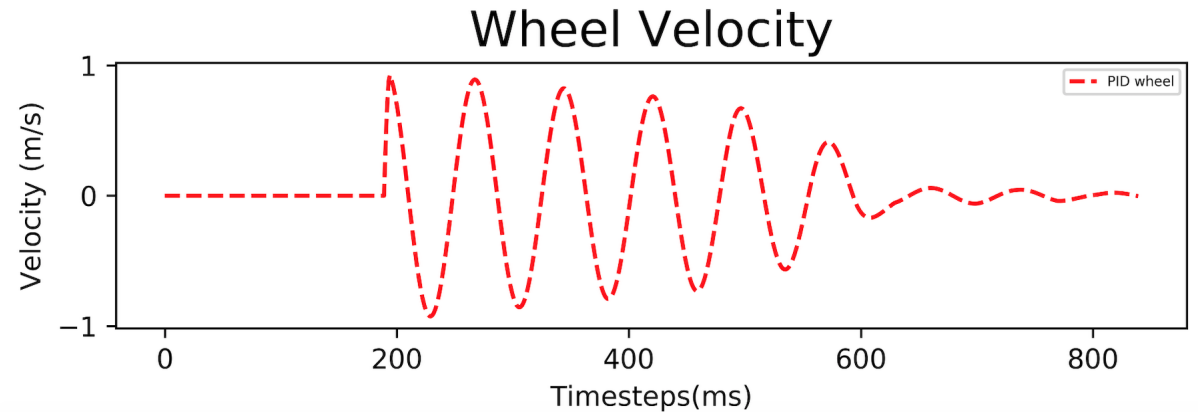

Figure 4.2: Wheel Velocity

The reason for using the wheel velocity and position instead of the chassis equivalents is due to the impact of a disturbance being felt on the wheel one timestep ahead of the chassis, hence it gives the agent a chance to react to protect the chassis.

These state vectors are calculated using a series of equations which take a number of factors into account, including the mass of the chassis and road disturbances.

4.1.2 Action Space

The action space is a scalar value in the range [100, 5000] which corresponds to the damping rate, also known as the K-value. The damping rate describes how much damping the damper should apply. The higher the damping rate, the more damping applied.

An important point to note is that the agent is only allowed to take an action every 35ms. This is due to the physical constraints of the system which can't accommodate constant changes in the damping rate. As a result, every action taken by the agent is maintained for at least 35ms. This can be seen in the figures below, where every action taken in Figure 4.3

is maintained for at least 35ms in Figure 4.11. By doing this, it can be considered as a roll-out of transitions which allows it to remain within the RL framework.

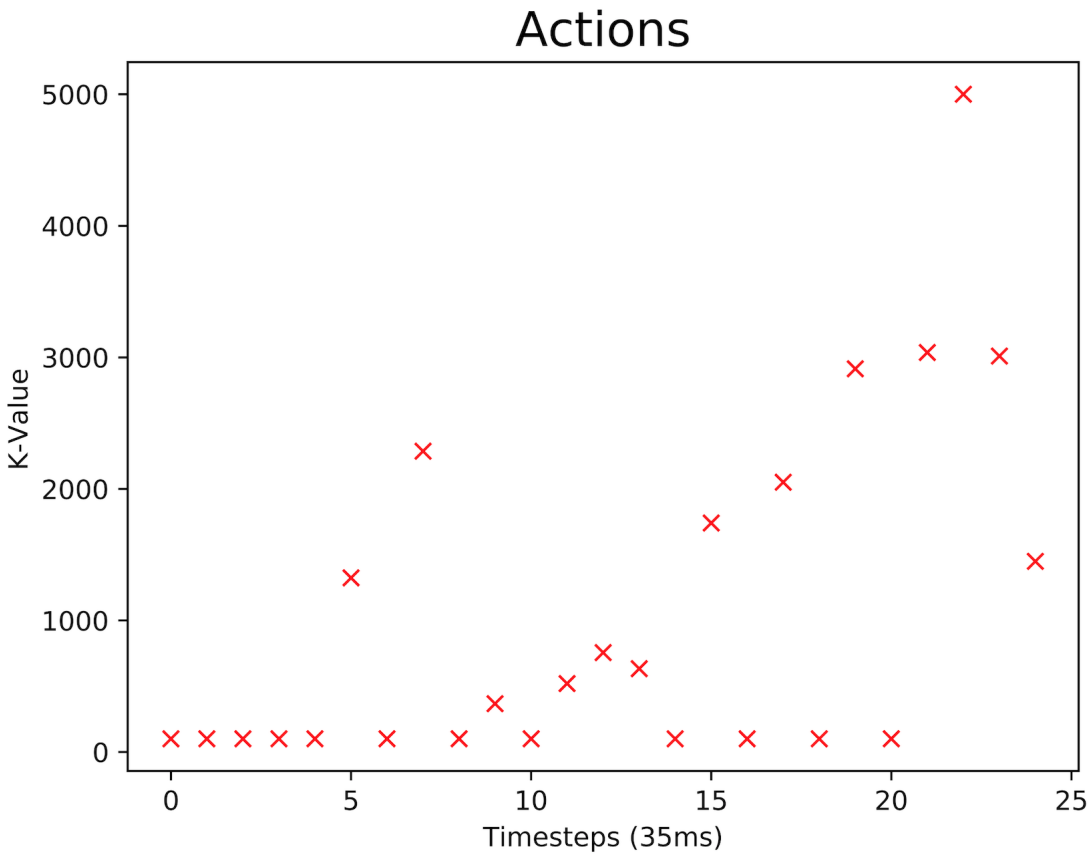

Figure 4.3: Agent's Actions Every 35ms

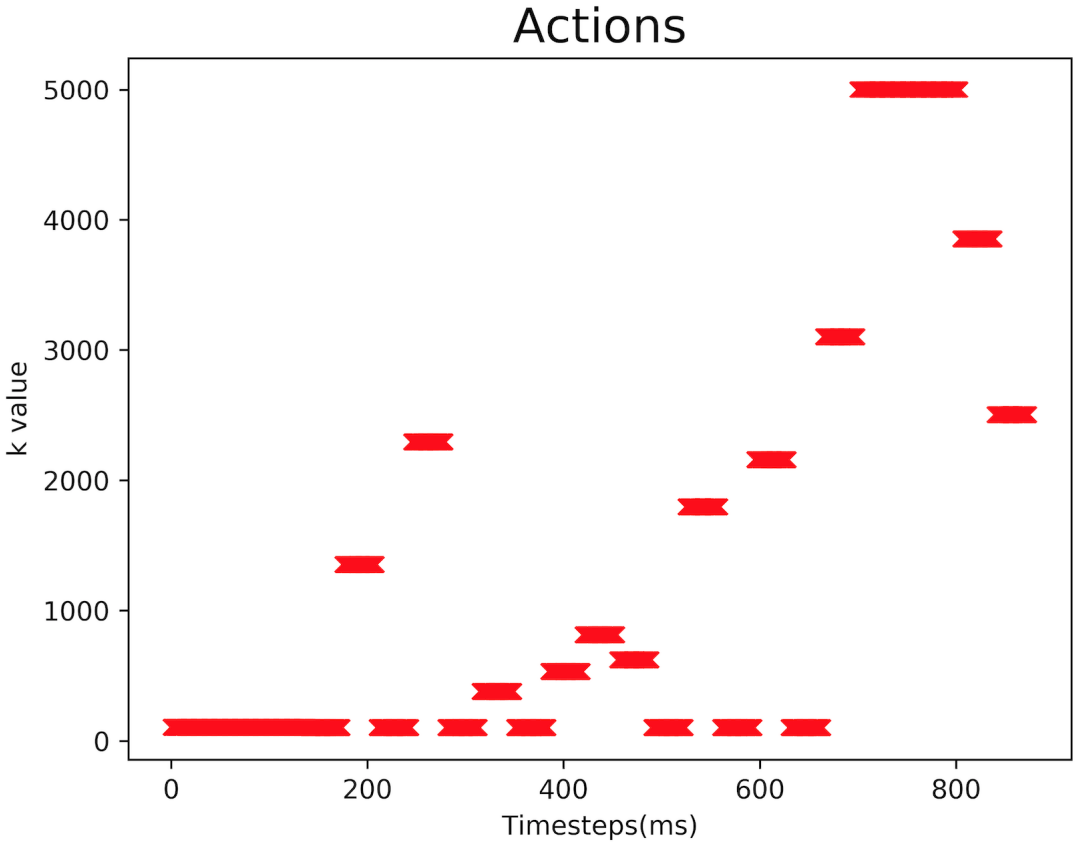

Figure 4.4: Agent's Actions **Note:** How the actions are maintained for at least 35ms

4.1.3 Reward Function

There were two reward functions tested and compared in order to determine which function resulted in the best overall performance. Performance was measured on a number of different metrics including maximum chassis acceleration, maximum chassis velocity and maximum chassis displacement. As well as this, the ISO 2631 Riding Comfort Standard [63] was used to determine the performance of each model. This standard is outlined below in Figure 4.5.

RMS (Root Mean Square) Acceleration	Ride comfort Criteria
a<0.315 m/s^2	Comfortable
0.315<a<0.63 m/s^2	A little Uncomfortable
0.5<a<1m/s^2	Fairy Uncomfortable
0.8<a<1.6m/s^2	Uncomfortable
1.25<a<2.5m/s^2	Very Uncomfortable
a>2.5m/s^2	Extremely Uncomfortable

Figure 4.5: ISO Riding Comfort Standard [20]

As can be seen from this picture accelerations over $0.315 m/s^2$ start to become uncomfortable. As such, each reward function was also compared based on how many times

chassis acceleration exceeded this value.

The first reward function tested, which will be labelled Reward Function 1, took the form;

$$\textit{Reward Function}\ 1 = -((\textit{clip}((\textit{abs}(\textit{chassis acceleration}) - 0.315), 0, 1) \times 100)) \qquad (4.1)$$

The term reward function may be slightly misleading here, as rather than rewarding the agent for taking the correct actions it essentially punishes it for allowing the chassis acceleration to exceed a $0.315 m/s^2$. As can be seen from Figure 4.6, when the acceleration remains within the boundaries the agent receives a reward of 0. As the acceleration increases outside these boundaries there is a linear decrease in the reward the agent receives until a saturation point of -100 occurs when the acceleration exceeds $1.315 m/s^2$. It is also important to note the use of the absolute function here to ensure that both accelerations in the positive and negative direction i.e. up and down are treated equally. This is in line with the use of the RMS acceleration shown in Figure 4.5.

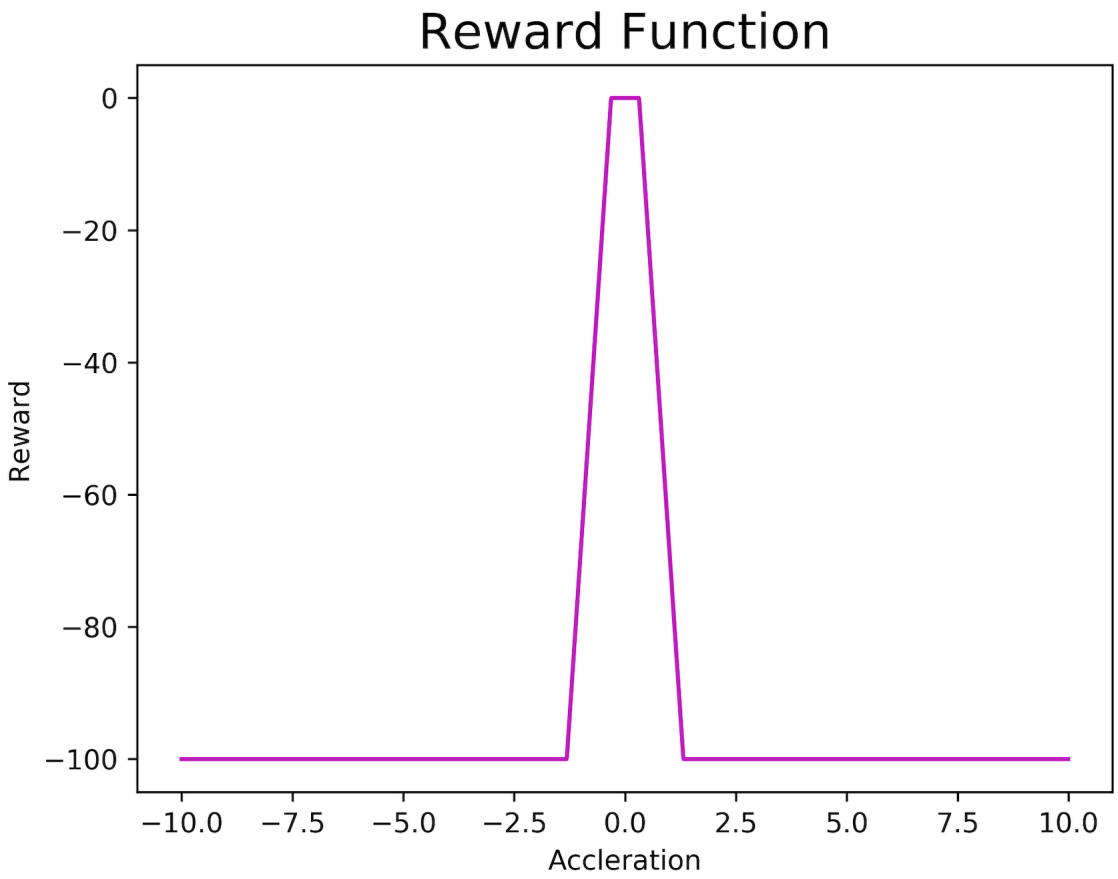

Figure 4.6: Reward Function 1

The second reward function, which will be labelled as Reward Function 2, took the form;

$$\textit{Reward Function } 2 = -((ceil(clip((abs(\textit{Chassis Acceleration}) - 0.315), 0, 1)) \times 100)) \quad (4.2)$$

Again, this function punishes the agent for chassis accelerations which exceeds the boundary of $0.315 m/s^2$. However, in this case the function takes the form of a unit function, as can be seen in Figure 4.7, which simply punishes the agent with -100 as soon as the chassis acceleration boundary is breached. This is different to Reward Function 1 which had a linear build up to a punishment of -100. Instead, Reward Function 2 is discontinuous and therefore may be more difficult to learn.

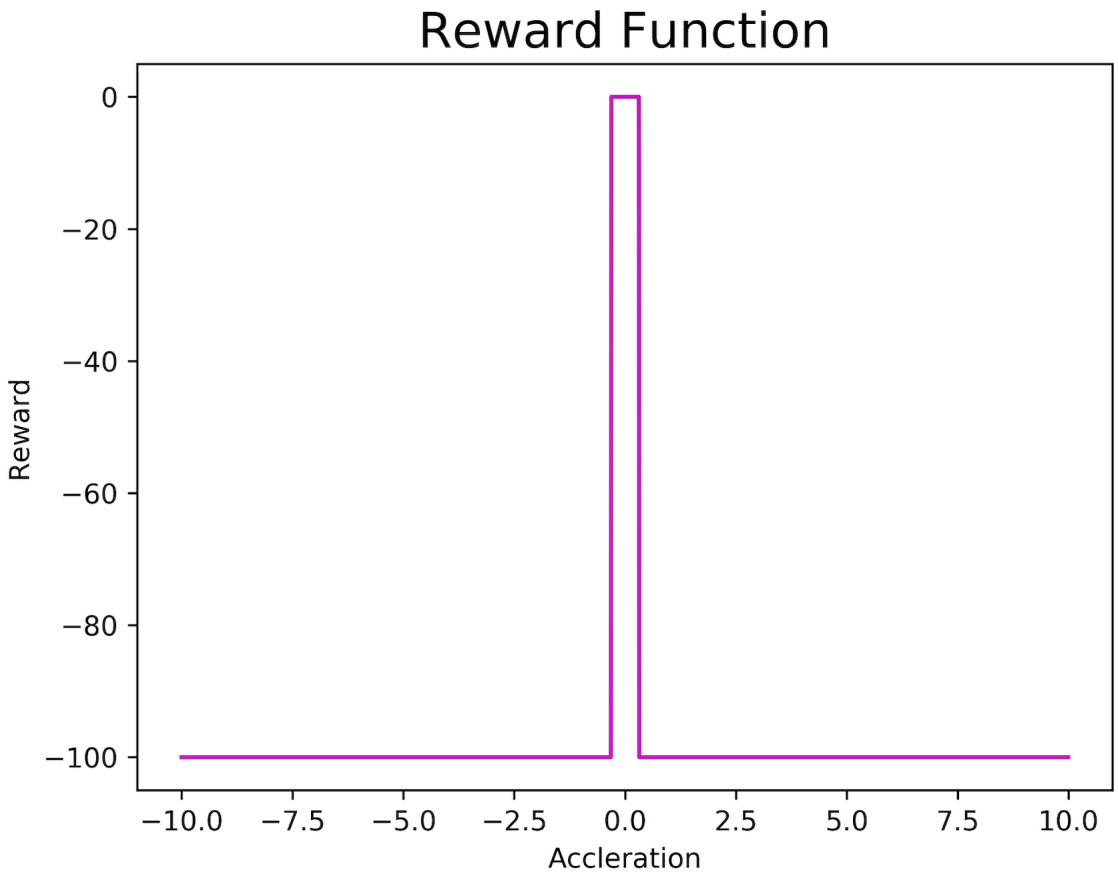

Figure 4.7: Reward Function 2

These two reward functions were compared on the exact same simulation, using the exact same road disturbances and the same set of parameters, which can be seen in Table 4.1

Parameter	Value
Episodes	30000
Critic Learning Rate	0.0000001
Actor Learning Rate	0.0000001
Explore Rate (Epsilon)	0.3
Decay Rate	0.99995
Target Network Update Value (τ)	0.01
Experience Replay Size	20000
Mini Batch Size	256
Discount Factor (γ)	0.9

Table 4.1: Reward Function Testing Parameters

Using these implementations each reward function was tested 3 times. The following results were recorded for each reward function.

Reward Function	Max Acceleration $\pm$0.02	Max Chassis Speed $\pm$0.002	Max Displacement $\pm$0.0001	Avg. No. of Times Boundary Exceeded $\pm$2
1	0.98	0.019	0.0027	296
2	1.03	0.02	0.0028	299

Table 4.2: Reward Functions **Note:** the table shows the average number of times the boundary was exceeded over the three runs. All other measurements have regions of error to accommodate for deviations between results in the 3 tests

As can be seen from Table 4.2, Reward Function 1 outperformed Reward Function 2 in nearly every metric. This may be due to the fact that Reward Function 2 is discontinuous and as such is harder to learn. In contrast, Reward Function 1 is continuous and therefore it is easier for the model to learn.

As such, based on these results Reward Function 1 was deemed to be the most practical and effective reward function to use and as such is the one that all models were trained on.

It is important to note here that as the agent is only authorized to take an action every 35ms, the actual reward received by the agent is the accumulative sum of rewards over the following 35ms from taking the current action.

4.1.4 Terminal Goal

The end of the simulation occurs when a terminal goal is reached. In this case the terminal goal occurs when chassis acceleration falls to below $0.2m/s^2$ for 200 consecutive timesteps. This is done to determine when the chassis has settled down following the road disturbance. When this is achieved the simulation ends.

4.2 Deep Reinforcement Learning Algorithm

This section compares the different deep reinforcement learning techniques applied to this problem and outlines why Residual Policy Reinforcement Learning was deemed to be the best option.

In previous work carried out in [1], attempts were made to apply the DDPG algorithm, described in section 2.3.4, to suspension control in order to replace PID-controllers. However, as can be seen in Figure 4.8, it was found that this led to instabilities in learning.

