CHALMERS
UNIVERSITY OF TECHNOLOGY


Images by Volvo Car Corporation – ID: 151946 & 149306

Method To Improve a Wheel Suspension Design Using VI-CarRealTime and Reinforcement Learning

Final term report for the TME180 – Automotive Engineering Project

Manuel Denneler
Christoph Heilig
Vinayanand Bangalore Venkatesh Prasad
Vivekanandan Madhuravasal Narasimhan
Abhishek Amit Kolekar

In Collaboration with Volvo Car Corporation

Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2024

Final report for the TME180 course -
Automotive Engineering Project

Method To Improve a Wheel Suspension Design Using VI-CarRealTime and Reinforcement Learning

Manuel Denneler
Christoph Heilig
Vinayanand Bangalore Venkatesh Prasad
Vivekanandan Madhuravasal Narasimhan
Abhishek Amit Kolekar

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Vehicle Dynamics group
Chalmers University of Technology
Göteborg, Sweden 2024

Method To Improve a Wheel Suspension Design Using VI-CarRealTime and Reinforcement Learning

Final Report

Division of Vehicle Engineering and Autonomous Systems
Department of Mechanics and Maritime Sciences
Vehicle Dynamics group
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Examiner:
David Sedarsky, sedarsky@chalmers.se
Volvo Car Corporation Stakeholders:
Max Boerboom, max.boerboom@volvocars.com
Kenneth Ekström, kenneth.ekstrom@volvocars.com
Supervisors at Chalmers:
Yansong Huang, yansong.huang@volvocars.com
Bengt Jacobson, bengt.jacobson@chalmers.se

Contents

Preface

In this study, simulations for generating wheel suspension designs have been done with Matlab, VICarRealTime and a physical vehicle dynamics simulator. The tests have been carried out from September 2023 to January 2024. The work is a part of a research project concerning the development of a method making the design process of wheel suspension faster. The project is carried out at the Division of Vehicle Engineering and Autonomous Systems in the Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Sweden. The project is financed through the company Volvo Car Corporation as the main stakeholder.

This part of the project has been carried out with the examiner David Sedarsky, Max Boerboom as Volvo Car Corporation supervisor, and Yansong Huang and Bengt Jacobsen as supervisors at Chalmers University of Technology. All tests have been carried out in the test simulation center at Volvo Car location in Gothenburg, Sweden. We are grateful to those involved in the for their help during the project. We would also like to thank Volvo Car Corporation for their co-operation, involvement, and the provision of the simulation opportunities. We extend our sincere gratitude to the VEAP department at Chalmers for accommodating the team with meeting rooms and fueling our brainstorming sessions with delightful hot chocolate.

Finally, it should be noted that the tests could never have been conducted without the sense of high quality and professionalism of the laboratory staff.

There are two different versions of this final report. One as the main version presented here, the other as a shortened version focused on the results provided for Volvo Car Corporation and their supervisors.

Göteborg, Sweden 2024-01-07

Manuel Denneler, Christoph Heilig, Vinayanand Bangalore Venkatesh Prasad, Vivekanandan Madhuravasal Narasimhan, Abhishek Amit Kolekar

Abbreviations

Abbreviation	Description
AI	Artificial Intelligence
OEM	Original Equipment Manufacturer
RL	Reinforcement Learning
KP	Kinematic Parameters (chosen) – camber [deg], SVA [deg], toe [deg], X-displacement [mm], Y-displacement [mm]
SVA	Side View Angle [deg]
K_u	Understeer gradient [deg/g]
RG	Roll gradient [deg/g]
YGM	Yaw-rate Gain Margin
PGM	Yaw-rate Phase Margin
DDPG	Deep Deterministic Policy Gradient

List of figures

List of tables

Abstract

The project focuses on the enhancement of wheel suspension design through the utilization of VI-CarRealTime and Reinforcement Learning techniques. The primary objective of the study is to improve vehicle dynamics and autonomous systems, thereby contributing to the advancement of automotive engineering. The development of vehicle suspension systems is a complex and iterative process, involving the adjustment of various parameters to meet quantitative and qualitative metrics. The report emphasizes the significance of simulating different suspension setups to achieve optimal design solutions. It highlights the essential collaboration between simulation engineers and design engineers to ensure the successful development of suspension systems.

The project group aimed to use optimisation techniques and artificial intelligence to streamline the process of developing an optimal suspension in a time-saving manner. The use of the VI-CarRealTime simulation tool facilitated the analysis and synthesis loops in the suspension design development process and enabled the evaluation of kinematic properties and system requirements. Furthermore, this report deals with the application of machine learning theory, in particular with concepts of reinforcement learning. A comprehensive overview of reinforcement learning, its elements, workflows and classification is provided, highlighting its potential for suspension design optimisation. A detailed comparison of reinforcement learning with other optimisation methods is also presented, highlighting its benefits in the context of suspension development.

The development and description of a MATLAB script for the project is presented, highlighting the technical aspects of implementing reinforcement learning techniques in the context of suspension design. This report concludes with a discussion of the potential impact of the research on the automotive industry, emphasising the importance of the results for the advancement of vehicle dynamics and automotive engineering as a whole. To summarise, the project represents a contribution to improving suspension design through the integration of VI-CarRealTime and reinforcement learning techniques. The findings and insights presented in this report have the potential to significantly impact the automotive industry by contributing to the development of more efficient and optimised vehicle suspension systems.

1. Introduction

A vehicle has to fulfil a variety of system requirements to be considered fit for production and release. To fulfil all requirements equally and in full, each expectation must be quantifiably abstracted so that both the objective and subjectively perceptible characteristics can be evaluated using objective measurement parameters. Secondly, each requirement must be assigned to the individual subsystems so that each can be developed parallelly and by the experts for each corresponding subsystem.

The subsystem of a vehicle in this project corresponds to the wheel axle and the related suspension design. The development process for the suspension subsystem traditionally is an iterative process running in loops at various levels. At the beginning of each iteration, the task focuses on finding suspension design solutions which fulfils system requirements ("synthesis"). At the subsystem level of the wheel suspension, this design consists of so-called "hard points", the locations of the suspension mounting points on the vehicle chassis. Although these can often simulate the required kinematic behaviour of the wheel due to their free placement, they must also be in areas specified by the design in order to prevent an arbitrary, inefficient design of the vehicle ("packaging constraints"). If this cannot be guaranteed during the final check, the design must be adjusted iteratively in new loops.

This development process is time consuming, and one cannot always be sure that the best design is found. Finding out how closely a given suspension design with its hard points approximates the desired kinematic characteristics in order to fulfil the system requirements describes the development process in reverse. This frontloading of the suspension design is called "analysis". The following picture shows the described process.

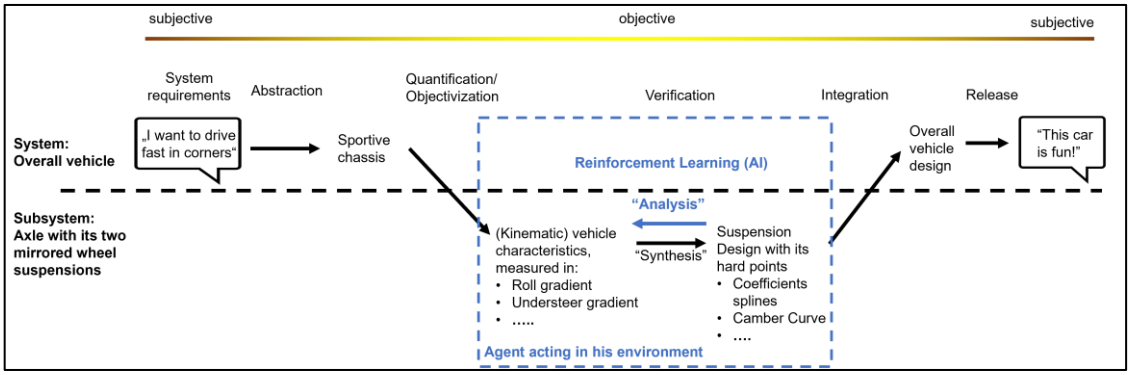

Figure 1: "Analysis" and "Synthesis" loops in the development process of a suspension design

1.1. Background

Suspension design for vehicles is a highly iterative process with quick loops that involves changing different parameters to reach a set of quantitative and qualitative metrics. It is also always a part of the overall vehicle design process, which means that the surroundings of the suspension, and the requirement on the suspension, often can change between the iterations. To improve the suspension development, it is necessary to simulate different suspension setups. Therefore, the simulation department has a design matrix which shows the complete vehicle behavior for the different target areas based on the quantitative and qualitative metrics. To get a good suspension design, the simulation engineers and the design engineers must work hand in hand.

1.2. Aim

This project, in association with Volvo Cars, aims to "front load" the suspension design process, by moving to a top-down suspension design approach which allows for faster exploration and analysis of design alternatives, enabling the reduction in time for iterations required to assess their impact on overall vehicle attributes.

This project aims to go away from the iterative approach to developing a suspension design and replace it with a novel reinforcement learning (RL) based method that can be trained to understand the relations between the specific suspension design, the corresponding wheel motion curves and the behaviour of the vehicle. Hence, the project aims to prove the concept of making the suspension design quicker in each development loop by successfully training and testing an RL agent that can generate a design given the requirements. The wheel motion curves (referred to as curves, henceforth) can be calculated mathematically by integrating the location and orientation of the wheel over the entire steering (driver input) and jounce (vertical motions) ranges.

1.3. Methodology

The methodology for this project is divided into the following sub-processes:

- Pre-study: Exploration of Reinforcement learning strategies for application on suspension design. Set control and noise states, and penalty functions. Understand the effect of different parameters on suspension curves for use in the optimization function.
- Implementation: Create an optimization function which cooperates well with the current MATLAB script. Quantitative metrics can be obtained from the design and performance requirements for the vehicle and qualitative metrics can be obtained from the feel of the vehicle in driving simulators.