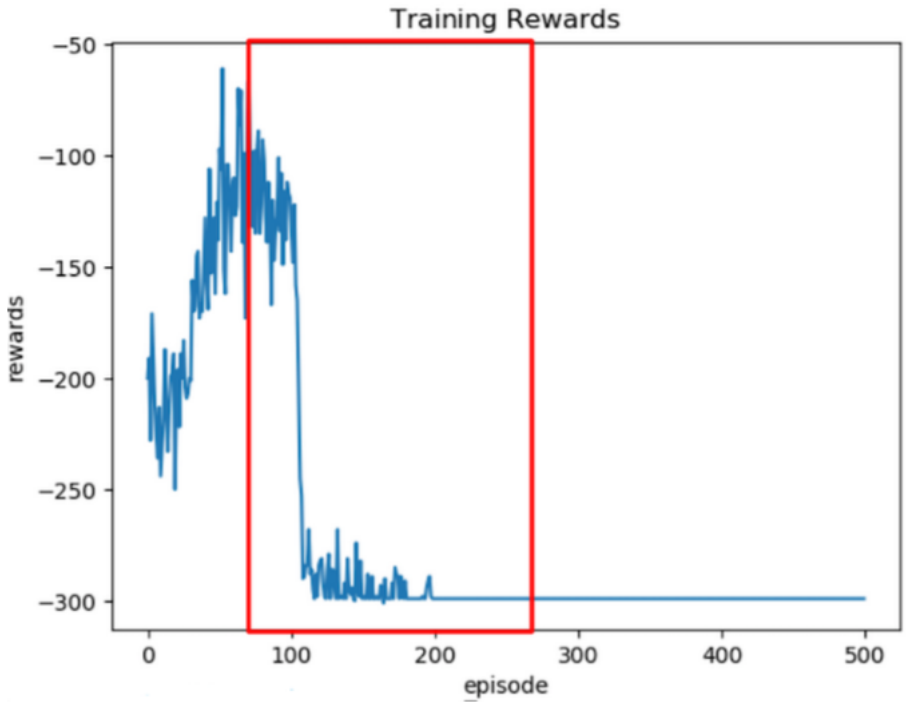

Figure 4.8: Divergence when applying DDPG to suspension control [1] **Note:** the red box indicating the point where divergence occurred

These instabilities often resulted in complete divergence, as seen in Figure 4.8 and this has been attributed to the long and varying horizon which is caused by the use of an end goal, as described in section 4.1.4. This makes it more difficult for the critic to learn an accurate prediction for the Q-value approximation as the length of the horizon varies, therefore this can lead to divergence.

In order to overcome this, the Residual Policy Reinforcement Learning algorithm was chosen. As mentioned in section 3.3, this algorithm works by using a base policy alongside the reinforcement learning policy. This base policy provides guidance to the reinforcement learning policy and also increases the consistency in horizon length and trajectory making it easier for the critic to learn. As a result, and as can be seen in Figure 4.9, this reduces the instability experienced during learning.

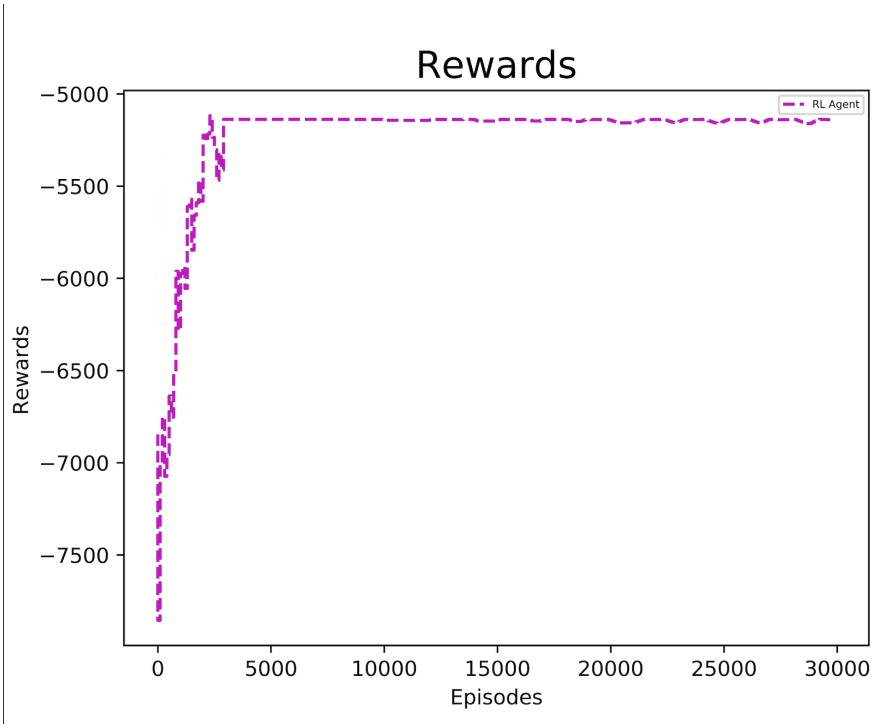

Figure 4.9: Residual Policy Reinforcement Learning Rewards **Note:** the enhance stability

Due to this improvement in stability Residual Policy Reinforcement Learning was chosen for this study. However, it is important to note that the algorithm does not guarantee full stability and the agent can still slip in and out of convergence very slightly, as can be seen with the oscillations in Figure 4.9. As such, early stopping and performance thresholds were used to further enhance the stability and consistency of training, these methods will be discussed in section 4.5 and 4.4 respectively.

4.3 Residual Policy

The Residual Policy Reinforcement Learning algorithm used in this thesis combined a PID-controller as the base policy and a reinforcement learning agent as the residual policy. The PID-controller was tuned for use in the simulation. The reinforcement learning agent was trained using the DDPG algorithm described in section 2.3.4, further details on the training will be provided in section 4.7.

The PID was pre-tuned meaning it was consistent throughout training. As such it was only the RL agent being updated during the training phase.

The final policy was given as the sum of the two polices and was used to decide the action i.e. the damping rate, as described in section 4.1.2.

4.4 Performance Threshold

Whilst the use of Residual Policy Reinforcement Learning improves the stability of the agent during training, as described in section 4.2, there is still slight fluctuation in performance in training, even when the agent is in relative convergence. As such, the use of a performance threshold was introduced whereby if an agent was performing better than the threshold, and the agent was no longer exploring i.e. there was no random actions, then the model was saved. This meant that should the agent slip out of convergence before training was over, there was still a good performing model available.

4.5 Early Stopping

Early stopping is the process of terminating training before the dictated number of episodes are reached. This is done as a way of saving time during training and is usually carried out when the agent has reached convergence. However, it can also be used to improve performance. As mentioned in section 4.4 even though Residual Policy Reinforcement Learning can improve stability during training the agent can still slip slightly out of convergence and end up with a policy slightly below the performance threshold. The use of early stopping helps avoid this by terminating training early, whilst also preventing any unnecessary training when in convergence, thus saving time.

4.6 Deep Neural Network's Architecture

The architecture of the deep neural network is based on the DDPG algorithm described in section 2.3.4 and as such is an actor-critic model as can be seen in Figure 4.10. It is shown here that the actor takes in the state as input and takes an action, which gets passed to the critic and the environment. The critic then takes this action as input, as well as the state, and returns a Q-value approximation to the actor, which is used to update the actor. The critic then receives the reward signal from the environment, which it uses to update itself.

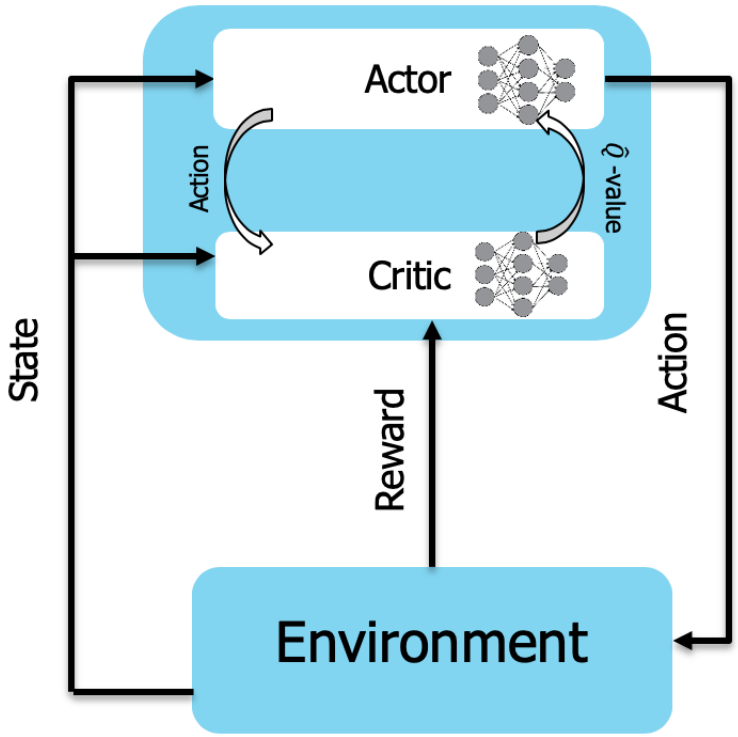

Figure 4.10: Actor Critic's Architecture [1]

The exact architectural design of each network is similar to that described in [21], however there are less hidden layers for the actor in the implementation in this thesis. This is because it was found that the use of two hidden layers, as was recommended in the [21], resulted in no improvement in policy performance compared to a network with one hidden layer, but instead reduced the speed of training due to the extra parameters that needed to be updated.

As a result, the actor's architecture consisted of an input layer, a single hidden layer and an output layer as can be seen in Figure 4.11. The hidden layer was made up of 300 neurons, which is the same as the second hidden layer of the actor network described in [21].

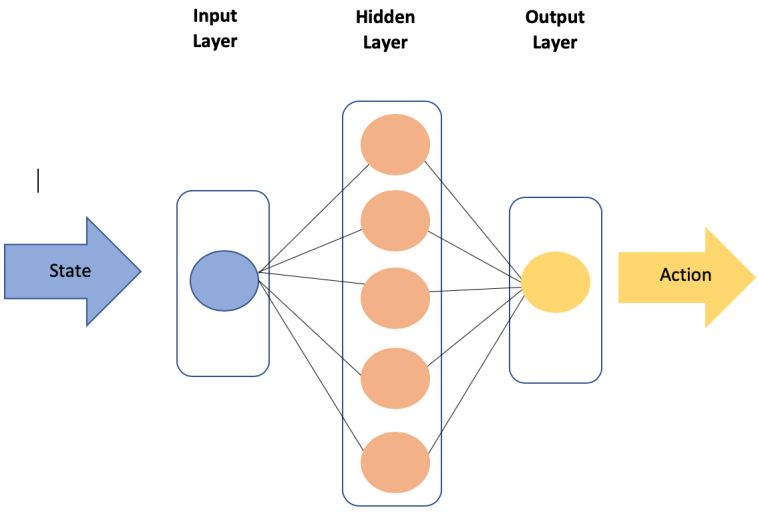

Figure 4.11: Actor's Architecture

The critic's architecture was identical to that described in [21], this consisted of two input layers, two hidden layers and one output layer. This architecture differs slightly from traditional network implementations as the second input layer connects directly to the second hidden layer i.e. it skips hidden layer one. This results in a structure whereby the state is taken as an input in input layer one and passed into hidden layer two. However, the action enters the network at input layer 2 and passes directly into hidden layer 2, where it is combined with hidden layer one and passed through to the output layer. The first hidden layer of the critic was made up of 400 neurons, whilst the second hidden layer was made up of 300 neurons, exactly as described in [21].

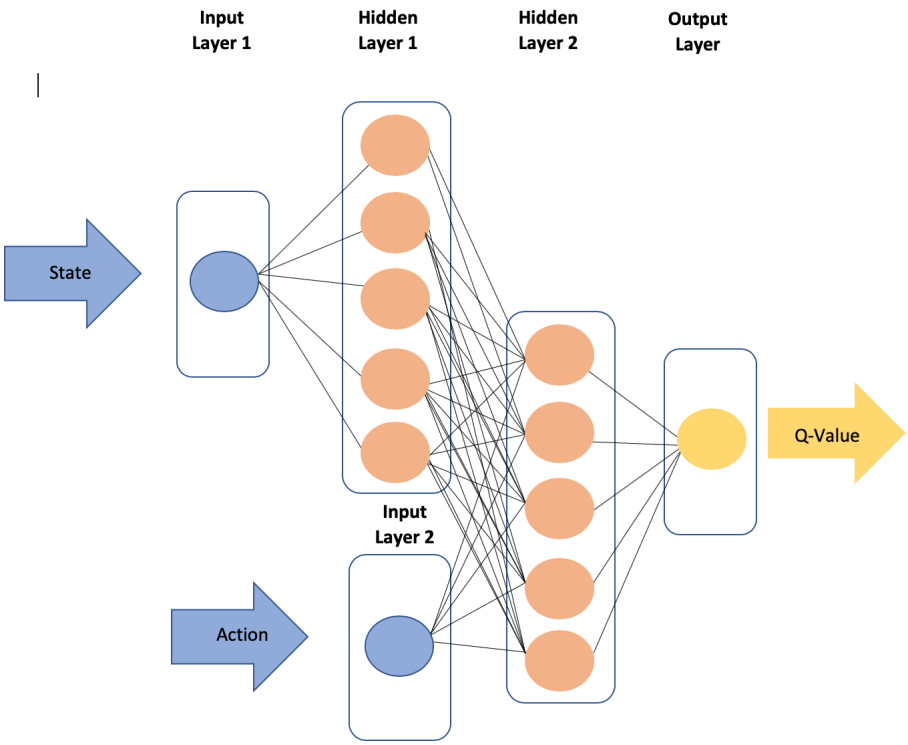

Figure 4.12: Critic's Architecture

4.7 Training

The training for the reinforcement learning agent was the same as that for the DDPG algorithm described in [21]. At each 35ms timestep the agent was allowed to take an action, in the current state, which results in the agent transitioning to a new state and receiving the reward. This action was summed with the action of the base policy before being applied to the suspension control problem. The new state corresponded to the state reached after the agent's action has been applied to 35 consecutive 1ms timesteps. Similarly the reward corresponded to the sum of rewards over the 35ms, as outlined in section 4.1.3. A tuple comprised of the current state, current action, new state and current reward, $< s_t, a_t, s_{t+1}, R_t >$, were stored in a memory buffer so that the agent could relearn from these experiences at a later stage.

Every time the agent took an action, a sample of experiences were randomly selected from the memory buffer and used to create minibatches to train the network. As these minibatches were from old experiences the algorithm can be deemed off policy.

Following each update of the main network the target networks were also updated. These target networks are used to improve the stability of learning as discussed in section 2.3.1

The full pseudocode for the DDPG implementation can be seen in Figure 4.13.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 4.13: DDPG Algorithm [21]

Following training the critic and the two target networks were discarded so that there was only the actor left.

4.8 Summary

This section described the design of suspension control as a RL problem, as well as the state and action spaces, reward function and terminal goal of this RL problem. It also described the additional features used, the training and the architecture of the deep neural network model, as well as the design of the Residual Policy Reinforcement Learning Algorithm.

5 Implementation

This section will describe the exact implementation of the simulation, Residual Policy Reinforcement Learning algorithm, PID-controller and RL agent.

5.1 Simulation Implementation

The simulation was provided by ZF Friedrichshafen AG and followed the form of the OpenAI Gym format [64]. This template is written using the Python programming language and can cater to almost any simulation. In this case, the simulation describes the impact road disturbances and the Residual Policy Reinforcement Learning algorithm's actions have on chassis and wheel acceleration, velocity and position in a quarter car suspension system model. The simulation itself is responsible for creating the road disturbances and it is also responsible for calculating the rewards for each action. There are a number of functions used within the OpenAI Gym's template, these will be discussed now.

- ***init(self)*** - initialises all the variables needed for use within an instance of a gym environment.
- ***step(self, action)*** - creates road disturbances and calculates and describes the impact these road disturbances and the chosen actions have on particular states. It calculates the rewards for these actions in the current state and also calculates the new states. It also monitors the terminal goal to determine when it is reached. If this terminal goal is achieved the function changes a Boolean switch to true. This switch is known as the done variable. It returns the next state, the reward, the done variable and additional information. The addition information variable is null in this implementation. Each time the step() function is called it corresponds to 35 timesteps.
- ***reset(self)*** - resets the environment back to the original state. Returns the initial state value.
- ***render(self)*** - plots the simulation results.

5.2 Residual Policy Reinforcement Learning Algorithm Implementation

The Residual Policy Reinforcement Learning algorithm was made up of two components: the base policy (PID-controller) and the residual policy (DDPG RL Agent). The final policy was the sum of these two. The interaction of the Residual Policy Reinforcement Learning algorithm and the OpenAI Gym environment, described in section 5.1, can be seen in Figure 5.1.

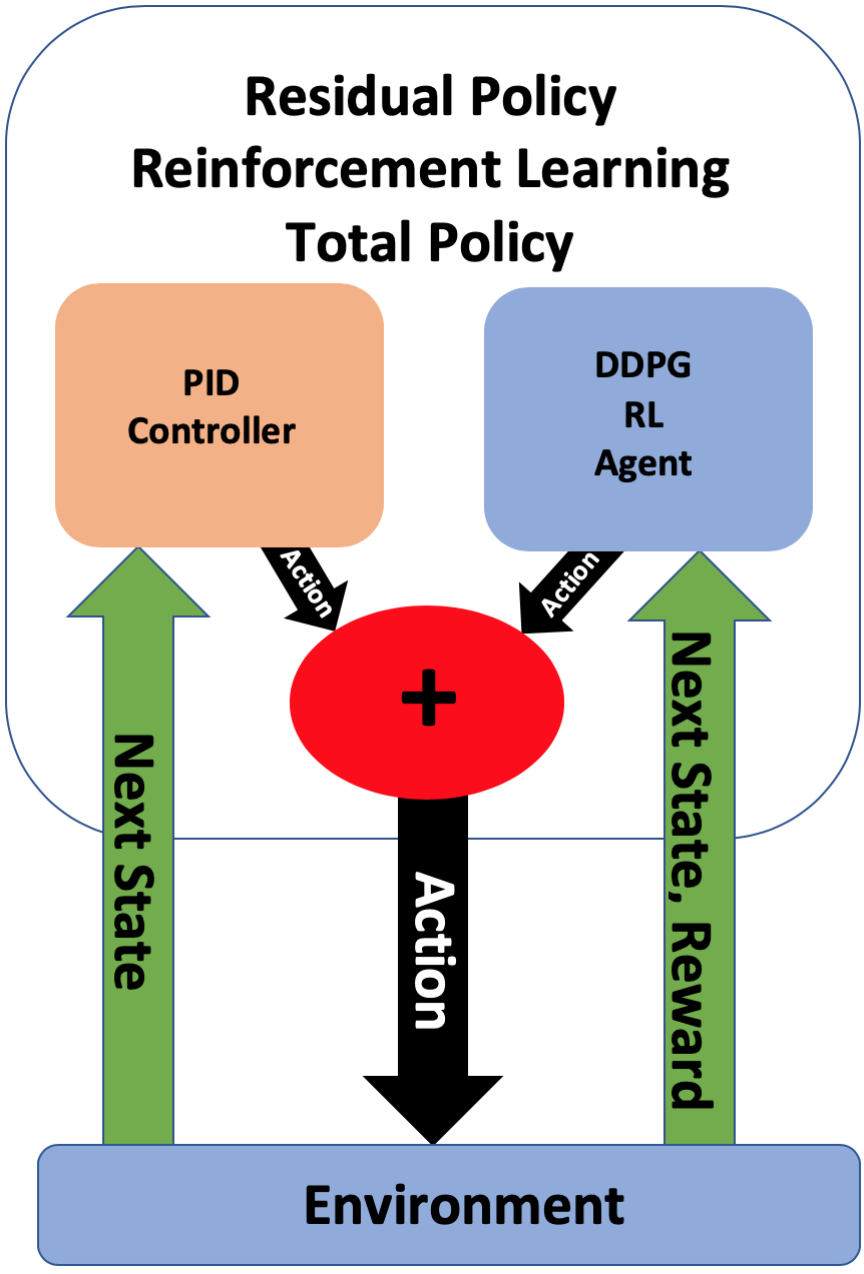

Figure 5.1: Residual Policy Reinforcement Learning Algorithm Interaction with the Environment

As can be seen from Figure 5.1 the final action passed to the environment is the sum of the two actions from the PID-controller and the DDPG RL agent. This is passed to the environment through the ***step(self, action)*** function described in section 5.1. In turn, the environment returns a number of variables, namely the next state and the reward. Note that the reward is only used by the DDPG RL agent, as the PID-controller does not update during training and as such has no use for the reward. The done variable, which is described in section 5.1 and describes if the terminal goal is completed, is not passed to either the PID-controller or the RL agent, but is instead used separately to determine if the episode is complete.