- Post-processing: Plotting pertinent graphs to analyze how the virtual design parameters influence the complete vehicle measures and assess how well the objective targets are met.
- Subjective Testing: The end goal of the project is to alter kinematic curve polynomials. Experienced drivers can assess the suspension design subjectively on the driving simulator using VI-CarRealTime, after selecting/generating appropriate driving scenarios or "load cases".

1.4. Stakeholders and participants

The project is an automotive product development task inspired by research and by the cooperation between Chalmers University and Volvo Cars AB.

Table 1: Stakeholders and participants

Name	Organization	Role
Manuel Denneler	Chalmers	Student/Team member
Christoph Heilig	Chalmers	Student/Team member
Abhishek Kolekar	Chalmers	Student/Team member
Vivekanandan Madhuravasal Narasimhan	Chalmers	Student/Team member
Vinayanand Bangalore Venkatesh Prasad	Chalmers	Student/Team member
Yansong Huang	Chalmers	Academic supervisor
Bengt Jacobson	Chalmers	Academic supervisor
Max Boerboom	Volvo Cars	Industrial supervisor
Tobias Brandin	Volvo Cars	Research project main supervisor

1.5. Project outcome

The project seeks the usage of optimization techniques/artificial intelligence to enable achieving better suspension design in a time efficient way. In many cases, a good performance on one metric may result in a sub-par performance on others, requiring a compromise. In this project we aim to find a method by which the suspension design can be manifested by updated requirements on the suspension and the complete vehicle.

1.6. Project deliverables

- Identify objective requirements on complete vehicle behaviour for stability in longitudinal and/or lateral motion scenarios, using vehicle dynamics simulation.
- Identify/create test scenarios for evaluating the influence from different suspension designs on complete vehicle measures.
- A review of numerical methods (optimization/artificial intelligence/machine learning methods). Select one method to use in present project and identify which have potential for future work.
- Implement an optimization method for obtaining set objective targets on complete vehicle or on suspension. Ensure that the strategy avoids non-feasible solutions.
- Demonstrate the optimization method for one axle on one vehicle.
- Test the optimized vehicle setups on the Volvo Cars Simulator (stretched target for project).
- Improve on the implementation of the optimization strategy for faster computation (stretched target for project).

1.7. Limitations

- Facilities not available on time (simulator): Verification of the models on the simulator must be done by Volvo and simulator verification phase can be delayed or out of the project timeline.
    - Solution: Booking and confirmation of the availability of the simulator must be done well in advance and the models must be ready.
- Scope management issues: Twofold problem of not matching required scope and addition of extra features that are not required.
    - Solution: Scope and deliverables must be clearly listed and agreed upon by all parties.
- Integration issues: Different optimization or AI tools used while developing the optimization strategy might not be compatible with existing and accepted programs or software.
    - Solution: Integration testing must be done before spending time and effort in developing a program or code using a software. Software recommendations from the OEM can be used.
- Data leaks: Internal data from Volvo (vehicle design, parameters, other confidential data), can be unintentionally put on a platform where unauthorized access can occur.
    - Solution: Confidential data must be maintained in a central, non-public location. A list of confidential/internal files can also be maintained to track files.
- Delay due to learning curve: Optimization/AI tools must be used and understood which could lead to delays.
    - Solution: Timelines can be established for learning and using new tools. Experts in the respective tools can be consulted to smoothen the learning curve.
- Only a limited number of complete vehicle measures and suspension design parameters (the coefficients). Only the high fidelity model in VI Car Real Time has been used, i.e. not simplified more approximative vehicle simulation tools.
    - Solution: It is not included to go all the way to real suspension design parameters (such as hard point coordinates), but only to the polynomic coefficients.

1.8. Ethics assessment

Ethical Implications for consequences of the project:
The project's outcome will not only streamline the design process, but also curtail the need for extensive manual iterative phases, leading to notable savings in both time and resource consumption. This would not only streamline the work of engineers engaged in the design process but also substantially enhance overall efficiency. Furthermore, the outcome of this project would integrate with established principles of ethically neutral product development, mitigating any potential ethical concerns.

Ethical Implications for carrying out the project:
It is imperative that due credit is accorded to resources that have been created by others, whether they are textual references or code snippets, as it is paramount in maintaining the team's integrity. This commitment to proper attribution not only upholds ethical standards but also ensures fairness to the original creators. 5

Furthermore, it is prudent to address the responsible usage of large language model tools like ChatGPT. While these tools undoubtedly expedite development and debugging, it is crucial for our team to exercise caution to prevent over-reliance, preserving our capacity for independent problem-solving and innovation. Furthermore, language models usually output erroneous solutions, which cannot be relied upon. Thus, information taken from ChatGPT must be vetted before it can be implemented. AI models do the same mistakes as the human who trained them. Small mistakes from AI at the start of the project can produce a chain of events which can lead to unexpected problems at the later stages.

2. Vehicle development and Simulation

To design a suspension system that gives optimal suspension characteristics, a platform must be created to evaluate these characteristics subject to change in design parameters. This chapter describes the method to generate vehicle models, simulate driving maneuvers, and evaluate vehicle performance characteristics.

2.1. Important definitions

- Toe - Wheel inclination from X axis measured in X-Y plane of vehicle. Unit: [deg]
- Camber - Wheel inclination from Z axis measured in Y-Z plane of vehicle. Unit: [deg]
- Side View Angle - Angle made by the trajectory of wheel center measured from Z axis in X-Z plane. Unit: [deg]
- X-position - Change in X position of wheel center during wheel motion due to jounce and/or steering input. Unit: [mm]
- Y-position - Change in Y position of wheel center during wheel motion due to jounce and/or steering input. Unit: [mm]
- Jounce - Wheel vertical travel. Unit: [mm]
- Coefficients – Coefficients of kinematic characteristics expressed as polynomials. Unit: [-]

2.2. VI-CarRealTime and Subsystem Modification

Since it is required to generate various suspension (kinematic) characteristics without involving the process of physical design (such as hardpoints development), VI-CarRealTime (VI-CRT) was chosen as the tool for vehicle performance evaluation. Unlike tools using multi-body based approach for conducting driving maneuvers, VI-CRT uses "look-up tables" in the form of .XML files for every subsystem to define the vehicle characteristics. Each subsystem in the vehicle, such as front and rear suspension, wheels, powertrain, braking, is represented by a .XML file, which contains information on all vehicle characteristics within the subsystem.

To modify the front suspension and steering kinematic characteristics, a look up table is generated which defines the kinematic curves for each jounce and steering input. To define these curves, each parameter (such as bump steer, bump camber, anti-dive etc.) is expressed as a polynomial curve with respect to an input variable (change in jounce or steering input). The coefficients of these polynomials are parameterized to obtain the desired kinematic characteristics.

Equations of motion are then developed which are used to evaluate the wheel position and orientation as a function of jounce input, and as a function of steering input to obtain the following wheel characteristics: Toe, Camber, Side View Angle, X position, and Y position of the wheel center. The evaluated wheel positions and orientations give two datasets- one with steering as input, and the other with vertical wheel travel as input. To evaluate the state of the wheel subject to both steering and jounce, the wheel positions and orientations evaluated for steer input is added to its corresponding position/orientation subject to jounce input. This gives an approximation of the state the wheel would be for a specific jounce level and steering input.

This combined map of wheel state as a function of steering and jounce inputs is then used to generate .XML file for the front steering subsystem using MATLAB API functions from VI-CRT. Later in the project, an additional parameter, front spring stiffness multiplier, was also added and used to modify the front suspension subsystem .XML file to achieve some performance targets.

2.3. Manoeuvres/Simulation load cases in VI-CRT

After generating vehicle models by modifying their subsystems, it is necessary to assess the performance of each vehicle to understand how the design changes affect vehicle performance. The results from these assessments are then fed into the reinforcement learning agent as observations to tune the characteristics and satisfy all the target criteria. It is imperative to understand how the modification of specific subsystems would affect the complete vehicle; so that some maneuvers can be chosen to assess the degree of change in performance in the complete vehicle when a specific design parameter is changed.

Knowing that the modification in the mentioned design parameters affects the vehicle dynamics characteristics, the task is to choose maneuvers to assess these characteristics. To assess the lateral performance of the vehicle, some handling maneuvers were chosen – Constant Radius Cornering (CRC) and Frequency Sweep. To assess how the suspension characteristics would affect the longitudinal performance of the vehicle, a straight-line raking maneuver was also chosen.

After generating .XML files for the maneuver characteristics chosen, a script was developed on MATLAB to simulate these maneuvers using the modified vehicle model. The simulation results were then post-processed to obtain performance characteristics denoted in the table below.

Table 2 - Description of Driving Manoeuvres and Evaluated Outputs

Driving Manoeuvre	Manoeuvre Characteristics	Evaluated Outputs
Constant Radius Cornering	• Initial velocity 10 km/h • 100m corner radius • Final velocity 80 km/h	Understeer gradient [deg/g] and Roll gradient [deg/g] (roll angle per unit lateral acceleration) for lateral accelerations 0.1-0.35g's
Frequency Response Sweep	• Velocity constant at 80 km/h • Steering angle amplitude: 42° • Steering frequency: 1-4 Hz	• Yaw rate gain margin [dB] • Yaw rate phase margin [rad]
Straight Line Braking	• Initial velocity 90 km/h • Brake ramped to 1 in 1 sec	Pitch gradient – pitch angle per unit longitudinal acceleration [deg/g]

3. Machine learning theory

The theoretical background of machine learning methods is necessary to create a general understanding of how the reinforcement learning based optimization method of this project works.