The full system is implemented using Python. The exact implementation of each component will now be discussed in section 5.3 and section 5.4.

5.3 PID

The PID-controller, as described in section 2.6, is implemented in Python. In this implementation the PID-controller outputs an acceleration value which can then be used to calculate a K-value or damping rate. This K-value corresponds to the action which is passed to the final policy.

The controller itself is not updated during training as it is pre-optimized using a brute force algorithm which tests the performance of each gain value in the range [-1000,1000], with iteration size 0.1. Each gain is optimized separately to save computational time.

The final result of this optimisation is a set of PID gains outlined in Table 5.1

Gain	Value
Proportional	0
Integral	0
Derivative	0

Table 5.1: PID Gain Values

These values can be considered to be somewhat odd due to the fact, and as described in section 2.6, these gains are multiplied by the operation in each of the sections of the

PID-controller. As such, there will be a constant output of 0 from the PID-controller, which seems to be unusual.

However, it is believed that this may be the result of the tuning method, which iteratively tries the different gain values to see which gain values perform the best. Each gain is tuned separately as opposed to simultaneously. As previously stated, this was done to save time as tuning PID-controllers can be quite tedious. In future work it may be advisable to use a better tuning method such as the Ziegler and Nichols method [23].

Nonetheless, these values did exhibit good performance and were tested and found to outperform passive systems, which are suspension systems with fixed damping rates. Therefore they were deemed acceptable for this research.

5.4 Reinforcement Learning Agent

This section describes the RL agent's implementation in Tensorflow, as well as giving an insight into the parameter's chosen and the activation functions and optimisation algorithms used.

5.4.1 Tensorflow Implementation

The reinforcement learning agent is built using Tensorflow with a Keras backend and is implemented using a Python class. The implementation involves the construction of two actor models following the architecture described in section 4.6, with one of these models acting as the target network for the actor and the other being the actor itself. Similarly, the critic model was implemented using the same architecture as described in section 4.6. Again two networks were created using this architecture, one being for the critic and the other being for the critic's target network.

The implementation contains the following functions:

- ***init(self, TensorflowSession, Environment, CriticLearningRate, ActorLearningRate, NumberEpisodes, Exploration, ExplorationDecay, FolderName, MaxActionRange, MinActionRange, ActivationFunction)*** - Responsible for initializing an instance of the class with the given parameters.
- ***create_actor(self, action, trainable)*** - creates the actor network
- ***create_critic(self, action, state, trainable)*** - creates the critic network
- ***train(self)*** - updates both critic and actor network
- ***store(self, CurrentState, Action, Reward, NewState)*** - stores experiences in the replay buffer

- ***actor_target_update(self)*** - updates the actor's target network
- ***critic_target_update(self)*** - updates the critic's target network
- ***act(self,State)*** - triggers the actor to take an action. Returns the action

This results in a tensorflow graph as can be seen in Figure 5.2

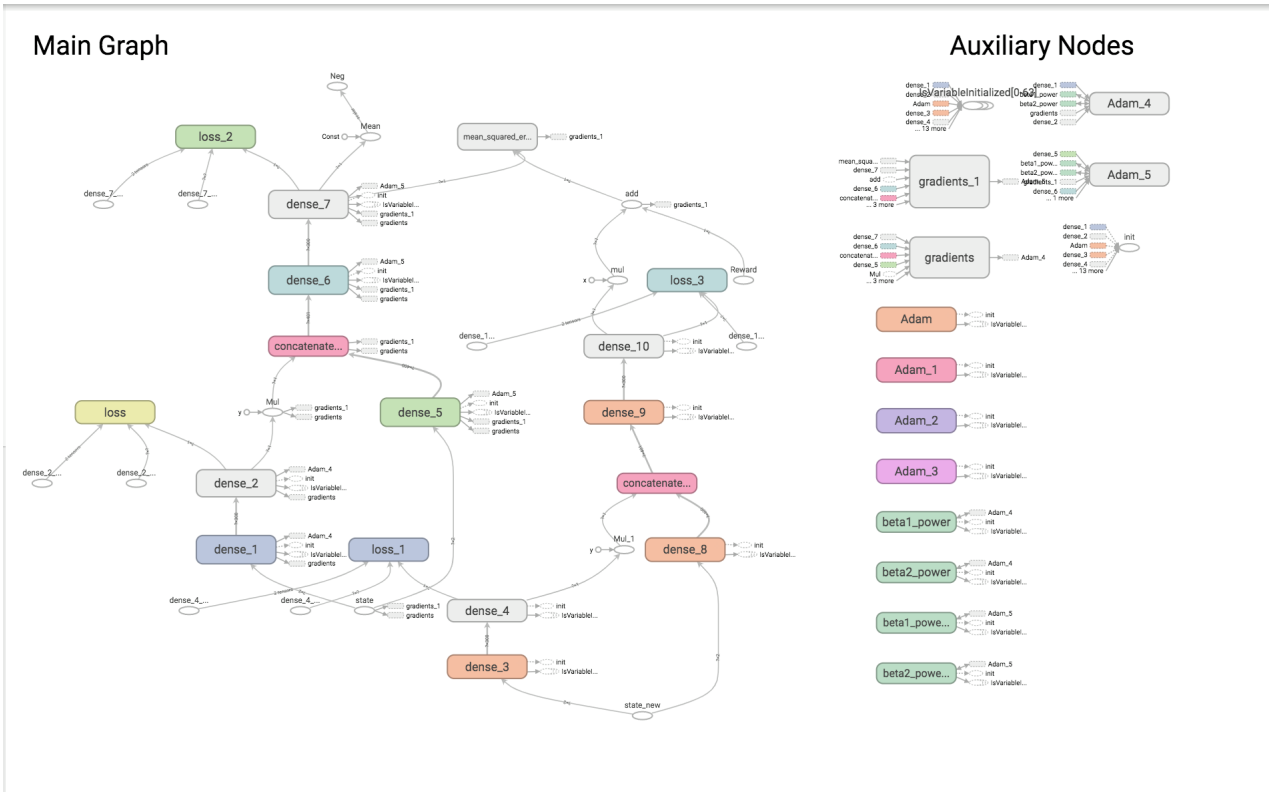

Figure 5.2: Tensorflow Graph

As can be seen this diagram is quite complicated, therefore a clearer version can be seen below in Figure 5.3 and Figure 5.4. These figures essentially break the full Tensorflow graph down into two simpler sub graphs which show the main actor-critic model in Figure 5.3, and the target actor-critic networks in Figure 5.4.

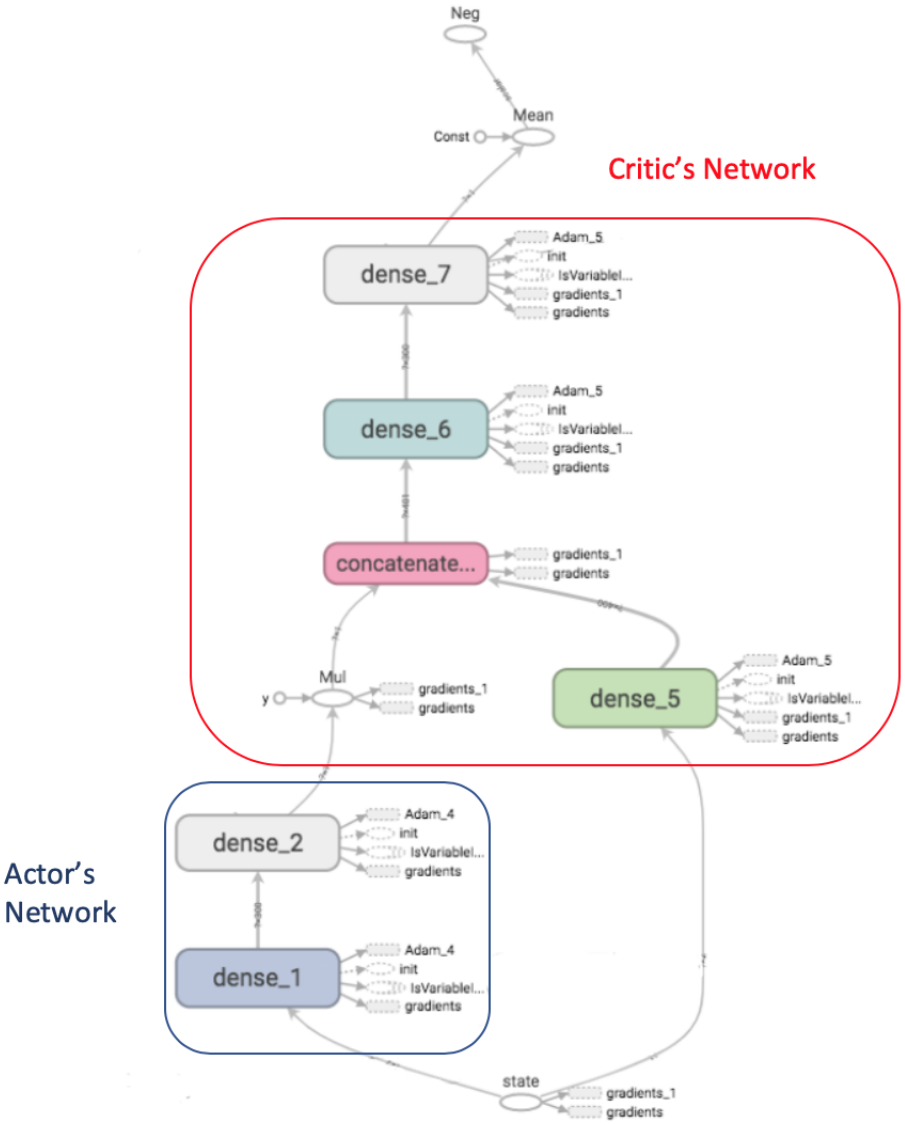

Figure 5.3: Main Actor Critic Network Tensorflow Graph

As can be seen from Figure 5.3 the actor network is composed of two dense layers, whilst the critic makes use of three dense layers and a concatenate layer. The concatenate layer is responsible for combining the first dense layer, which is the same as hidden layer 1 in Figure 4.12, with the action which comes directly from the actor network. It can also be seen that the state is passed into both the actor, through dense_1, and the critic, through dense_5.

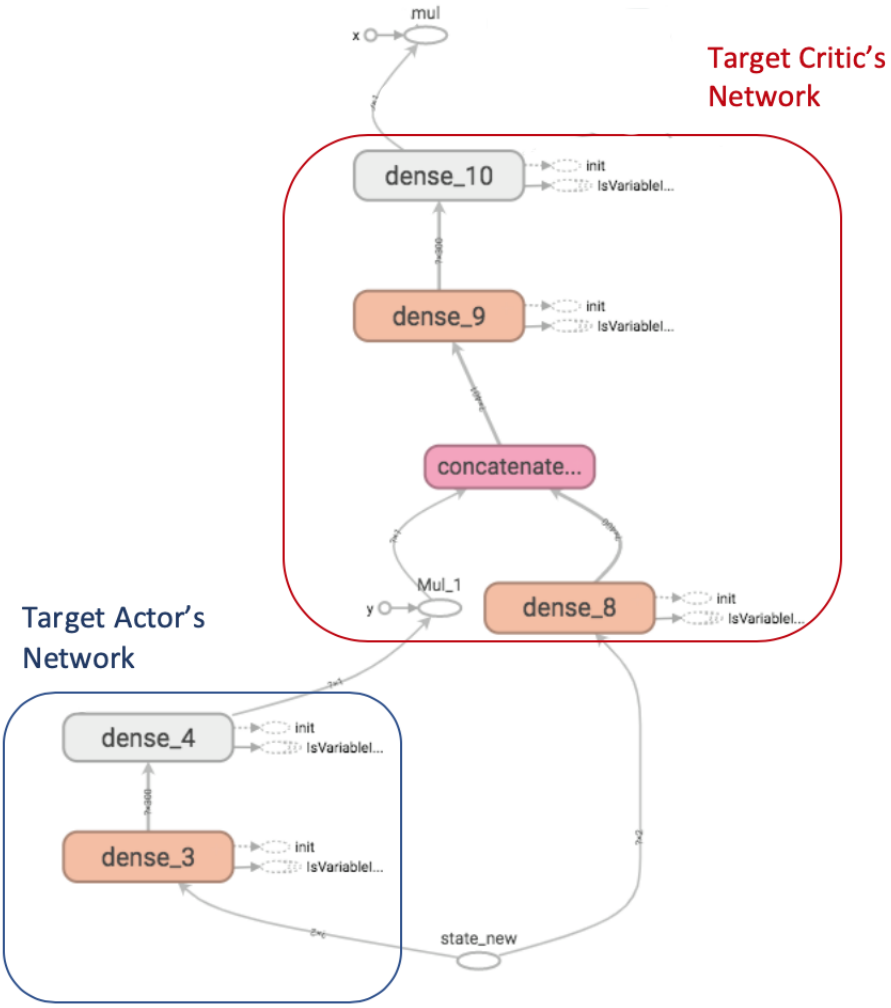

Figure 5.4: Target Actor Critic Network Tensorflow Graph

Figure 5.4 is almost identical to that seen in Figure 5.3, however, the big difference is that the target networks receives the new state as input as opposed to the current state. This is due to the fact that the target network's main job is to stabilise temporal difference learning, and it does this by predicting the future reward based on a new state and a predicted new action. As with the main network seen in Figure 5.3, this new action is provided by the actor's target network.

5.4.2 Hyperparameters

There was a high number of hyperparameters in this implementation. These hyperparameters include:

- ***Episodes*** - the number of episodes the agent will be trained on
- ***Critic Learning Rate*** - the step size for the critic's updates (too small and it takes too long to learn, too big and it could miss a minimum)

- ***Actor Learning Rate*** - the step size for the actor's updates (too small and it takes too long to learn, too big and it could miss a minimum)
- ***Exploration Rate (ϵ)*** - Dictates the probability of taking an exploratory action i.e. a random action to explore the environment. It is a decimal value in the range [0,1].
- ***Decay Rate*** - dictates how much the exploration rate should be reduced by at each timestep. It too is a decimal value in the range [0,1]. At each time step it is multiplied by the exploration rate to give a new exploration rate.
- ***Target Network Update Value (τ)*** - a decimal value dictating how large target network updates should be in comparison to main network updates. Usually very small and in the lower range of [0,1].
- ***Experience Replay Size*** - the size of the buffer used to store memories or experiences.
- ***Mini Batch Size*** - the number of experiences sampled at each update
- ***Discount Factor (γ)*** - the temporal difference value dictating how much of reduction is placed on each future reward. It is in the range [0,1]

As with most reinforcement learning algorithms, a DDPG agent is highly sensitive to hyperparameter values. As such, a substantial amount of time was spent iteratively testing a variety of different values for each hyper-parameter. Table 5.2 shows the different hyperparameters and the values assigned to them. It also shows the range and incremental value used to tune the hyperparameter. For example, a hyperparameter trained between 0 and 5, tested using only integers, would have a range of [0,5] in the table and incremental value 1, indicating the values 0,1,2,3,4,5 were tested.

Note: Any value in the incremental column with an Asterix beside it indicates the number was multiplied as opposed to added on to the original value. For example, for a range [1,100] with increment size *10, this means values 1, 10 and 100 were tested.

Hyper-Parameter	Value	Range	Incremental Value
Episodes	30000	[500,30000]	500
Critic LR	0.0000001	[10^{-10},1]	*10
Actor LR	0.0000001	[10^{-10},1]	*10
Exploration Rate	0.3.	[0,1]	0.1
Decay Rate	0.99995	[0.9,0.9999999]	See *Note*$_1$
Target Network τ	0.01	[0.01,0.1]	0.01
Experience Replay	20000	[2000,2000000]	*10
Mini Batch Size	256	[64,512]	64
Discount factor γ	0.9	[0.7,1]	0.02

Table 5.2: Hyperparameter values, tuning ranges and step sizes **Note:** any value in the incremental column with an Asterix beside it indicates the number was multiplied as opposed to added on to the original value. For example, for a range [1,100] with increment size *10, this means values 1, 10 and 100 were tested.

Note$_1$:the decay rate was tested using the value set {0.9,0.95,0.99,0.995,0.999,0.9995,0.9999,0.99995,0.99999,0.999995, 0.999999, 0.9999995, 0.9999999}

5.4.3 Activation Functions

Another important aspect of the implementation was the activation function. As described in section 2.2.3, there are a wide variety of possible activation functions. In this implementation the ReLU activation function was chosen for all hidden layers, as it avoids the exploding gradient and vanishing gradient problems. Whilst the critic networks used a linear output function. This is in line with the implementation described in [21] and [9].

However, for the output layer of the actor it was harder to determine the best activation function. This was because both [21], the paper that proposed the DDPG algorithm, and [9], the paper that proposed the Residual Policy Reinforcement Learning algorithm, used two different implementations. [21] used a zero centred TanH function, which assumed the action space was zero centred and symmetric, whilst [9] used a simple linear output. In order to determine which implementation was best each was applied to the suspension control problem and compared.

The linear function was simple to implement, however, the TanH function outputs a value in the range (-1,1). Therefore, to successfully implement Tanh, it was necessary to create a zero centred symmetric action space. In this case a number of different action spaces were

tested including: [-1000,1000], [-2000,2000], [-3000,3000], [-4000,4000] and [-5000,5000]. It was found that the range [-5000,5000] was optimal, which makes relative sense as the action space allowed by the suspension system is [100,5000]. Therefore, by having this output range, the actor had the ability to completely change any output from the PID-controller. For example, if the PID-controller outputted 5000 the RL agent could output -5000, bringing the total output to zero, which would be clipped to a damping rate of 100. Equally, the RL agent could make a small PID-controller output become very large in the final policy.

Both the linear and the TanH, with output range [-5000, 5000], activation functions were used to train 3 separate models and compared against each other. 3 models were trained in order to ensure the performance of each function was reproducible. Based on these tests the TanH activation function was found to outperform the linear function. This was due to the fact the linear function repeatedly converged to an output of zero, meaning the RL agent contributed nothing to the total policy, and performance was based solely on the PID-controller's behaviour. The TanH activation function on the other hand, showed the ability to make small adjustments to the overall policy, which was shown to improve performance.

Table 5.3 gives a breakdown of the activation functions used.

Model	**Hidden Layer**	**Output Layer**
Actor	ReLu	TanH
Critic	ReLu	Linear

Table 5.3: Implementation: Activation Functions

5.4.4 Optimisation Algorithms

In order to choose the correct optimisation algorithm a number of potential algorithms were tested, namely: Adam, RMS_Prop and Stochastic Gradient Descent (SGD). Each algorithm was used to train 3 models and the performances compared against each other. Again, 3 models were trained using each algorithm to ensure reproducibility. The results can be seen in Figures 5.5, 5.6 and 5.7. Each of these graphs are plotted alongside the rewards achieved by the PID-controller alone as reference.

Figure 5.5: Implementation: Adam Optimizer

Figure 5.6: Implementation: RMS_Prop Optimizer

Figure 5.7: Implementation: Stochastic Gradient Descent Optimizer

As can be seen from these figures, the Stochastic Gradient Descent algorithm is by far the worst performing algorithm with a massive divergence at approximately Episode 24,000. Therefore, it was discarded as a potential option. The Adam optimisation algorithm and RMS_Prop optimisation algorithm both performed quite well with only small differences between them. Therefore the Adam optimisation algorithm was chosen for this implementation as this was the one chosen in [21].