3.1. Machine Learning concepts

Some important concepts of Machine Learning which are relevant to this project are described briefly in the following section:

3.2. Probabilistic graphical models

Probabilistic graphical models (PGM) are statistical models that use graphs to encode complex probability distributions for multivariate distributions. PGMs can be used to capture a set of independences that hold in the specific distribution. [1, 2] Similar to graphs, PGMs can be directed (Bayesian models) or undirected (Markov Random Fields).

PGMs can be used to find the distribution of one or more random variables (inference) and estimate the parameters of the random variables (learning). Inference in machine learning is used in classification, detection, regression, and identification problems. Learning in machine learning is used in control, autonomous vehicles, pattern recognition and language related problems.

3.3. Weights and biases

Weights and biases are the learnable parameters in a machine learning model. In a PGM representation, the weight controls the strength of the signal between two nodes of the graph and the bias controls the activation of the node to send an output and is a form of a threshold.

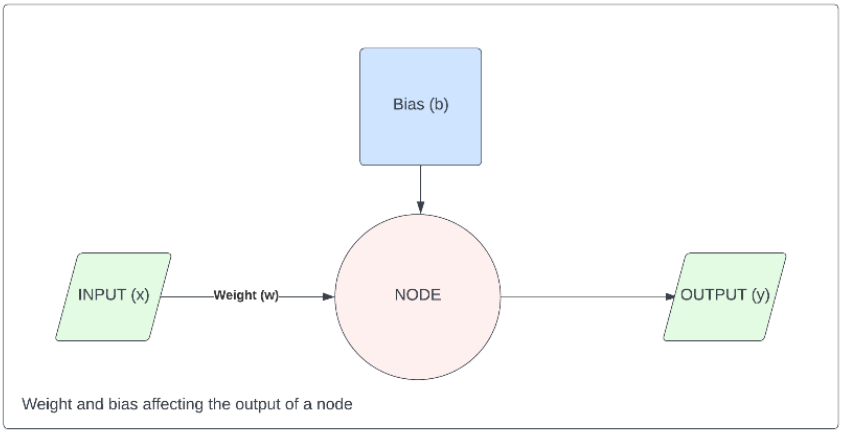

Figure 2 - Weight and Bias Affecting the Output of a Node

From Figure 2, The output of the node is given by $y = f(wx + b)$. The weights and biases are used commonly with an activation layer following the output of the node. Thus, the output of the node is active only when the activation layer is activated (only when the output y is above an activation value).

3.4. Types of machine learning

Machine Learning can be broadly subcategorized into three main types: supervised learning, unsupervised learning and reinforcement learning (RL) [3]. A common example used to explain the different types of learning is the collection of a hundred thousand people and their food buying attributes and their social behavior (input).

Unsupervised learning is used to find patterns or hidden structures in unlabeled datasets that have not been categorized [3]. Using unsupervised learning could group the people or cluster them into similar features (output). Unsupervised learning is used to create a model that takes the features of a collection as an input and transforms it to a single value or a vector (collection of values). [4]
In case of the food buying example, unsupervised learning could classify the buyers into 2 or more groups depending on what they buy. The classification could then be used to predict the future purchases of the buyer based on purchases by other buyers with the same group.

Supervised learning allows for the training of a computer to assign a label to a provided input (data) [3]. To continue with the example before, there are inputs with the food buying attributes, such as the age ranges of our buyers (output). Supervised learning can be used to train a mathematical model to categorize the food buyers into age ranges based on the inputs. By telling the system whether the guessed age range is right or wrong, supervised learning can optimize the model. The model can then be used to predict what a age group a new customer could belong to.

Reinforcement learning is a combination of those two frameworks. While supervised and unsupervised learning work with a static data set, Reinforcement Learning (RL) works with a dynamic environment [3]. The objective here is not data clustering or labeling, but rather identifying the most favorable sequence of actions to achieve the best possible outcome. This essentially translates to maximizing cumulative rewards.
Using the example, reinforcement learning can be used to recommend or discourage certain purchases depending on the expected reaction of the buyer.
In short:

- Unsupervised learning creates a model that can classify input data to different groups or classes. [4]

- Supervised learning creates a model that can label input data based on a set of features. [4]
- Reinforcement learning creates a model that can explore and exploit the environment to gain the most reward/outcome. [4]

Thus, RL presents the best approach to optimize a suspension design based on existing values and data.

4. Reinforcement Learning theory

4.1. Reinforcement Learning concepts

Reinforcement learning is a framework that allows an 'agent' to learn behaviors and actions by receiving 'rewards' after interacting with an 'environment'. The agent, rewards and environment hence represent the most important part of the RL framework. The most interesting aspect of RL is that the actions do not have to be specified explicitly, rather, they are learned over time by interacting with the environment and receiving rewards.

Reinforcement Learning is motivated by the way human and animal behavior. An inexperienced individual takes action that are random and uninformed. With time, the individual learns what actions helps it achieve the goals consistently and reliably and a complex understanding of the environment is built up. [5]

There are two ways an agent can learn under RL. The first method involves searching the action space for an action that yields a good reward. The second method involves using statistical and mathematical measures to judge the value of a particular action. [6]

4.2. Exploration vs. Exploitation

The main point of a reinforcement learning agent is that it must explore the environment and take actions to understand the consequences and receive rewards. Like animal behavior, the agent has the chance to either explore the environment in search of a better action (hence reward) than the one it has, or exploit the current state and take the best reward it has till time.

Another factor to consider is the effect of the delayed reward. The reward the agent receives in a future state is dependent on the current state of the environment as well. If the agent has to take a long sequence of actions to reach the final state, then the initial states will affect the final reward the agent receives.

4.3. Working of RL

At the core of the model lies the "agent," which actively explores, interacts with, and learns from its environment [3]. The agent's actions influence the state of the environment, which, in turn, generates a reward corresponding to those actions. The action of the agent not only affects the immediate reward but also the subsequent rewards [7] as the current action can affect what actions will be available in the future and hence the corresponding rewards. For instance, if an agent moves a walking robot towards a corner, its action space, and hence rewards it can receive for the next action become limited.

By utilizing this feedback and the received rewards, the agent can adapt its future actions, thus acquiring knowledge from the environment to maximize the reward it earns per action.[3] This is presented in Figure 3.

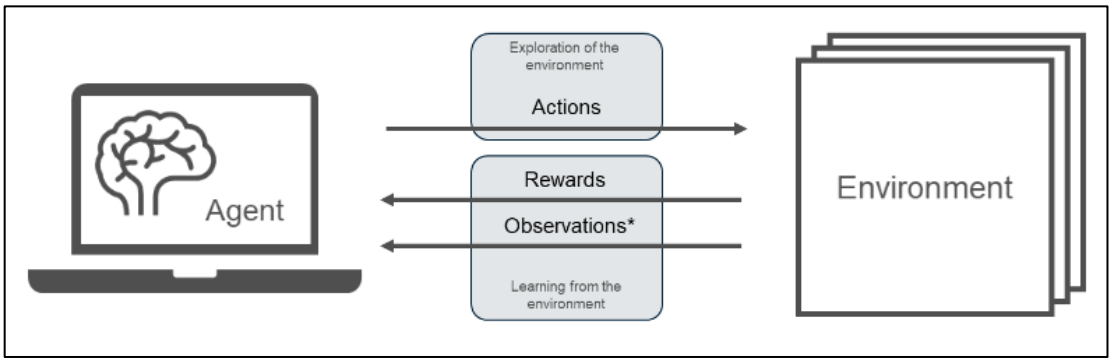

Figure 3 - Overview of the Agent, Environment and Signals involved

In our application of RL to optimize a suspension design, the different key signals are:

- Actions: Set of numerical values that correspond to coefficients in the spline curves. The spline curves describe the motion of the wheel under jounce and steering motions.
- Rewards: A scalar numeric value that is used to quantify how well the action fulfils the complete vehicle target subject to several manoeuvres.
- Observations: Set of numeric values that correspond to the difference between complete vehicle targets and the values obtained from the current coefficients (actions). As the targets and subsystems are interdependent, this helps in establishing relations.

The agent starts with random actions through which it can build up a knowledge of how the environment rewards the actions. With time, the actions of the agent start to resemble a trained agent such that it can take the best actions needed to reach a final state from any starting state.

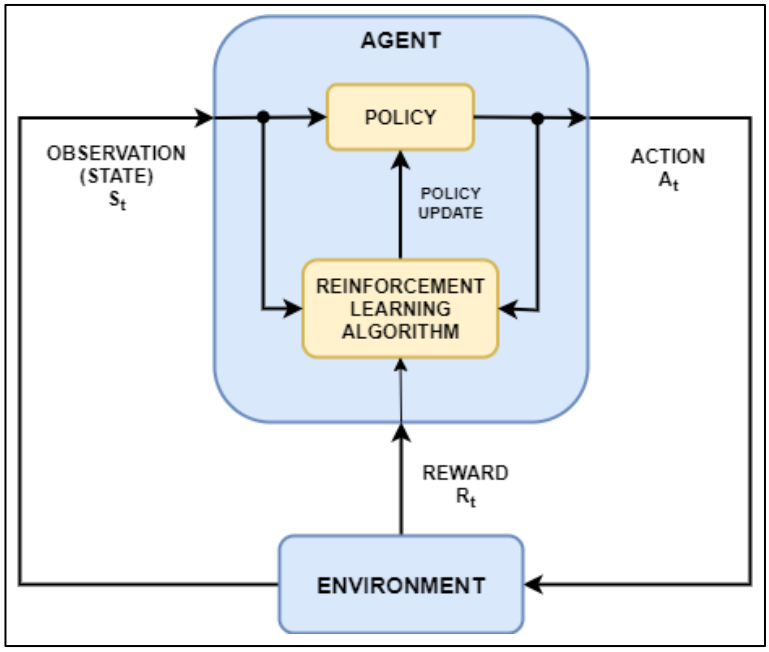

Figure 4 - Policy Update Loop [8]

The agent learns the best action to be taken using a policy. In every iteration of the training loop the policy is updated using an algorithm. The algorithm uses the reward and the observations from the environment to decide the changes to the policy of the agent. Figure 4 represents the training process in a flowchart.