5.4.5 Initialisation

One important aspect of the Residual Policy Reinforcement Learning algorithm implemented in [9] was the initialisation of the networks. Here, the actor network's output was set to zero at the beginning of training so as to allow the actor to experience the rewards associated with the PID-controller alone. This was done in an attempt to prevent the algorithm diverging too much away from the PID-controller in the beginning, especially if this divergence was causing the overall policy's performance to deteriorate. A similar approach was adopted in this implementation with the actor's output layer being initiated to zero.

[9] also introduced a wear in period whereby only the critic was trained for an initial number of episodes. This was done so the critic could learn a relatively accurate approximation of the Q-value before it was used to update the actor. This is important as the actor relies on the critic to learn itself. As such, waiting a number of episodes before updating the actor is thought to improve overall performance. A similar implementation was used in this thesis.

5.4.6 Performance Threshold and Early Stopping

As described in section 4.4 and 4.5 two additional features were used to improve the agent's overall performance and reduce training time, these were early stopping and a performance threshold. The exact implementation of these will be discussed now.

For the performance threshold the performance of the PID-controller alone was used. This meant that once the Residual Policy Reinforcement Learning algorithm's performance surpassed the PID-controller's performance the model would be saved. At this point the threshold would be updated to the performance of this model i.e. the performance threshold will always be the PID threshold or that of a better performing model. This meant that during training the best performing model would always be saved and this would protect against the possibility of divergence.

For the early stopping implementation, early stopping would occur following 15 consecutive episodes of the same reward. The same reward was deemed to mean two reward totals within 0.5 of each other.

5.5 Summary

This section described the exact implementation of the suspension environment and the Residual Policy Reinforcement Learning algorithm including each of the individual components i.e. the PID-controller and the reinforcement learning algorithm. It also described the hyperparameters, activation functions and optimisation algorithms used in this implementation, as well as the initialisation of each network and the critic's wear in period. The implementation of the performance threshold and early stopping techniques were also outlined.

6 Evaluation

This section describes the evaluation of the Residual Policy Reinforcement Learning algorithm. This includes outlining the objectives of the evaluation, a description of the metrics used to analyse performance and a specification of the road disturbances used to test the performance. In addition, the results themselves will also be presented and an analysis will be provided.

6.1 Objectives

The objective of this evaluation is to highlight the ability of the Residual Policy Reinforcement Learning agent to enhance the performance of the PID-controller on a number of metrics. As outlined in section 1, PID-controllers are quite tedious to tune and tuning techniques often disagree on the optimum parametric values, as such optimal performance is rarely achieved. Performance can also be impacted by non-linearities in the system. Hence, one of the objectives is to compensate for these problems by using the RL agent in the policy to improve the overall performance. In addition, PID-controllers do not adapt well to environmental change due to the sensitivity of the parameters in specific environments. Therefore, it is also the objective of this research to show how Residual Policy Reinforcement Learning can overcome this by allowing the RL agent to re-adapt and re-optimise performance. Finally, it is also the objective of this work to try to demonstrate the ability of the Residual Policy Reinforcement Learning algorithm to reduce the need for tedious tuning of PID-controllers by allowing it to try improve and optimise the performance of an untuned PID-controller.

These objectives are outlined below:

1. To show that the Residual Policy Reinforcement Learning algorithm is capable of enhancing a PID-controller's performance on a number of metrics which will be described in section 6.2 and improve the ride comfort of the vehicle.
2. To show that the Residual Policy Reinforcement Learning algorithm has the ability to adapt and re-optimise the PID-controller's performance following changes to the

environment.

3. To show that the Residual Policy Reinforcement Learning algorithm has the ability to optimise the performance of untuned PID-controllers, hence reducing the need for tedious tuning methods.

6.2 Metrics

As mentioned previously, ride comfort is directly related to the chassis acceleration, velocity and position. However, the chassis acceleration is particularly important as the velocity and position are derived from this. This means that if low acceleration can be achieved then by extension there will be very little velocity or movement in the chassis. As such the metrics used here mainly focus on acceleration, however, maximum chassis velocity and position are also considered. The metrics are as follows:

- Maximum Chassis Acceleration
- Maximum Chassis Velocity
- Maximum Chassis Movement
- Reward Function Score
- ISO 2631 Riding Comfort Standard

The final metric, the ISO 2631 Riding Comfort Standard, describes how the chassis acceleration relates to passenger comfort. It groups acceleration ranges into categories of comfort. The full breakdown can be seen in Figure 6.1

RMS (Root Mean Square) Acceleration	Ride comfort Criteria
$a<0.315\ m/s^2$	Comfortable
$0.315<a<0.63\ m/s^2$	A little Uncomfortable
$0.5<a<1 m/s^2$	Fairy Uncomfortable
$0.8<a<1.6 m/s^2$	Uncomfortable
$1.25<a<2.5 m/s^2$	Very Uncomfortable
$a>2.5 m/s^2$	Extremely Uncomfortable

Figure 6.1: Evaluation: ISO 2631 Ride Comfort Standard [20]

Performance for this metric is evaluated based on the algorithm's ability to reduce the number of times the ride comfort enters the lower comfort bands e.g. "Very Uncomfortable" or "Extremely Uncomfortable".

6.3 Evaluation Scenario

This section describes the scenarios used to evaluate the Residual Policy Reinforcement Learning algorithm. This includes describing the techniques used for comparison purposes and the different road disturbances used for testing.

6.3.1 Evaluation Techniques

The performance of the Residual Policy Reinforcement Learning algorithm is compared against a PID-controller operating by itself. This PID-controller was pre-tuned on this environment as per the description given in section 5.3. This resulted in a PID-controller with the gains as follows:

Gain	Value
Proportional	0
Integral	0
Derivative	0

Table 6.1: Evaluation: PID Gain Values

6.3.2 Evaluation Scenarios

The following list outlines the scenarios that will be tested.

- **Sine Wave** - The algorithm is tested on a sine wave, which is common practice and was used in [65, 66]. In this case the sine wave form was used to resemble a divot in the road. Figure 6.2 shows the exact road disturbance used. Two separate sine wave sizes were used, one at 2.5cm and one at 5cm in depth.

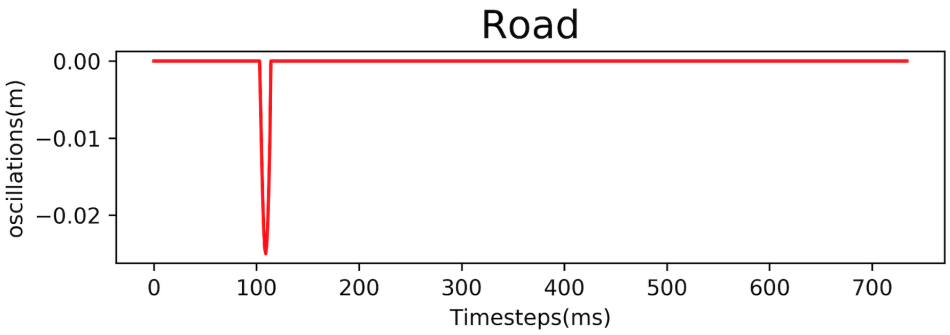

Figure 6.2: Evaluation: Sine Wave Road Disturbance

- **Pot Hole** - The algorithm was also tested on a pothole which is slightly sharper than the sine wave used above. Again this is common practice for suspension systems with similar disturbances being used in [67, 68]. Figure 6.3 shows the exact road disturbance used. Again, two different depths were used for testing: 2.5cm and 5cm.

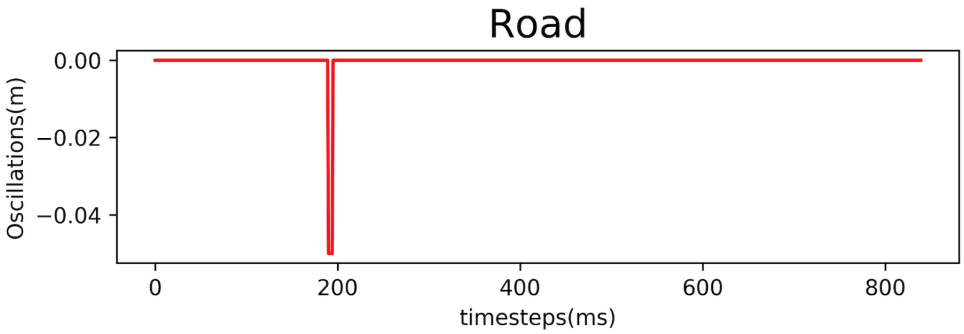

Figure 6.3: Evaluation: Pot Road Disturbance

- **Hybrid** - The algorithm is also tested on a hybrid disturbance, which is a mix of a sine and step function. Figure 6.4 shows the exact road disturbance used. Again, two different depths were used for testing: 2.5cm and 5cm.

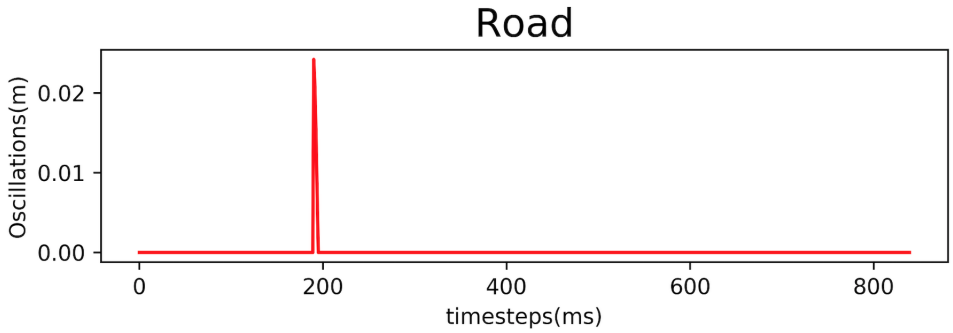

Figure 6.4: Evaluation: Hybrid Road Disturbance

- **Mass Change** - The algorithm was also tested on its ability to deal with environmental change, which was simulated by altering the mass of the chassis. This

mass change was from 650kg to 600kg which is similar to unloading a vehicle for example. See $Note_2$. This was tested on the hybrid road disturbance seen in Figure 6.4. This test was only conducted on a road disturbance of height 2.5cm.

- **Untuned PID** - Finally the algorithm was also tested on its ability to optimise the performance of an untuned PID-controller. In order to do this the PID gains were randomly selected and will be specified in section 6.5.5. Again the model was tested only on a 2.5cm hybrid road disturbance.

Note$_2$ - the reason a mass reduction was chosen as opposed to introducing a higher mass was due to the dynamics of the vehicle. Increasing the mass in this simulation was shown to reduce the chassis acceleration due to the equations used. As such it would be too easy for the algorithm to adapt. Hence why mass reduction was used.

6.4 Setup

This section describes the environment used.

6.4.1 Suspension Model

The suspension system used here is known as a quarter car model and corresponds to a simulation of a single wheel and suspension system encountering road disturbances i.e. bumps. Due to intellectual property restrictions, as agreed with ZF Friedrichshafen AG, the exact implementation cannot be discussed, however, [60, 61, 62] all use similar simulations. Figure 6.5 shows the mechanics of such a system. Interestingly, in these models the wheel is itself described as its own suspension system, with spring constant k_t and damping value b_t. m_u describes the wheel's mass. This is due to the wheel's mechanical properties which are similar to those exhibited by a spring and damper system.

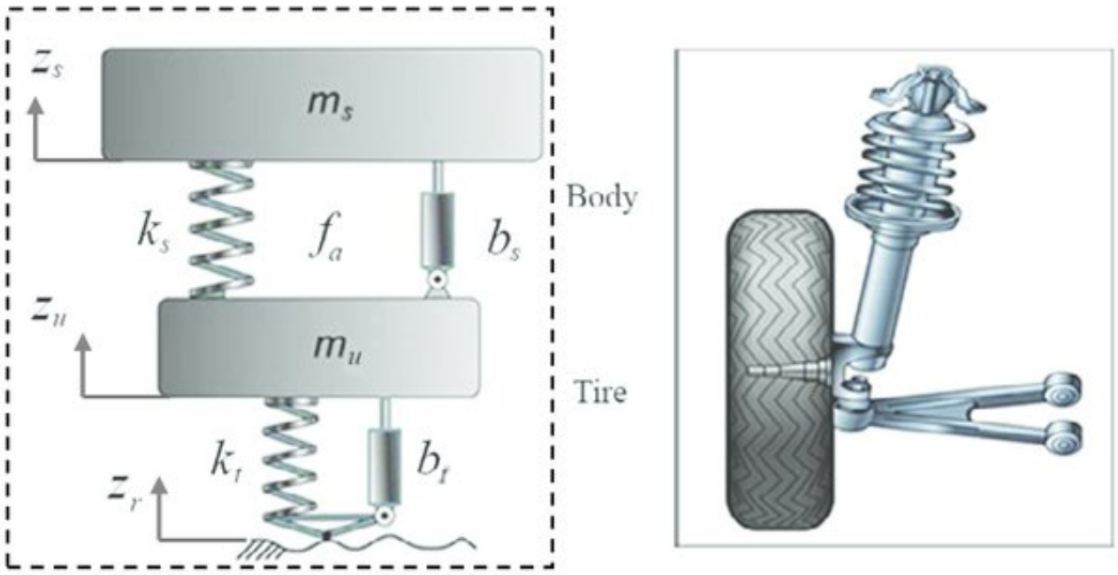

Figure 6.5: Evaluation: Quarter Car Simulation Model [22]

6.4.2 Simulation Parameters

The simulation, whilst confidential, does allow for certain parameters to be changed. This includes the masses of the chassis and wheel and the road disturbances. Using these parameters different road disturbances and chassis masses can be tested.

6.5 Results and Analysis

In this section the results, and an analysis of those results, is presented. All evaluation scenarios described in section 6.3.2 will be discussed and the performance of the model in each scenario will be considered based on all the metrics described in section 6.2. The hyperparameters used are the same as those outlined in Table 5.2, any changes to these hyperparameters will be outlined and justified where necessary. In addition, all graphs used in this section describe the systems behaviour over the course of a full episode i.e. graph lengths are not reduced. As the models can take different times to reach the terminal goal, as described in section 4.1.4, the episode end is determined by the time it takes the RL model to complete the task. As such, in all graphs shown it is possible that the PID-controller is still working to achieve the terminal goal after the timesteps specified in the graphs.

6.5.1 Sine Wave

The first test conducted involved training and testing the algorithm on a sine wave road disturbance, which mimics a divot in the road. Two separate simulations were used, the first simulation showed divot depth of 2.5cm whilst the second simulation showed a divot depth of 5cm. Each experiment, which involved training the algorithm and testing it on the different environments, was carried out 9 times each to ensure reproducibility. The performance of the algorithm in each of the two simulations will be discussed now.

Sine Wave - 2.5cm

The algorithm was trained for 30,000 Episodes on a simulation of a 2.5cm sine wave. This was done 9 times to ensure reproducibility. Table 6.2 shows the performance of the algorithm, with error ranges indicating the variations in performance in each model. The reward function is also shown to indicate the algorithms ability to increase the rewards received, which indicates improved behaviour. This algorithm is compared directly with the performance of a PID-controller.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	1.4	1.370$\pm$0.005
Maximum Chassis Velocity (m/s)	0.027	0.026
Maximum Chassis Movement (m)	0.0041	0.0040
Reward	-5822	-5799 $\pm$15

Table 6.2: Evaluation: 2.5cm sine wave results

Using the ISO 2631 ride comfort standard, seen in Figure 6.1 the distributions of accelerations can also be grouped into different comfort bins. The number of timesteps each model spent at each comfort level can be seen in Table 6.3

Comfort Level	PID-controller	Residual Policy RL
Comfortable	588	554±1
Little Uncomfortable	32	29±1
Fairly Uncomfortable	73	76±1
Uncomfortable	28	31±1
Very Uncomfortable	11	10
Extremely Uncomfortable	0	0

Table 6.3: Evaluation: 2.5cm sine wave results as per ISO 2631 ride comfort standard **Note:** The total number of timesteps are different between the two models. This is because it takes the PID-controller longer to achieve the goal and hence requires more timesteps.

As can be seen from the results in Tables 6.2 and 6.3 the performance between the two models i.e. the PID-controller and the Residual Policy Reinforcement Learning algorithm is very similar with the PID-controller performing slightly better on the ISO 2631 standard. However, the Residual Policy Reinforcement Learning algorithm is capable of reducing the maximum chassis acceleration, velocity and movement, which all indicate improvements in performance. In addition, it has also been able to outperform the PID-controller in terms of the reward received and has reduced the number of timesteps the chassis spends in the "Extremely Uncomfortable" comfort bin.

Other improvements can be seen in Figure 6.7. This plot highlights the difference in magnitude between the PID-controller and the Residual Policy RL algorithm. Negative values on this graph indicate that the Residual Policy RL algorithm has a smaller acceleration than the PID-controller. As can be seen, the majority of this graph is below zero suggesting the PID-controller has larger acceleration magnitudes for most of the episode.

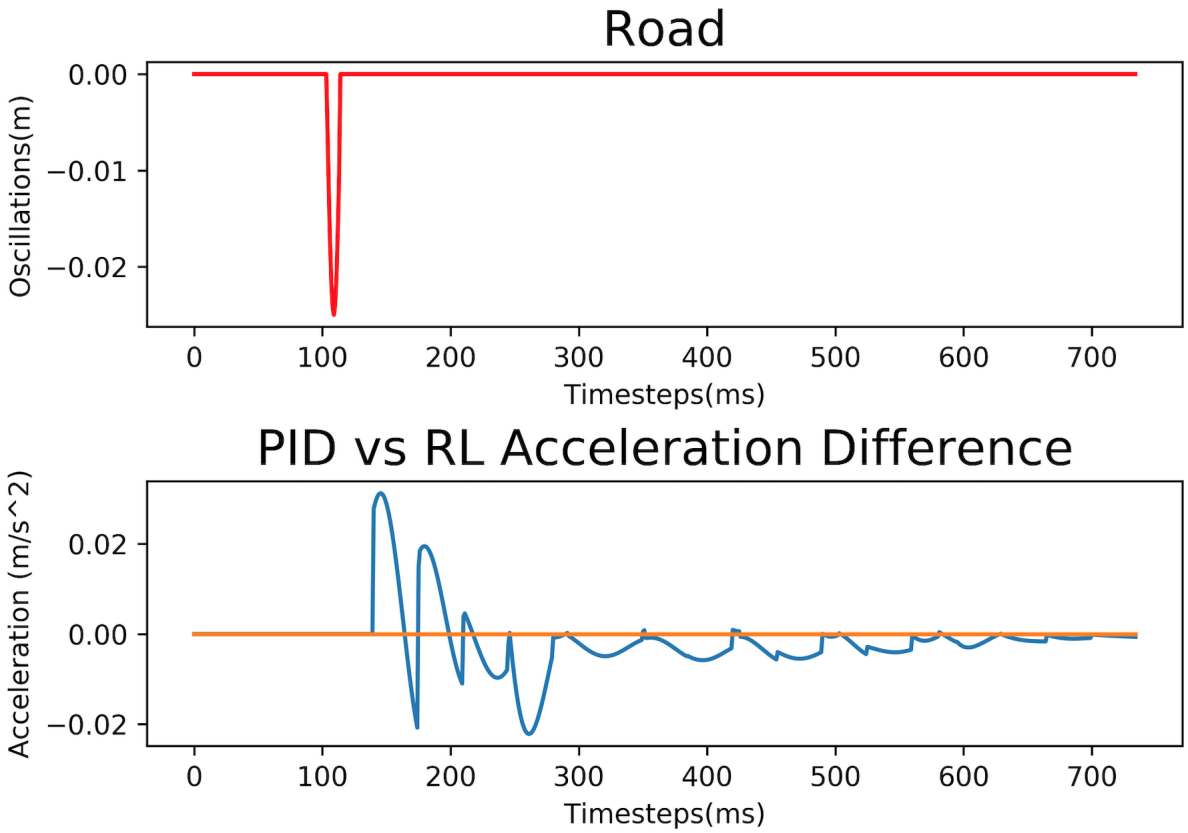

Figure 6.6: Evaluation: 2.5cm sine wave - acceleration magnitude differences. Negative readings indicate the Residual Policy RL algorithm has lower accelerations than the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

This can also be seen in Figure 6.7 where the exponential cumulative acceleration for the PID-controller is far larger than the Residual Policy Reinforcement Learning algorithm indicating reduced acceleration on the Residual Policy RL algorithm's behalf throughout the episode.

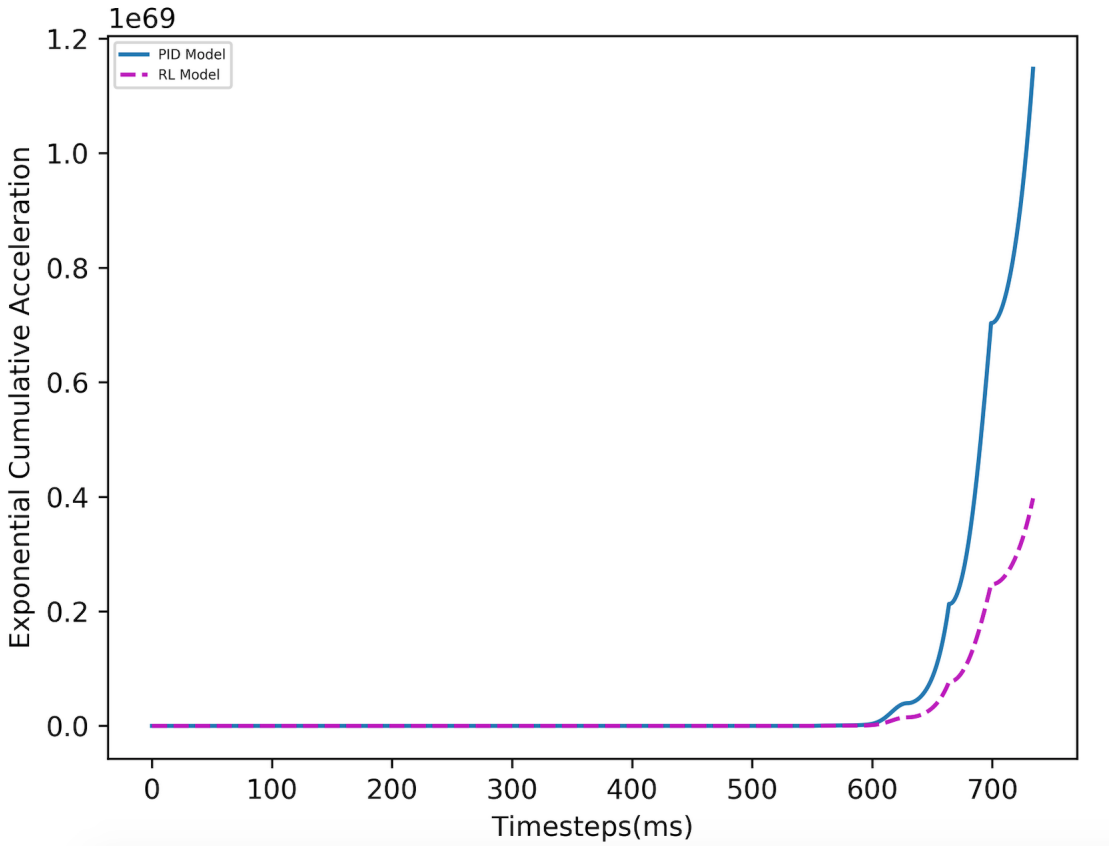

Figure 6.7: Evaluation: 2.5cm sine wave - Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller. **Note:** over time the PID experiences far more acceleration **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

However, all these differences are very small and even though the PID-controller does experience more acceleration over the course of the episode, it also performs arguably better on the ISO 2631 standard. As such it is hard to definitively state the Residual Policy RL algorithm shows clear improvements over the PID-controller in this test.

Sine Wave - 5cm

This exact experiment was then repeated using the exact same road disturbance, however, on this occasion the road divot's depth was set to 5cm. Table 6.4 shows the differences between the two models on the metrics related to maximum chassis acceleration, velocity and position, as well as the overall reward received.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	2.4	2.36±0.02
Maximum Chassis Velocity (m/s)	0.048	0.0455 ±0.0003
Maximum Chassis Position (m)	0.0071	0.0067 ±0.0001
Reward	-14075	-13870 ±20

Table 6.4: Evaluation: 5cm sine wave results

Table 6.5 outlines the performance between the models when compared using the ISO 2631 standard.

Comfort Level	PID-controller	Residual Policy RL
Comfortable	529	533 ±1
Little Uncomfortable	125	129±1
Fairly Uncomfortable	60	50±1
Uncomfortable	47	48±1
Very Uncomfortable	79	77±1
Extremely Uncomfortable	0	0

Table 6.5: Evaluation: 5cm sine wave results as per ISO 2631 ride comfort standard

Based on these results it can be seen that the Residual Policy RL algorithm is capable of reducing the maximum chassis acceleration, velocity and position, and also improving the

reward when compared with the PID-controller. In addition, it can also be seen that the Residual Policy RL algorithm is also effective at reducing the number of occurrences the chassis acceleration spends in the "Very Uncomfortable" band with the maximum number of occurrences being 78 compared with 79 for the PID-controller. There is a slight increase in the number of occurrences the chassis spends in the "Uncomfortable" comfort band, however, this was found to be predominantly caused by the reduction of times the acceleration reached the "Very Uncomfortable" band i.e. the Residual Policy RL algorithm reduced occurrences in the "Very Uncomfortable" band in favour for increasing occurrences in the "Uncomfortable". Similar trends are seen in the "Fairly Uncomfortable" and "Little Uncomfortable" bands with the Residual Policy RL algorithm tending to reduce occurrences in the "Fairly Uncomfortable" band in preference of increasing occurrences in the "Little Uncomfortable" band. All of these metrics indicate the Residual Policy RL algorithm is enhancing the PID-controller's performance and by extension improving the suspension control of the vehicle.

This improvement can also been seen in Figure 6.8 which shows the differences between the magnitudes of the Residual Policy RL algorithm and the PID-controller. Negative numbers indicate the points where the Residual Policy RL algorithm has lower acceleration magnitudes compared to the PID-controller. The orange line here indicates zero. As can be seen the line is predominately lower than zero, indicating that the Residual Policy RL algorithm has a lower acceleration compared to the PID-controller for the majority of the episode. Interestingly, one of the few times the PID-controller does outperform the Residual Policy RL algorithm is directly after the disturbance is encountered. However, whilst it appears the PID-controller does dramatically outperform the Residual Policy RL model here, the PID-controller does also have a higher maximum chassis acceleration as described in Table 6.4. Therefore, whilst this difference is substantial, it can be mostly ignored as it is not representative of the general trend.

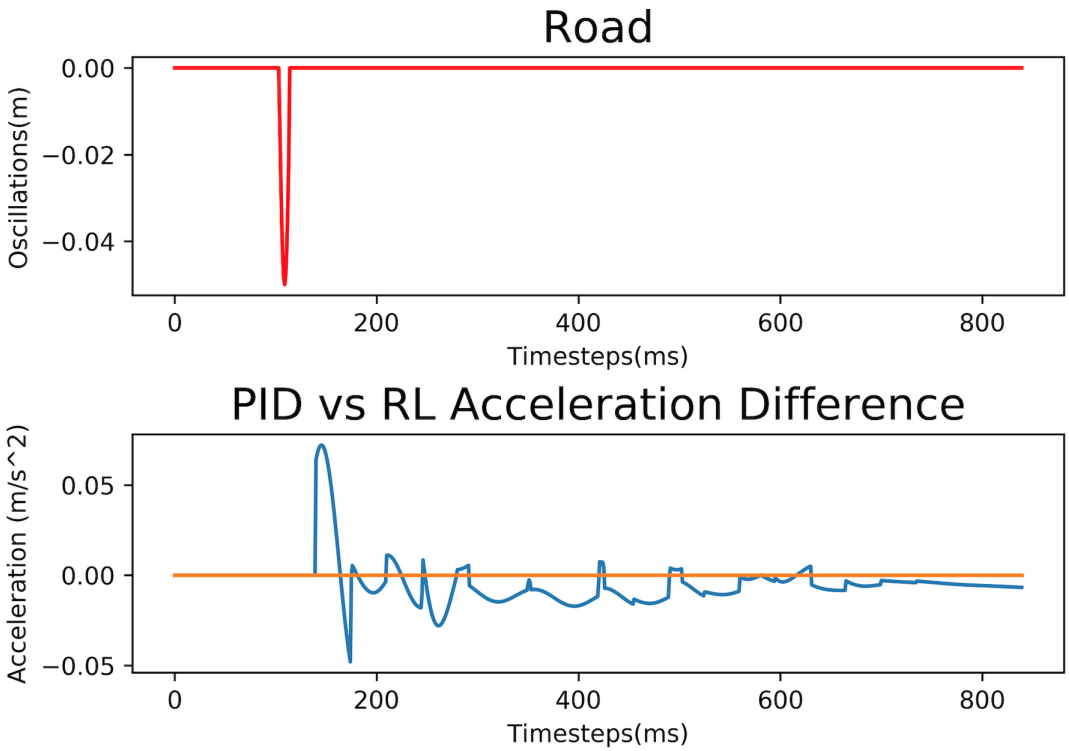

Figure 6.8: Evaluation: 5cm sine wave - acceleration magnitude differences. Negative readings indicate the Residual Policy RL algorithm has lower accelerations than the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

The Residual Policy RL algorithm's ability to consistently reduce the chassis acceleration is also evident in Figure 6.9 where the exponential cumulative magnitudes of acceleration are much lower for the Residual Policy RL algorithm compared to the PID-controller.

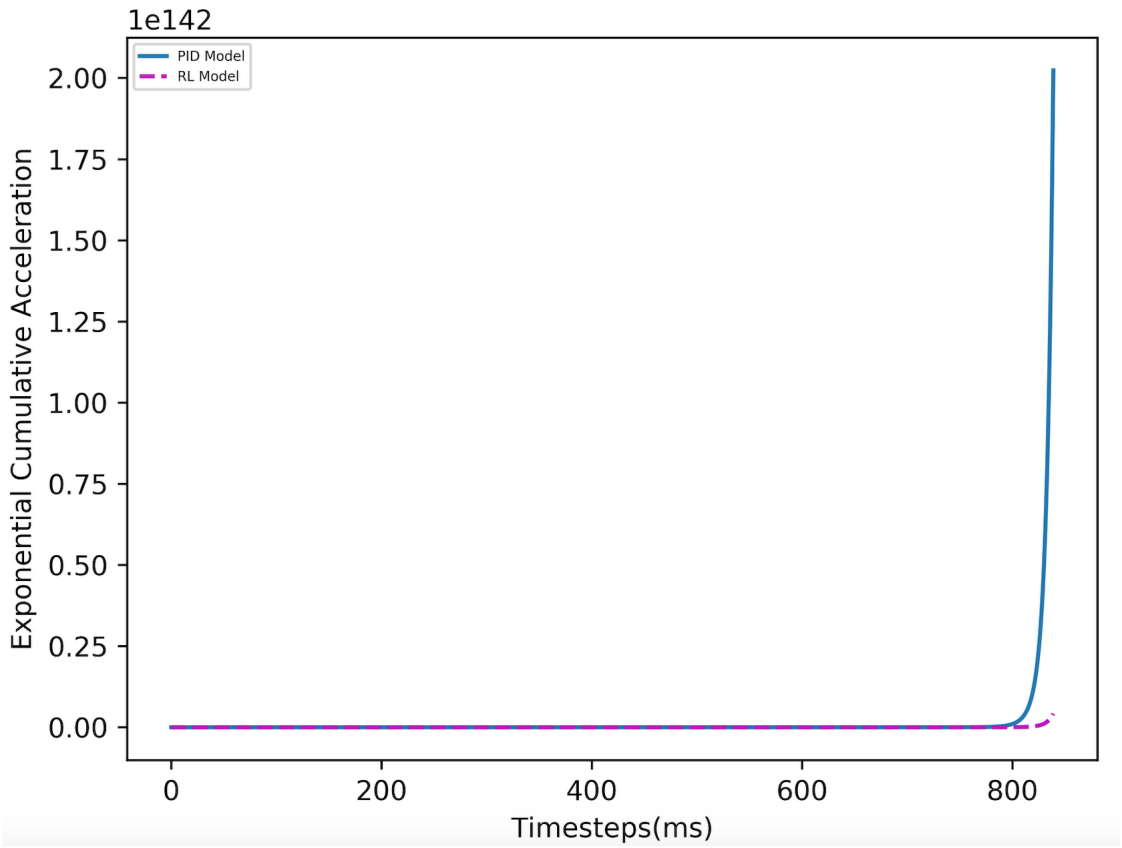

Figure 6.9: Evaluation: 5cm sine wave - Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller. **Note:** over time the PID experiences far more acceleration **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

Due to these results it is clear the Residual Policy RL model does outperform the PID-controller.

6.5.2 Pothole

The second evaluation involved the use of a step function pothole. Again two separate simulations were used, the first simulation showed bump depth of 2.5cm whilst the second simulation showed a bump depth of 5cm. Each experiment was carried out 9 times each to ensure reproducibility. However, during the course of running this experiment it became apparent that 30,000 episodes was not sufficient to train the agent to develop a good policy. As such training was extended and the decay rate was changed. The updated parameters can be seen in Table 6.6. These parameters were only used for the pothole experiments.

Hyper-Parameter	Value
Episodes	300000
Critic LR	0.000000
Actor LR	0.0000001
Exploration Rate	0.3
Decay Rate	0.999999
Target Network τ	0.01
Experience Replay	20000
Mini Batch Size	256
Discount factor γ	0.9

Table 6.6: Hyperparameters for Pothole Experiment

The results will now be discussed.

Pothole - 2.5cm

As stated previously 9 separate models were trained on a 2.5cm deep pothole, which was created using a step function. The results can be seen in Table 6.7.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	0.74	0.65$\pm$0.1
Maximum Chassis Velocity (m/s)	0.014	0.014 $\pm$0.01
Maximum Chassis Position (m)	0.0019	0.0018 $\pm$0.0002
Reward	-2884	-2050 $\pm$300

Table 6.7: Evaluation: 2.5cm pothole results

Table 6.8 outlines the performance between the models when compared using the ISO 2631 standard.

Comfort Level	PID-controller	Residual Policy RL
Comfortable	600	635 $\pm$25
Little Uncomfortable	206	185$\pm$25
Fairly Uncomfortable	34	18$\pm$3
Uncomfortable	0	0
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

Table 6.8: Evaluation: 2.5cm pothole results as per ISO 2631 ride comfort standard

By analysing the results it can be seen that the Residual Policy RL agent is capable of outperforming the PID-controller in terms of maximum chassis acceleration, maximum chassis velocity and maximum chassis position, as well as reward function. However, it is interesting to note the large range of performance differences between models. This results in the Residual Policy Reinforcement Learning algorithm, on some occasions, having higher maximum acceleration, velocity and position than with the PID-controller alone. This occurs even though the reward function is always superior for the Residual Policy RL algorithm. This is most likely due to a local maxima which allow the agent to optimise the reward whilst also showing some undesirable behaviour characteristics. This may be overcome by identifying better reward functions but this can be quite tedious.

Nonetheless, any deterioration in performance on these metrics was very small and had very little impact on the model's performance on the ISO 2631 standard. As can be seen in Table 6.8 the Residual Policy Reinforcement Learning algorithm was consistently better than the PID-controller, with the Residual Policy RL algorithm reducing the number of timesteps spent at both the "Fairly Uncomfortable" and "Little Uncomfortable" comfort levels in favour of increased time spent in the "Comfortable" level. This is a clear indication of an improvement in ride comfort.

Further evidence of this improvement can be seen in Figure 6.11, where the PID-controller

displays higher accelerations more frequently than the Residual Policy Reinforcement Learning algorithm. In fact, the Residual Policy Reinforcement Learning algorithm is **1.49** times more likely to have a smaller acceleration than the PID-controller.

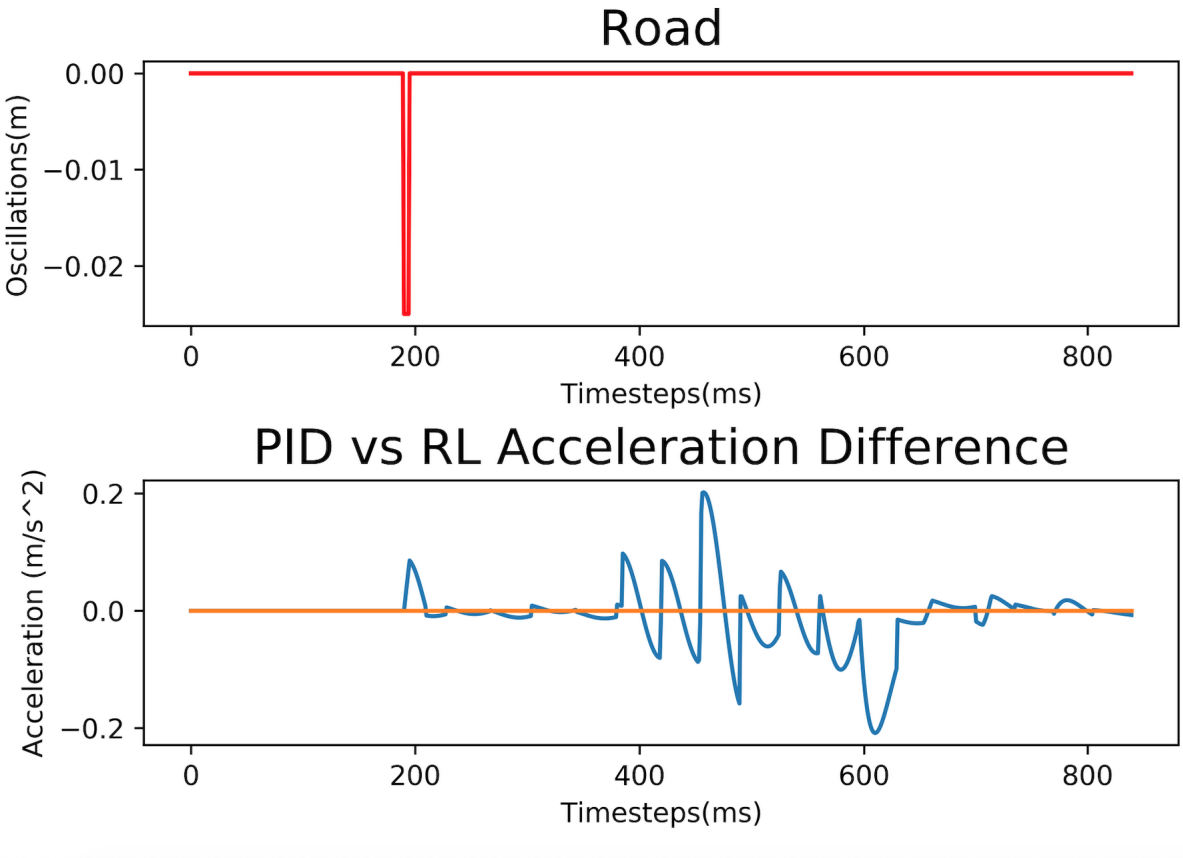

Figure 6.10: Evaluation: 2.5cm pothole - Acceleration Difference between the PID-controller and the Residual Policy RL algorithm. In this case the Residual Policy RL algorithm is **1.49** times more likely to have a reduced acceleration than the PID-controller. **Note:** Negative values indicate the Residual Policy RL algorithm has a lower acceleration and is outperforming the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

This improvement is further seen in Figure 6.11 where the cumulative acceleration is far higher for the PID-controller than the Residual Policy RL agent. As such it can be stated that the Residual Policy RL agent does outperform the PID-controller in this simulation.