4.4. Elements of RL

Agent refers to the algorithm that is trained in the framework by taking actions and receiving rewards from the environment.
Environment refers to everything within the framework that the agent interacts with in the training process. The environment receives an action, converts that to the next state depending on the current state and returns an observation.
Policy defines how the agent behaves given a state of the environment. In a way, the policy is the mapping from the input (state of the environment) to the output (actions taken). [7]
Action is the output of the agent that is used as the input to the environment to decide a future state and the reward. The action taken depends on the current state of the environment and the policy of the agent. [7]
Reward is the goal of the RL. The agent receives a reward as a single value for every action taken and is the basis to which the policy is altered. The agent aims to maximize the reward achieved by changing its behavior (policy) until it achieves the maximum. [7]
Observations are the information that is sent from the environment to the agent so that it can understand the current state of the environment. The observations may or may not be used and depends on the type of agent used. [7]
Value denotes the cumulative reward that the agent can get in the long run. The agent has to estimate the value of the current state and take actions that

maximize the value as well. Sometimes, the reward for an action may be lower than another action but can lead to a higher value. In situations like these, the agent must learn to compromise and pick the best option. [7]

4.5. Classification of RL

Reinforcement leaning can be classified into different groups. Some of the classifications can be:

Discrete and Continuous action spaces describe the type of actions the agent can take. Discrete space corresponds to some fixed values of action whereas the continuous space corresponds to a range of values for the actions taken by the agent.

Model-based and model-free agents differ on how the environment is used to build up a model from experience that the agent uses. Model-based agents build up a model of the environment and the rewards it yields based on experience and can use it to evaluate future actions. Model-free agents do not have a model of the environment and the rewards and use only the current actions and previous rewards to evaluate future actions.

On-policy and off-policy agents differ on how they use the experience to learn. On-policy agents use the experience to improve the current policy whereas off-policy agents use the experience to develop the optimum policy. As a result, on-policy agents tend to converge faster but tend to be stuck in local minima.

4.6. Reinforcement Learning workflow

The following diagram (Figure 5) highlights the RL workflow used in this project. Reinforcement learning is generally used for control related problems and hence not much literature or previous solutions exist on our application.

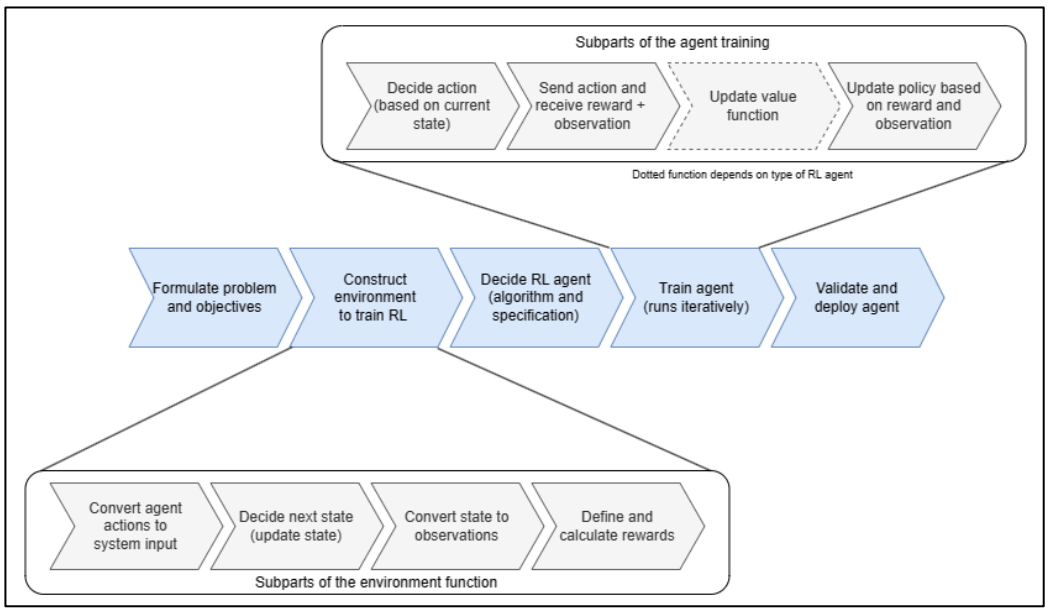

Figure 5 - Workflow of the Reinforcement Learning Algorithm

4.6.1 Formulation of the problem

The problem in the project corresponds to the optimization of a front suspension design subject to complete vehicle targets. The objectives of the project are to successfully train an RL agent to optimize the suspension design and make sure the proposed suspension design satisfies the physical constraints controlled by the packaging requirements and mechanical limits.