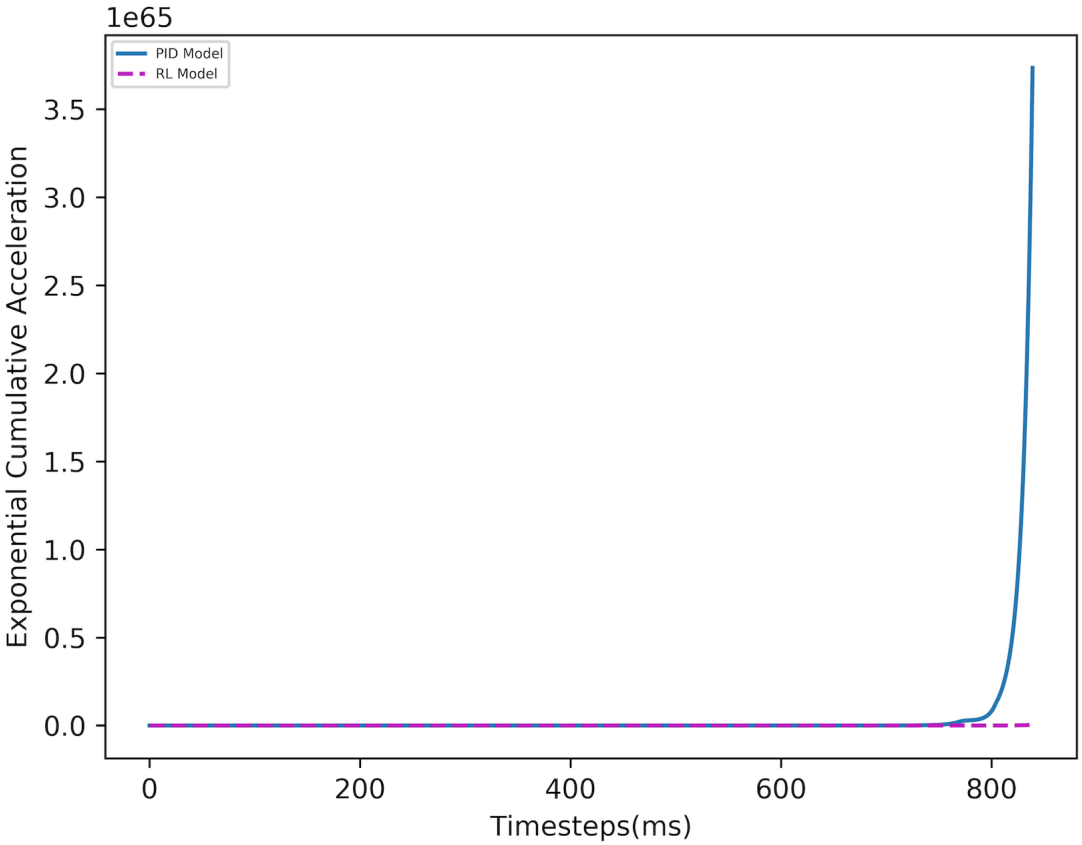

Figure 6.11: Evaluation: 2.5cm pothole - Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller. **Note:** over time the PID experiences far more acceleration. **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

Pothole - 5cm

This experiment was repeated on a pothole of depth 5cm. The results are contained in Table 6.9 and Table 6.10.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	1.49	1.45 $\pm$0.1
Maximum Chassis Velocity (m/s)	0.28	0.032 $\pm$0.001
Maximum Chassis Position (m)	0.0041	0.0032 $\pm$0.0005
Reward	-15152	-11200 $\pm$700

Table 6.9: Evaluation: 5cm pothole results

Table 6.10 outlines the performance between the models when compared using the ISO 2631 standard.

Comfort Level	PID-controller	Residual Policy RL
Comfortable	764	510 $\pm$10
Little Uncomfortable	62	95 $\pm$15
Fairly Uncomfortable	150	140 $\pm$30
Uncomfortable	131	77 $\pm$5
Very Uncomfortable	13	10 $\pm$2
Extremely Uncomfortable	0	0

Table 6.10: Evaluation: 5cm pothole results as per ISO 2631 ride comfort standard

Analysing the results from Table 6.9 it can be seen that the Residual Policy Reinforcement Learning Algorithm is capable of reducing the maximum chassis acceleration and position

whilst also maximising the reward received. These are all indications that the Residual Policy RL algorithm is outperforming the PID-controller. However, the Residual Policy RL algorithm was never able to improve the maximum chassis velocity, which is interesting as this value is directly derived from the acceleration. Nonetheless, whilst the PID-controller does have a lower maximum velocity, which is also a sign of improved comfort, the Residual Policy RL algorithm beats it on all other metrics in this table. As such it can be stated that based on these metrics the Residual Policy RL algorithm outperforms the PID-controller.

This improvement is also seen in Table 6.10 where the Residual Policy RL algorithm repeatedly reduces the number of occurrences the chassis experienced acceleration values in the lower comfort tiers e.g. "Very Uncomfortable" and "Uncomfortable". Instead the algorithm increased experiences in the more comfortable tiers such as "Little Uncomfortable" and "Fairly Uncomfortable".

As well as this, Figure 6.12 highlights the acceleration differences between the Residual Policy RL algorithm and the PID-controller. Again, negative values indicate the Residual Policy RL algorithm has lower accelerations compared to the PID-controller. In this case, the Residual Policy RL algorithm is **2** times more likely to have a lower acceleration than the PID-controller.

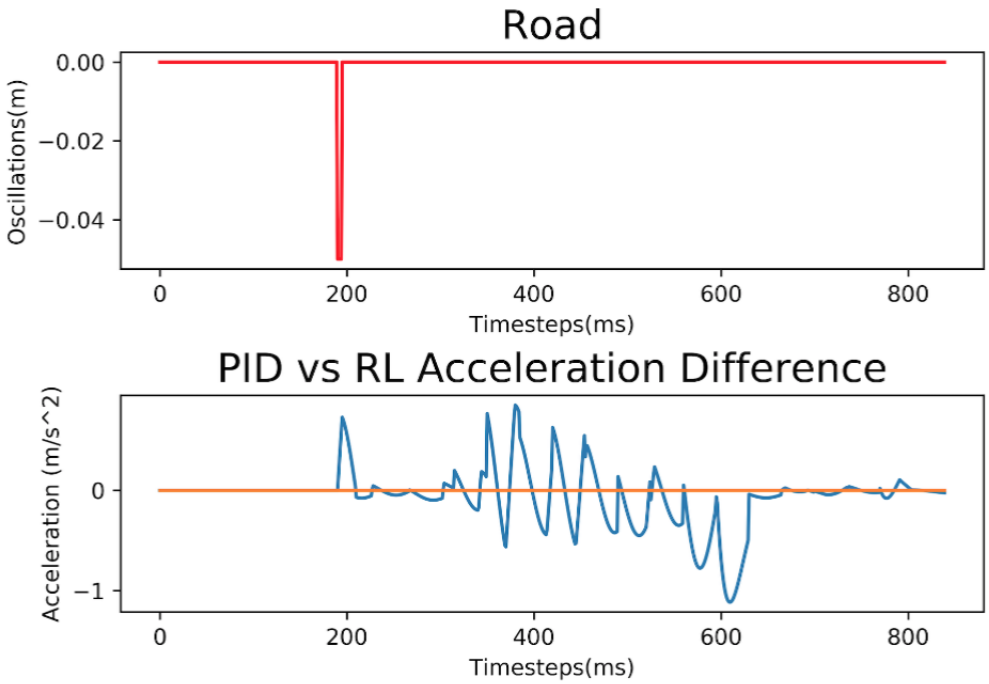

Figure 6.12: Evaluation: 5cm pothole - Acceleration Difference between the PID-controller and the Residual Policy RL algorithm. In this case the Residual Policy RL algorithm is **2** times more likely to have a reduced acceleration than the PID-controller. **Note:** Negative values indicate the Residual Policy RL algorithm has a lower acceleration and is outperforming the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

Again, this result is reflected in the exponential cumulative acceleration of the two, with the PID-controller having a far larger value. This is shown in Figure 6.13

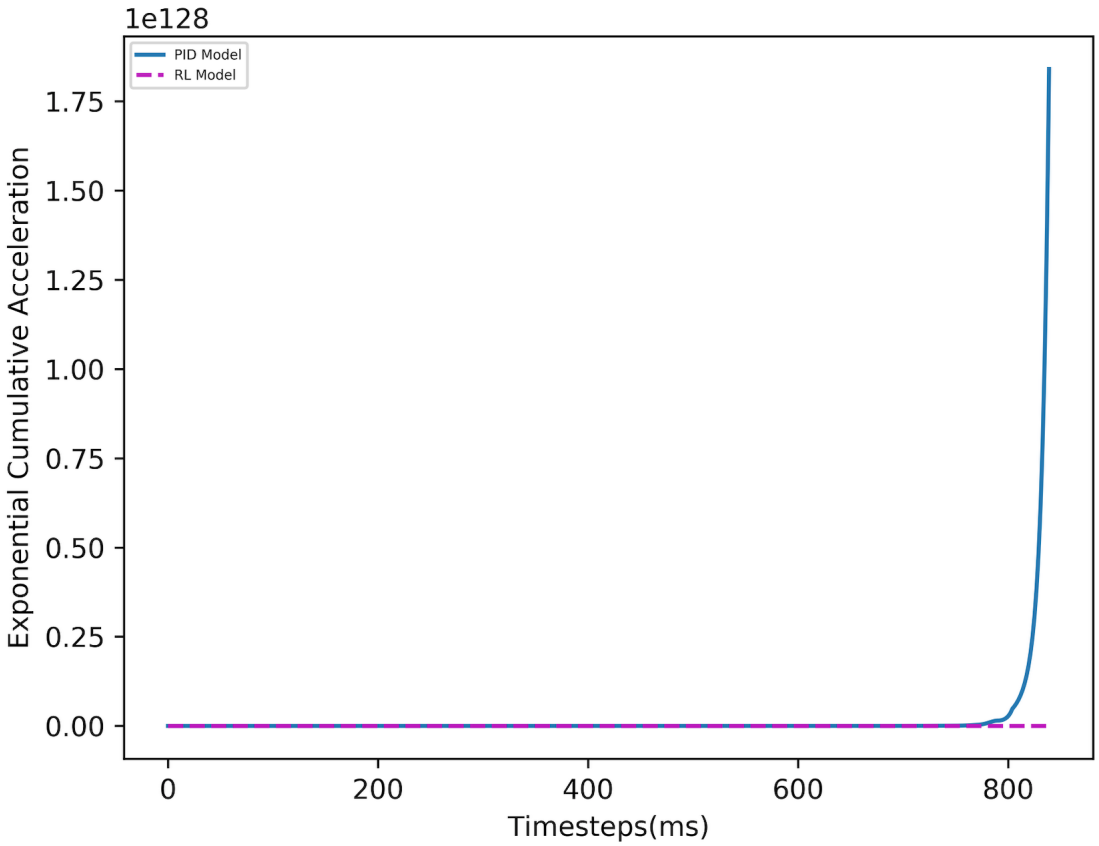

Figure 6.13: Evaluation: 5cm pothole - Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller. **Note:** over time the PID experiences far more acceleration **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

Once again it is clear the Residual Policy RL algorithm outperforms the PID-controller, with better scores on all metrics except for maximum chassis velocity. Hence, it can be stated it once again outperforms the PID-controller in terms of ride comfort and suspension control.

6.5.3 Hybrid

The third evaluation involved using a bump which was a hybrid of both a step function and a sine wave. In this case the front of the bump was a sharp edge like that created by a step function whilst the end of the bump was a sine wave. Again two different bumps heights were used and each experiment was carried out 9 times. The parameters described in Table 5.2 were used. The results are presented below.

Hybrid - 2.5cm

The initial height was set to 2.5cm. The results can be seen in Table 6.11 and 6.12.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	0.52	0.49±0.005
Maximum Chassis Velocity (m/s)	0.01	0.0097±0.0001
Maximum Chassis Movement (m)	0.0014	0.0013
Reward	-335	-280 ±15

Table 6.11: Evaluation: 2.5cm hybrid disturbance results

Comfort Level	PID-controller	Residual Policy RL
Comfortable	796	798
Little Uncomfortable	38	42
Fairly Uncomfortable	6	0
Uncomfortable	0	0
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

Table 6.12: Evaluation: 2.5cm hybrid disturbance results as per ISO 2631 ride comfort standard

Analysing these results shows that in Table 6.11 the Residual Policy RL algorithm consistently outperforms the PID-controller in terms of maximum chassis velocity and position whilst also maximising the reward function. In addition, the algorithm also

frequently outperformed the PID-controller in terms of maximum chassis acceleration, however, some models did fail to do this. Nonetheless, the vast majority of metrics indicate improved performance from the Residual Policy RL algorithm.

This improvement is further evident in Table 6.12 where the Residual Policy RL algorithm consistently remained within the acceleration boundaries of the "Little Uncomfortable" tier, as per the ISO 2631 standard. In contrast the PID-controller routinely exceeded these boundaries and entered into the "Fairly Uncomfortable" tier on 6 occasions.

This enhanced performance can also be seen in Figure 6.14 where the Residual Policy RL algorithm is **2.45** times more likely to have a lower acceleration magnitude compared to the PID-controller.

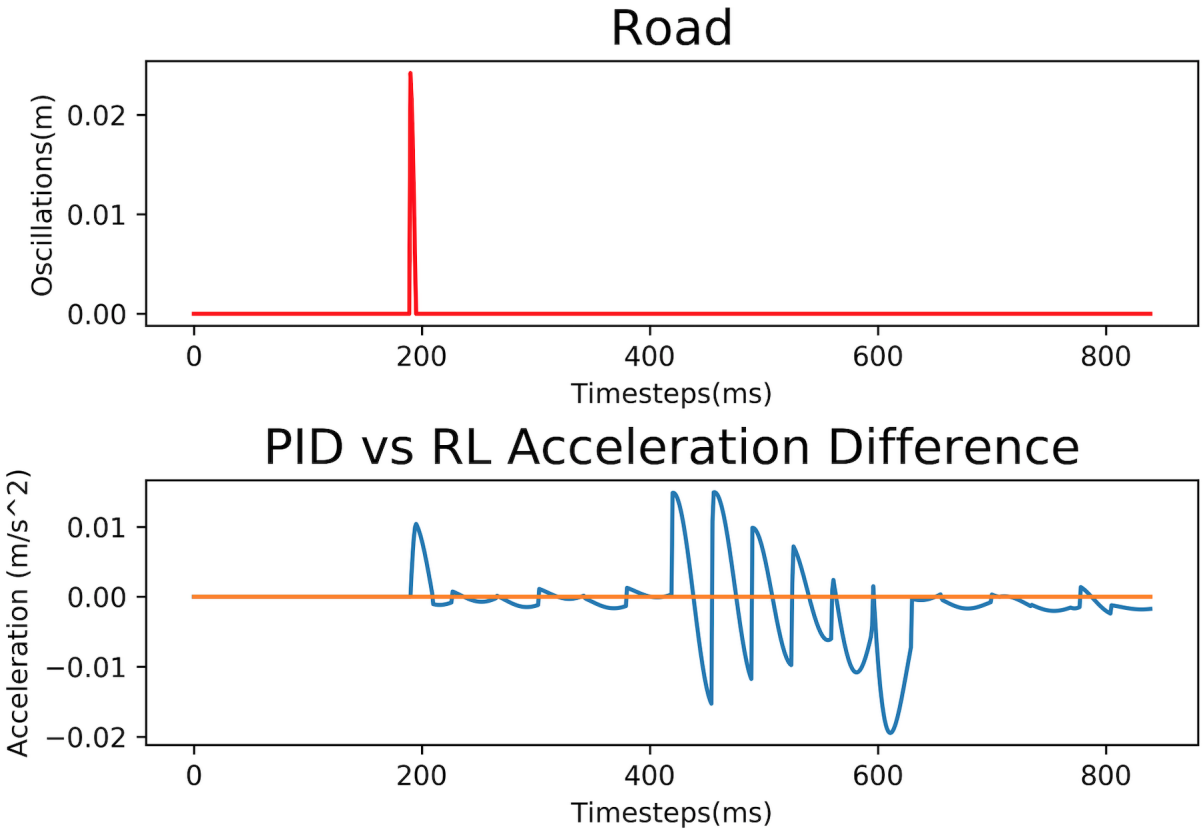

Figure 6.14: Evaluation: 2.5cm hybrid disturbance - Acceleration Difference between the PID-controller and the Residual Policy RL algorithm. In this case the Residual Policy RL algorithm is **2.45** times more likely to have a reduced acceleration than the PID-controller. **Note**: Negative values indicate the Residual Policy RL algorithm has a lower acceleration and is outperforming the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

Again, this results in the exponential cumulative acceleration of the Residual Policy RL algorithm being lower than the PID-controller, as can be seen in Figure 6.15.

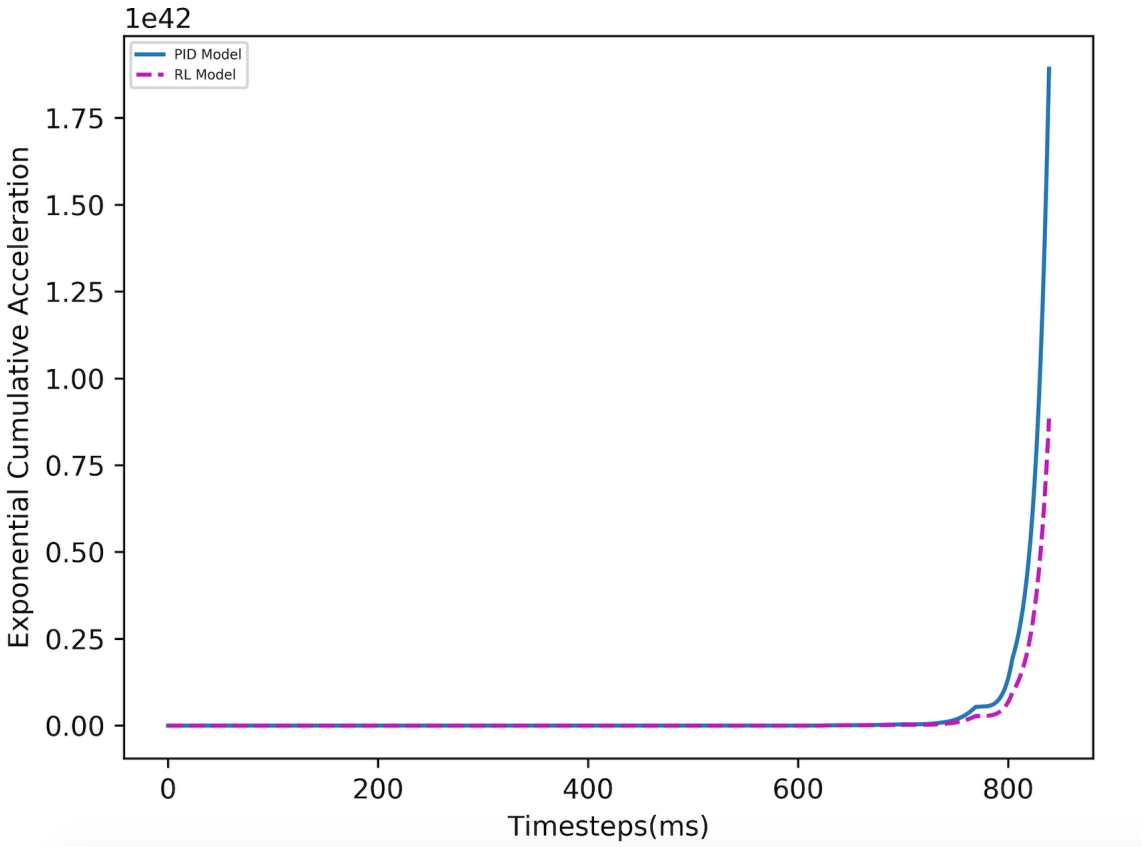

Figure 6.15: Evaluation: 2.5cm hybrid disturbance -Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller **Note:** over time the PID experiences far more acceleration **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

As a result, the Residual Policy RL algorithm can be deemed an improvement on the PID-controller in this case with improved performance seen in terms of the ride comfort.

Hybrid 5cm

Again, this experiment was repeated on the same road disturbance but this time at a height of 5cm. The results can be seen in Table 6.13 and 6.14.

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	1.03	0.983±0.001
Maximum Chassis Velocity (m/s)	0.02	0.0195±0.001
Maximum Chassis Movement (m)	0.0028	0.00268 ±0.00002
Reward	-6214	-6065 ±15

Table 6.13: Evaluation: 5cm hybrid results

Comfort Level	PID-controller	Residual Policy RL
Comfortable	540	544
Little Uncomfortable	142	142 ±1
Fairly Uncomfortable	141	139 ±1
Uncomfortable	17	15
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

Table 6.14: Evaluation: 5cm hybrid results as per ISO 2631 ride comfort standard

As can be seen from Table 6.13 the Residual Policy Reinforcement Learning Algorithm once again outperforms the PID-controller on all available metrics. In addition, the algorithm outperforms the PID-controller on the ISO 2631 standard with it once again showing the ability to reduce the timesteps spent in lower comfort bands such as the "Uncomfortable"

band. All these indicate that the algorithm is improving ride comfort.

This can also be seen in the acceleration differences between the two approaches. This is shown in Figure 6.16. Again it can be seen that the Residual Policy RL algorithm is **2.4** times more likely to have a lower acceleration than the PID-controller.

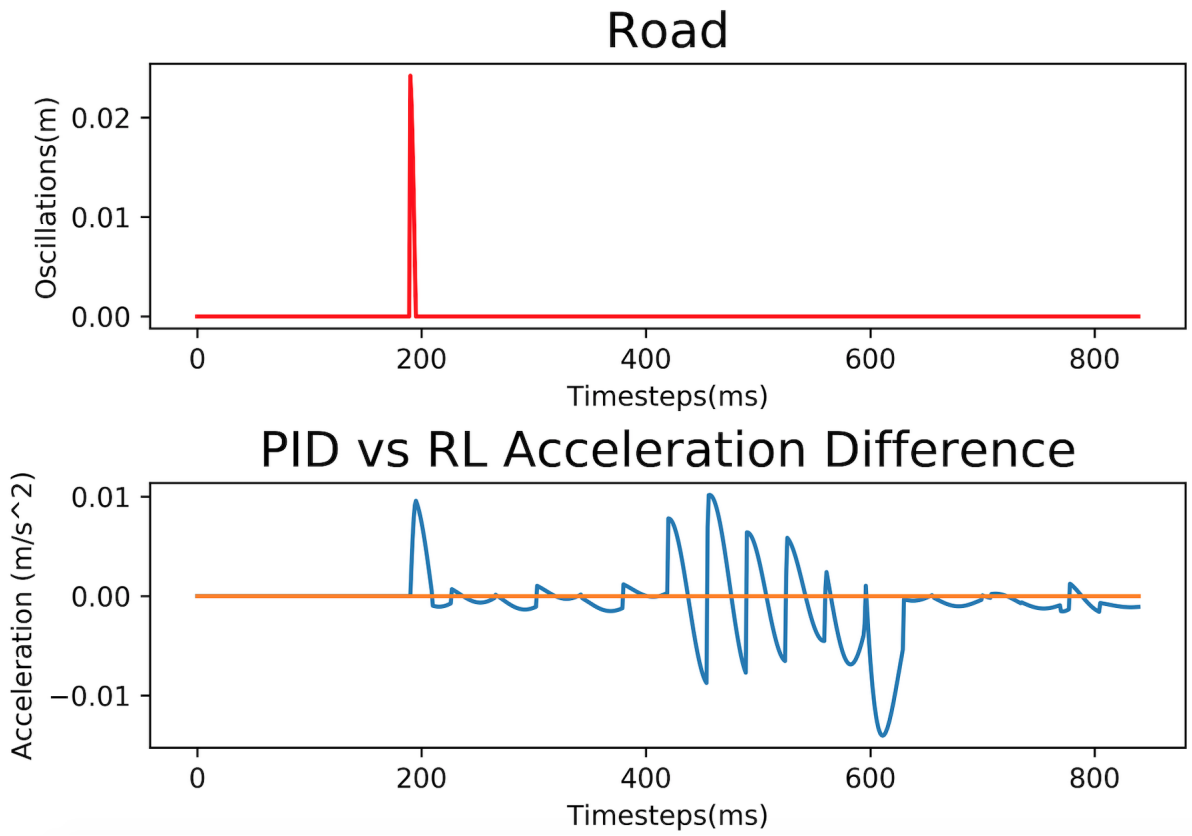

Figure 6.16: Evaluation: 5cm hybrid disturbance -Acceleration Difference between the PID-controller and the Residual Policy RL algorithm. In this case the Residual Policy RL algorithm is **2.4** times more likely to have a reduced acceleration than the PID-controller **Note:** Negative values indicate the Residual Policy RL algorithm has a lower acceleration and is outperforming the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

This is also reflected in the exponential cumulative acceleration which shows the PID-controller experiencing far more acceleration over the course of an episode. This can be seen in Figure 6.17

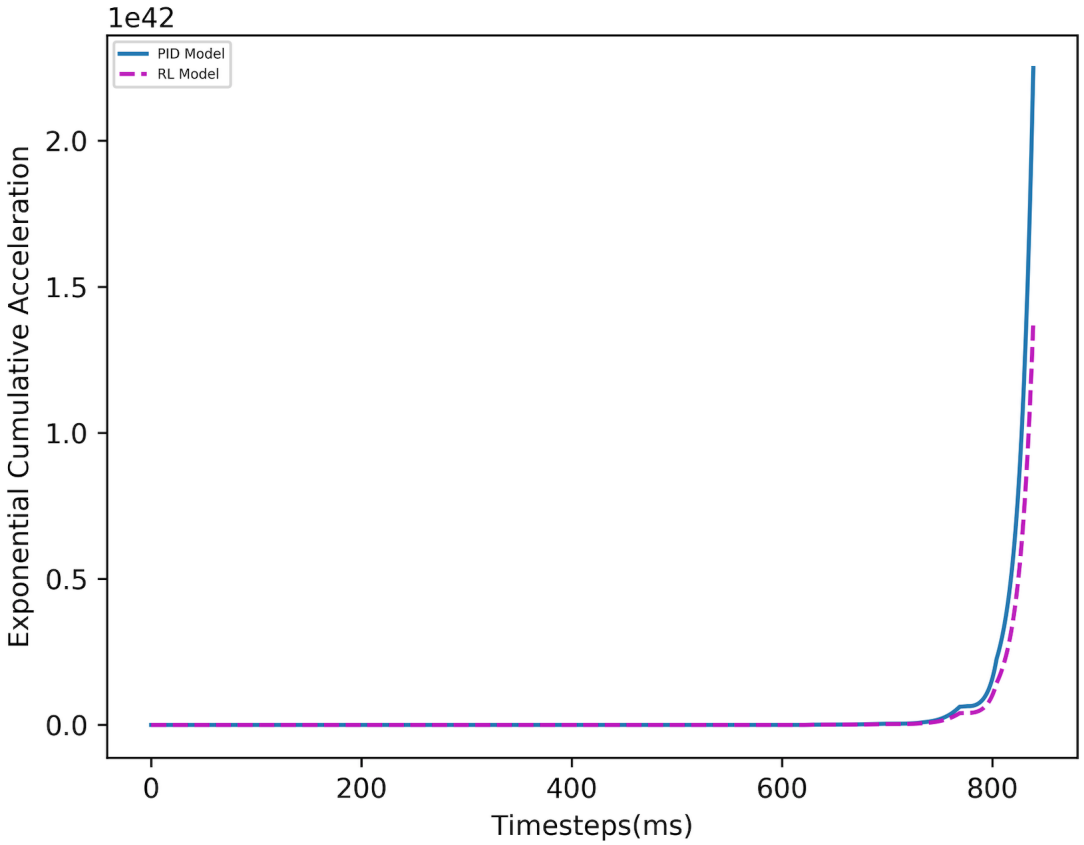

Figure 6.17: Evaluation: 5cm hybrid disturbance -Exponential Cumulative Acceleration for the Residual Policy Reinforcement Learning Algorithm vs the PID-controller **Note:** over time the PID experiences far more acceleration **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

As a result, it is clear that the Residual Policy Reinforcement Learning algorithm improves the performance of suspension control, and enhances the abilities of the PID-controller for suspension control.

6.5.4 Mass Change

The algorithm's ability to re-adjust and relearn following environmental change was also tested by varying the mass of the car from **650kg** to **600kg** during training. Following this change the PID was re-tuned to ensure it was still performing optimally and to ensure the PID-controller was not the cause of any drop in performance. This re-tuning did not result in a change to PID gains i.e. the gains were the same as those seen in Table 5.1. This test was only conducted on a hybrid road disturbance of height 2.5cm. The agent was allowed to re-explore its environment at the point of change. As outlined in section 2.6 one of the PID-controller's biggest weaknesses is its inability to adapt to changes in the environment and chapter 3 outlined a series of approaches put forward to overcome this. Therefore, if the Residual Policy Reinforcement Learning Algorithm can accomplish the readjustment it would

be a major bonus for the method.

Mass Change - 2.5cm

As with all other experiments this test was conducted 9 times with each model showing the ability to re-adapt and re-optimise the PID-controller. This can be seen in Figure 6.18 where the Residual Policy RL algorithm was able to re-optimise the PID-controller following the mass change, which is marked with the dramatic drop in performance of both the PID-controller and Residual Policy RL agent. It is shown that this mass change particularly impacts the Residual Policy RL algorithm which experiences a reward drop from more than -500 to less than -2500. However, following this drop in performance the agent begins to relearn and begins to outperform the PID-controller again, which is seen through the oscillations around the PID line. Whilst these oscillations may seem problematic, due to introduction of the performance threshold and early stopping techniques described in chapter 4 only the best model will be kept i.e. the models at the peaks of the oscillations. As a result, the end product is a re-optimised PID-controller.

The full set of results for this experiment can be found in section A1.2.

Figure 6.18: Evaluation: Rewards During Training **Note:** the agent re-optimises the performance of the PID-controller **Note:** the RL agent here refers to the whole Residual Policy RL algorithm

6.5.5 Residual Policy Reinforcement Learning Algorithm as a PID Tuner

The Residual Policy Reinforcement Learning algorithm was also trained using an untuned PID-controller in order to ascertain the ability of the algorithm to optimise the performance of this PID-controller and to determine the algorithm's ability and applicability to replace traditional tuning methods which are known to be very tedious.

In order to do this, a PID-controller with randomly chosen gains was deployed as the Residual Policy RL algorithm's base policy. Again, a DDPG agent was used for the residual policy. As with previous experiments the model was trained for 30,000 episodes using the hyperparameters outlined in table 5.2. The road disturbance used was the 2.5cm hybrid road disturbance. The performance was compared against a tuned PID-controller's performance using the gains outlined in table 5.1. The untuned PID-controller was randomly assigned the gains seen in table 6.15.

Gain	Value
Proportional	12
Integral	-5
Derivative	7

Table 6.15: Randomly Assigned PID Gain Values

It was found during this experiment that the Residual Policy Reinforcement Learning algorithm using the untuned PID-controller was incapable of achieving any kind of competitive performance when compared with the tuned PID-controller. In fact, it was found that the Residual Policy Reinforcement Learning agent struggled to even enhance the performance of the untuned PID-controller. This is seen in Figure 6.19 where the Residual Policy RL algorithm's performance actually drops when compared to the untuned PID-controller. This is a crucial find as it indicates that the performance of the Residual Policy RL algorithm is heavily dependent on the base policy used.

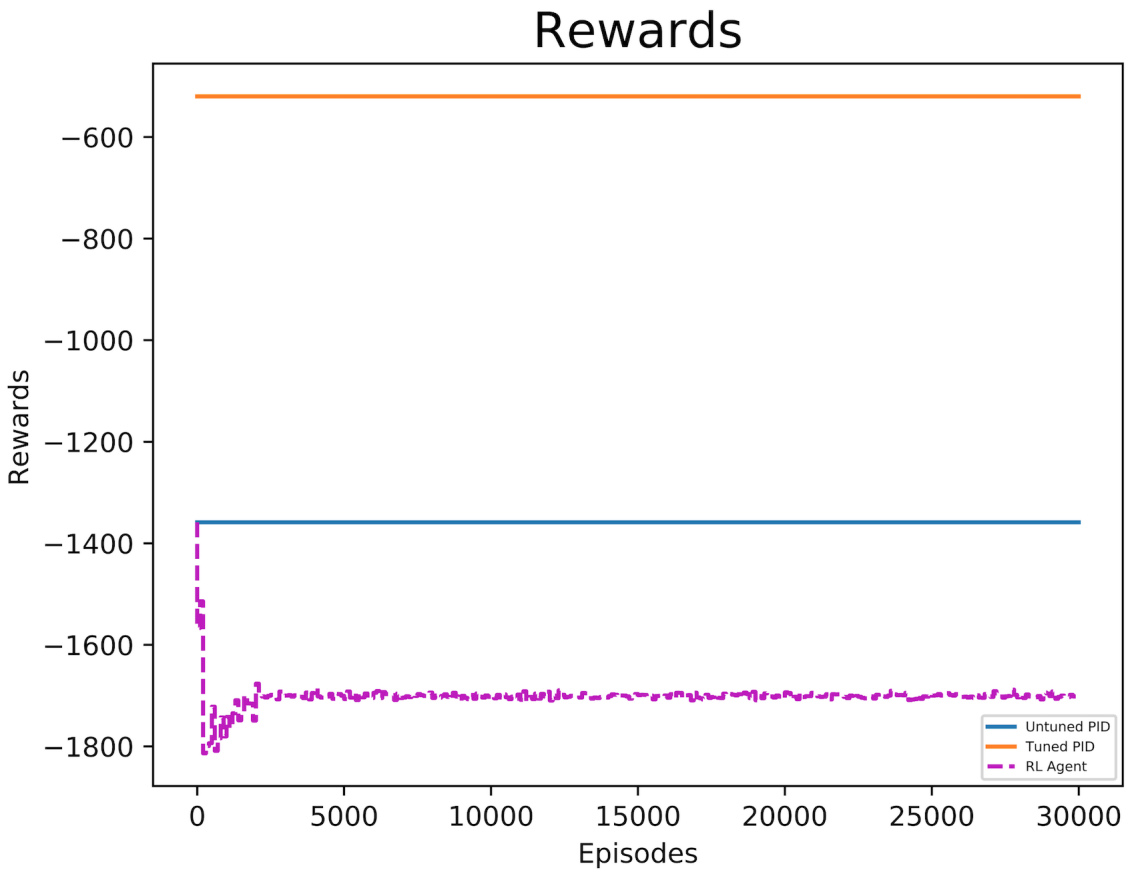

Figure 6.19: Evaluation: Rewards of the Residual Policy RL algorithm using a untuned PID-controller as a base policy **Note:** the RL agent here refers to the whole Residual Policy RL algorithm, whilst the untuned PID-controller refers to the untuned controller alone

This is interesting as throughout the experiments it was seen that the agent was able to vary its action magnitudes in order to take a bigger or smaller role in the final policy. This is seen in Figure 6.20 where actions were in the range [13,18] for the hybrid disturbance, but changed to the bigger range of [300, 800] in Figure 6.21 for the pothole disturbance.

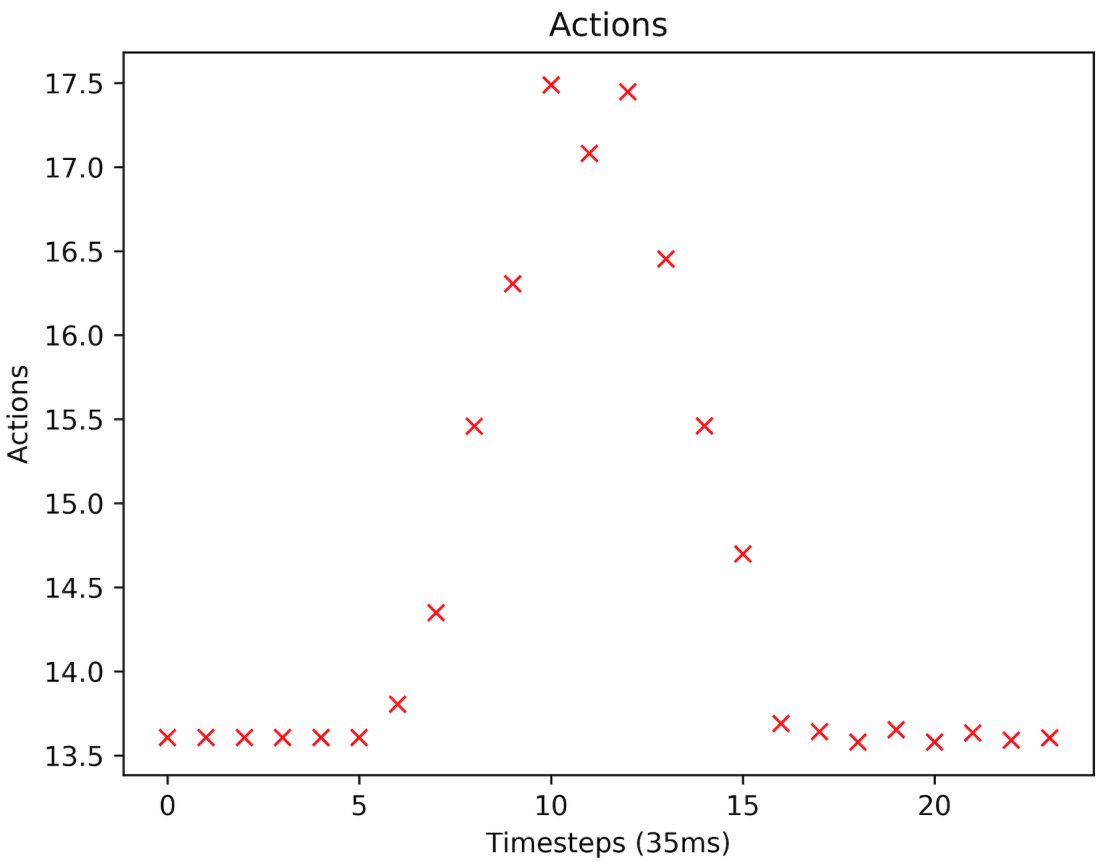

Figure 6.20: Evaluation: Agent's actions on the hybrid road disturbance

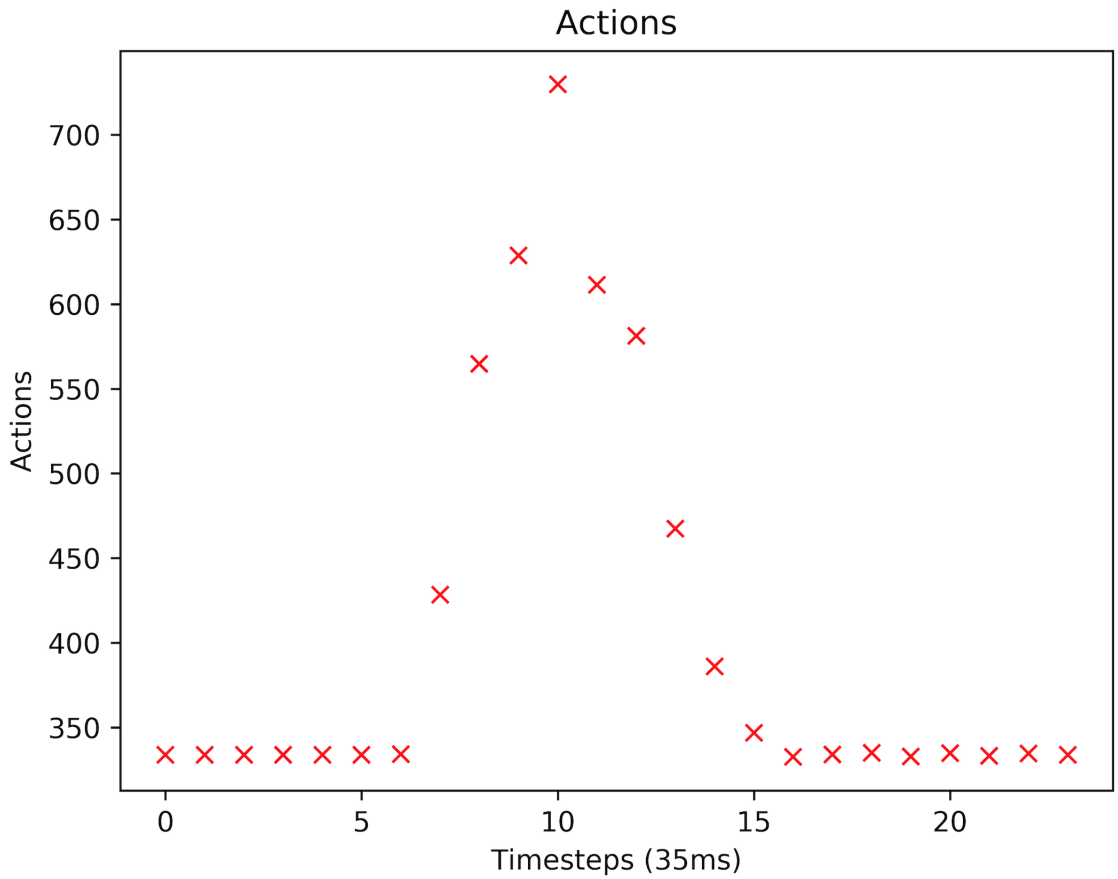

Figure 6.21: Evaluation: Agent's actions on the pothole road disturbance

This indicates that whilst the algorithm can vary its role, as suggested in [9], it is still reliant on a good base policy to ensure good performance. One possible reason for this is due to the exploration experienced by the agent. Specifically, because the base policy is so poor the agent never or very rarely experiences good behaviour and as such struggles to learn any way to improve the final policy. It may be possible to overcome this by introducing a better exploration method which allows the agent to encounter good experiences more frequently.

Nonetheless, this is an important result as it shows that even though Residual Policy RL is capable of dealing with environmental change and re-optimizing PID-controllers, as seen in section 6.5.4, performance is still highly dependent on the performance of the base policy. As such, if the base policy struggles to cope with the environmental change it may result in bad overall performance.

6.5.6 Challenges of Using the Residual Policy Reinforcement Learning

Whilst the Residual Policy RL algorithm has been shown to be very capable at enhancing PID and suspension control, whilst also being capable of adapting to change there were a number of difficulties associated with applying Residual Policy RL to suspension control.

The first problem came in the form of local maxima. The results illustrated above attempt to provide a fair and balanced insight into the general behaviour of models. As such, where applicable, all values provided give an error range. In addition, all the graphs used were chosen based on their representation of the general trends of models trained during the experiments i.e. the graphs used to illustrate differences in acceleration and cumulative differences can be considered as the average performance of models. However, in all experiments there were a number of outlier models which exhibited undesirable behaviour. For example, one pothole model trained the RL agent to output highly negative numbers in the range [-5000,-4500]. This resulted in the total policy output being a consistent damping rate of 100. Interestingly in all cases the outlier models achieved very good scores in terms of the reward. In fact, on several occasions these models achieved the highest rewards out of all models trained in their respective environments. This suggests the models did not fail to learn but instead found different maxima to the majority of models. This is down to the reward function chosen which allows for certain undesirable behaviours to be rewarded.

Whilst all these models were retrained and in all cases managed to achieve similar performance to the general trends of models following this retraining, this indicates one of the biggest difficulties when using an RL algorithm. RL algorithms work by identifying and learning behaviours that maximize reward, as such, if reward functions maximise undesirable behaviour it is possible that the agent can learn this behaviour. The best way to overcome this is by creating a different reward function which does not reward the agent for these bad behaviours, however this is not a trivial task, and usually requires a lot of trial and error.

Another major challenge was hyperparameter selection with models being very sensitive to changes in these values. In particular, agents struggled to learn when the learning rates were too large. As such a lot of time was spent iteratively tuning these variables.

6.6 Summary

In this chapter the experimental results were presented and discussed. As well as this the simulation scenarios used, the metrics used and the objectives for the evaluations were discussed.

The results showed that in nearly all experiments the Residual Policy RL algorithm was capable of improving the performance of the PID-controller and suspension system is terms ride comfort, as described by the ISO 2631 standard, maximum chassis acceleration and position, cumulative acceleration and total rewards. The one exception to this was seen in the sine wave road divot of depth 2.5cm as seen in section 6.5.1. In this case performance was very similar, with the PID-controller arguably performing better on the ISO 2631 standard. However, the Residual Policy RL algorithm did perform better on nearly all other

metrics, therefore it is very hard to determine which approach would be preferable here with the differences being very small.

It is worth noting that the results showed that the Residual Policy RL algorithm seemed to perform best when used on the road disturbances of larger depth or height. It is not entirely clear why this is, however, it appears to be related to the fact that the PID-controller alone is performing quite well on these road disturbances and therefore there may not be as much room to improve. In contrast, the larger road disturbances are more difficult to control and hence the Residual Policy RL algorithm has more space to learn and improve performance.

The results also showed the algorithm's ability to adapt to environmental changes when the vehicle experiences mass change. In this case the algorithm was once again able to outperform the PID-controller alone.

However, the final experiment highlighted the limitations of the algorithm, and the reliance of the algorithm on a good base policy. In this case the algorithm was unable to compensate for the poor performance of a untuned PID-controller, indicating that it is not an applicable option for replacing PID tuning methods.

As a result of these findings both objective 1 and 2, as described in section 6.1, have been achieved, but objective 3 was not.

7 Conclusion

In this chapter the thesis is reviewed and the thesis contributions are presented.

7.1 Thesis Contribution

This thesis proposes and evaluates a method for combining PID-controllers with RL techniques to create a highly effective and complementary system capable of enhancing PID-control and improving the performance of vehicle suspension systems in terms of ride comfort.

Chapter 1 outlined the motivation for the work, highlighting the drawbacks of PID-controllers and illustrating how RL techniques can overcome these. We argued the three main drawbacks of the PID-controller i.e. its linearity, how tedious it is to tune and its inability to adapt to environmental changes, are also the strengths of deep reinforcement learning algorithms. We proposed that by applying Residual Policy Reinforcement Learning these systems could be combined together to create an improved technique for controlling suspension systems.

In chapter 2, the background to the techniques used in this implementation was described. In particular, the DDPG algorithm, which was used to train the RL element of the Residual Policy RL algorithm, was described, as well as the Residual Policy RL algorithm itself. PID-control and suspension control were also discussed.

Chapter 3 outlined other attempts to improve PID performance. This included algorithms aimed at tuning PID-controllers in an online fashion to maintain optimal performance as the system changed. Reinforcement Learning techniques were also discussed, which highlighted previous attempts to both replace and tune PID-controllers using RL techniques. Finally, previous attempts to apply Residual Policy Reinforcement Learning to enhance the performance of a Proportional controller were also described.

Chapter 4 presented the exact design of the algorithm used in this thesis. In particular, the exact design of the Residual Policy RL algorithm was described, as well as the deep reinforcement learning algorithm used to train the agent and the architecture of the deep

neural networks. The additional features, namely: early stopping and the performance threshold, were also presented.

Chapter 5 showed the complete implementation of the algorithm and the simulation used for testing. The Tensorflow implementation was shown using Tensorboard graphs and the functions used to create the environment and agent were outlined. The PID-controller's implementation and gain values were also presented as were the hyperparameters used.

Finally, Chapter 6 presented the evaluation of the model including the scenarios and techniques used for this evaluation. These results showed the Residual Policy Reinforcement Learning algorithm was capable of enhancing the performance of both the PID-controller and the suspension system. The algorithm displayed the ability to repeatedly and consistently outperform the PID-controller on a variety of different metrics including the ISO 2631 Ride Comfort Standard. As well as this, the results showed the algorithm was very capable of relearning and adapting to changes in the environment which a PID-controller alone would struggle with. However, the results all highlighted the limitations of the algorithm and its reliance on good and effective base policies. Without this the algorithm is unable to achieve good performance and as such it does not offer a practical alternative to good tuning methods.

As such the main contribution of this thesis is the proposal for the design and implementation of an algorithm which can enhance the performance of suspension control systems when compared to traditional PID-controllers. It also provides a platform for future work in relation to not only suspension systems but PID-control in general. PID-controllers themselves are used in numerous applications and as such this thesis presents and outlines an algorithm which may be capable of being applied to other applications that use PID-controllers and could represent a huge industrial use for RL techniques. However, this work also highlights the limitations of this algorithm and illustrates the fact that whilst the Residual Policy RL algorithm is capable of enhancing PID-control and dealing with environmental change, it is not capable or ensuring good performance when used alongside poorly tuned or untuned PID-controllers or base policies.

7.2 Future Work

This section outlines areas where future work should be focused. This is based on areas identified during the evaluation of the models, other experiments that would be interesting to run and other applications where the model could be applied.

- **Real Road Data** - future work should focus on applying this algorithm to suspension systems using real road data. Whilst the experiments conducted here used road disturbances in line with the literature, real road data provides an additional challenge

and represents a far more realistic evaluation setting for the algorithm.

- **Noise** - Future work should also analyse the algorithms ability to deal with noise. In this experiment road surfaces were assumed to be smooth, however, this is never the case and as such noisy road surfaces should be used to help mimic the real world. This is similar to applying real road data.
- **Generalization** - In this thesis models were trained independently on a single road disturbance and only evaluated on this road disturbance. As such, future work should test the algorithm's ability to generalize to other road disturbances as in the real world it is very rare for road disturbances to be identical.
- **Improved Simulation** - As mentioned previously, the simulation model used here was very simple and neglected a number of aspects related to real suspension systems. For example, vehicle speed was ignored. As such, whilst the results presented here are promising, they do not directly show the algorithm is capable of controlling real world suspension systems. Therefore, future work should attempt to re-run these experiments on more complex environments.
- **New Applications** - Finally, as touched on previously, PID-controllers are used in many different systems, not just suspension control. As such, future work should also consider the performance of the Residual Policy RL algorithm on other PID-controlled systems. If the algorithm can be shown to work well on other applications it could represent a large step forward in industrial uses for Reinforcement Learning.

Bibliography

[1] Andrew Hynes. The applications of reinforcement learning techniques to car control. Report Outlining Work carried out during time in ZF, June 2019.

[2] Andrew Thomas. Reinforcement learning tutorial with tensorflow. URL `https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/`.

[3] P Sibi, S Allwyn Jones, and P Siddarth. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, 47(3):1264–1268, 2013.

[4] Kjell Magne Fauske. Example: Neural network. URL `http://www.texample.net/tikz/examples/neural-network/`.

[5] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[6] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[8] Tobias Ehlgen Andrew Hynes. Zf kolloqium presentation: An introduction to reinforcement learning. Presentation Introducing Reinforcement Learning, June 2019.

[9] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[10] TutorialPoint. Control systems. URL `https://www.tutorialspoint.com/control_systems/control_systems_tutorial.pdf`.

[11] Muir Alex. How car springs and dampers work. URL https://www.howacarworks.com/basics/how-car-springs-and-dampers-work.

[12] Ajith Abraham, Arijit Biswas, Swagatam Das, and Sambarta Dasgupta. Design of fractional order piλdμ controllers with an improved differential evolution. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1445–1452, 2008.

[13] Laiq Khan, Shahid Qamar, and Umair Khan. Adaptive pid control scheme for full car suspension control. *Journal of the Chinese Institute of Engineers*, 39(2):169–185, 2016.

[14] Hamed Khodadadi and Hamid Ghadiri. Self-tuning pid controller design using fuzzy logic for half car active suspension system. *International Journal of Dynamics and Control*, 6(1):224–232, 2018.

[15] Rodrigo Hernández-Alvarado, Luis Govinda García-Valdovinos, Tomás Salgado-Jiménez, Alfonso Gómez-Espinosa, and Fernando Fonseca-Navarro. Neural network-based self-tuning pid control for underwater vehicles. *Sensors*, 16(9):1429, 2016.

[16] Ignacio Carlucho, Mariano De Paula, Sebastian A Villar, and Gerardo G Acosta. Incremental q-learning strategy for adaptive pid control of mobile robots. *Expert Systems with Applications*, 80:183–199, 2017.

[17] Xue-Song Wang, Yu-Hu Cheng, and Sun Wei. A proposal of adaptive pid controller based on reinforcement learning. *Journal of China University of Mining and Technology*, 17(1):40–44, 2007.

[18] Bo Hu, Jie Yang, Jiaxi Li, Shuang Li, and Haitao Bai. Intelligent control strategy for transient response of a variable geometry turbocharger system based on deep reinforcement learning. *Processes*, 7(9):601, 2019.

[19] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. *arXiv preprint arXiv:1906.05841*, 2019.

[20] Y Susatio, L Oktaviana, NK Rizki, E Listijorini, and TR Biyanto. Design of half-car active suspension system for passenger riding comfort. In *Journal of Physics: Conference Series*, volume 1075, page 012030. IOP Publishing, 2018.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Ervin Alvarez-Sánchez. A quarter-car suspension system: car body mass estimator and sliding mode control. *Procedia Technology*, 7(0):208–214, 2013.

[23] John G Ziegler, Nathaniel B Nichols, et al. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[24] K. L. Chien, J. A. Hrones, and J. B. Reswick. On the automatic control of generalized passive systems. *trans. ASME*, 74, 1952.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[26] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2): 1–21, 2019.

[27] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[30] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[31] Richard S Sutton and Andrew G Barto. A temporal-difference model of classical conditioning. In *Proceedings of the ninth annual conference of the cognitive science society*, pages 355–378. Seattle, WA, 1987.

[32] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[33] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press, 2017.

[34] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[35] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[36] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul): 2121–2159, 2011.

[38] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[39] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*, 2012.

[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[42] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[44] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[45] A Samuel. Some studies in machine learning using the game of checkers (pp. 71-105). *Computers and Thought. New York: McGraw-Hill*, 1963.

[46] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[47] Lucas Lehnert, Romain Laroche, and Harm van Seijen. On value function representation of long horizon problems. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[48] MJ Crosby and Dean C Karnopp. The active damper: a new concept for shock and vibration control. *Shock and Vibration Bulletin*, 43(4):119–133, 1973.

[49] Jing Lei, Zuo Jiang, Ya-Li Li, and Wu-Xin Li. Active vibration control for nonlinear vehicle suspension with actuator delay via i/o feedback linearization. *International Journal of Control*, 87(10):2081–2096, 2014.

[50] Ales˘ Kruczek, Antoni´ n Str˘i´ brsky´, Kater˘ina Hyniova´, and Martin Hlinovsky´. H-infinity controlled actuators in automotive active suspension system. In *Engineering Systems Design and Analysis*, volume 48388, pages 243–247, 2008.

[51] M Senthil Kumar and S Vijayarangan. Design of lqr controller for active suspension system. 2006.

[52] Mark N Howell and Matt C Best. On-line pid tuning for engine idle-speed control using continuous action reinforcement learning automata. *Control Engineering Practice*, 8(2): 147–154, 2000.

[53] Jeffrey A Cook and Barry K Powell. Modeling of an internal combustion engine for control analysis. *IEEE Control Systems Magazine*, 8(4):20–26, 1988.

[54] Weinan Deng, Hao Li, and YuanQiao Wen. Data-driven unmanned surface vessel path following control method based on reinforcement learning. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 3035–3040. IEEE, 2019.

[55] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[56] Le Jiang, Yafei Wang, Lin Wang, and Jingkai Wu. Path tracking control based on deep reinforcement learning in autonomous driving. In *2019 3rd Conference on Vehicle Control and Intelligence (CVCI)*, pages 1–6. IEEE, 2019.

[57] Mirosław Tomera. Nonlinear controller design of a ship autopilot. *International Journal of Applied Mathematics and Computer Science*, 20(2):271–280, 2010.

[58] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[59] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[60] MMM Salem, Ayman A Aly, et al. Fuzzy control of a quarter-car suspension system. *World Academy of Science, Engineering and Technology*, 53(5):258–263, 2009.

[61] Abdolvahab Agharkakli, Ghob ad Shafiei Sabet, and Armin Barouz. Simulation and analysis of passive and active suspension system using quarter car model for different road profile. *International Journal of Engineering Trends and Technology*, 3(5): 636–644, 2012.

[62] Nemat Changizi and Modjtaba Rouhani. Comparing pid and fuzzy logic control a quarter car suspension system. *The journal of mathematics and computer science*, 2 (3):559–564, 2011.

[63] ISO ISO. 2631-1: Mechanical vibration and shock-evaluation of human exposure to whole-body vibration-part 1: General requirements. *Geneva, Switzerland: ISO*, 1997.

[64] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[65] Massimo Canale, Mario Milanese, and Carlo Novara. Semi-active suspension control using "fast" model-predictive techniques. *IEEE Transactions on control systems technology*, 14(6):1034–1046, 2006.

[66] Gregory D Buckner, Karl T Schuetze, and Joe H Beno. Active vehicle suspension control using intelligent feedback linearization. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*, volume 6, pages 4014–4018. IEEE, 2000.

[67] Wattanawong Romsai and Auttarat Nawikavatan. Optimal pida controller design for quarter car suspension system by intensified current search. In *2019 17th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pages 1–5. IEEE, 2019.

[68] Zhaojian Li, Ilya Kolmanovsky, Ella Atkins, Jianbo Lu, Dimitar Filev, and John Michelini. Cloud aided semi-active suspension control. In *2014 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)*, pages 76–83. IEEE, 2014.

[69] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[70] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

A1 Appendix

A1.1 Deep Learning Early History

The earliest work on neural networks was in 1943, when [69] proposed a non-linear computing unit known as a neuron, which was designed to model the behaviour of a neuron in the brain. This was then expanded by [70], which introduced the idea of weighted inputs for these neurons. This model used the weighted sum of the inputs to determine a binary output, in this case 1 if the weighted sum was positive or 0 if the sum was negative.

However, due to difficulties training perceptrons with multiple layers, progress on neural networks haulted until 1986 when [35] proposed the Backpropagation algorithm which was capable of training neural networks with many layers and thousands of neurons.

Since then deep learning has been progressing quickly and has been applied to a wide range of applications, from image classification, to sentiment analysis to text translations.

A1.2 Mass Change Results

Before Mass Change

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	0.52	0.495±0.005
Maximum Chassis Velocity (m/s)	0.01	0.00980±0.00005
Maximum Chassis Movement (m)	0.0014	0.00134 ±0.00001
Reward	-335	-290 ±5

Table A1.1: Evaluation: 2.5cm hybrid results before mass change

Comfort Level	PID-controller	Residual Policy RL
Comfortable	796	798
Little Uncomfortable	38	42 ±1
Fairly Uncomfortable	6	0
Uncomfortable	0	0
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

Table A1.2: 2.5cm hybrid results prior to mass change as per ISO 2631 ride comfort standard

After Mass Change

Metric	PID-controller	Residual Policy RL
Maximum Chassis Acceleration (m/s^2)	0.77	0.710±0.005
Maximum Chassis Velocity (m/s)	0.016	0.0150±0.0002
Maximum Chassis Movement (m)	0.0019	0.00182 ±0.00002
Reward	-2238	-2045 ±20

Table A1.3: Evaluation: 2.5cm hybrid results after mass change

Comfort Level	PID-controller	Residual Policy RL
Comfortable	627	628
Little Uncomfortable	191	194 ±1
Fairly Uncomfortable	22	18±1
Uncomfortable	0	0
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

Table A1.4: Evaluation: 2.5cm hybrid results after mass change as per ISO 2631 ride comfort standard