The complete vehicle targets are calculated from several measures that are calculated from manoeuvres described in Table 2 - Description of Driving Manoeuvres and Evaluated Outputs
. Complete vehicle targets are:

1. Lateral measures
    a. Understeer gradient
    b. Roll gradient
2. Steering response measures
    a. Yaw rate gain margin
    b. Yaw rate phase margin
3. Longitudinal measure
    a. Pitch gradient

The actions (coefficients of the spline curves) represent the following motions:

1. Bump steer (change of tow with jounce)
2. Bump camber (change of camber with jounce)
3. Anti-squat
4. Anti-lift
5. Roll center height

4.6.2 Reinforcement Learning environment

The environment function represents the external world or model that the agent interacts with to learn. The agent interacts iteratively with the environment using actions for which the environment presents rewards and observations to the agent. The system that the agent works with (simulation of a vehicle model, in our case) is a part of the environment function.

The functions of the environment can be divided into:

1. Converting agent action to a system input
2. Deciding the next system state using the agent input
3. Converting the system states into observations for the agent
4. Calculating the rewards for a particular action from the new state

The following flowchart (Figure 6) describes the processes in the environment function as pertaining to the project.

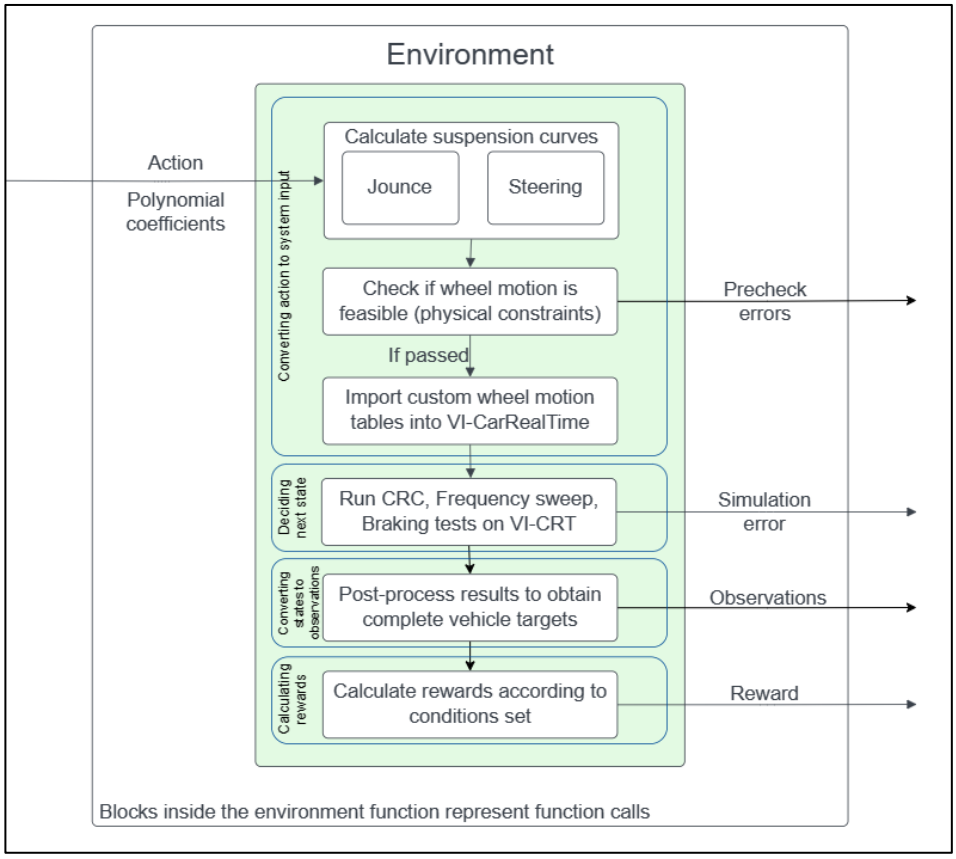

Figure 6 - Overview of the RL Environment

4.6.3 Reinforcement Learning agent

It has been decided to use the DDPG (Deep Deterministic Policy Gradient) agent as it meets all the requirements: continuous action and observation spaces, actor-critic network for memory efficiency and data buffer to learn from previous experiences.

The reinforcement learning agent will be used to optimize the values of the polynomial coefficients describing the wheel motion. The wheel motion can be calculated in jounce and in steering and the combined motion is obtained by the superposition of individual motions. There are five polynomials describing the wheel motion in jounce and five polynomials for wheel motion under steering. Since each polynomial is a third order equation with no constant terms, the RL agent will have to find the optimal values for 30 variables. To simplify the process, we have decided to optimize the quadratic and linear coefficients for only the jounce polynomials.

4.6.4 Training of Reinforcement Learning agent

Training of the RL agent is done in MATLAB using the Reinforcement Learning toolbox, [9] and VI-CarRealTime [10] (VI-CRT) from VI-Grade to simulate the complete vehicle.

MATLAB was chosen as all the team members had experience using MATLAB and the RL toolbox is well-documented and has many different types of agents that can be used. There also exists an API (application programming interface) for VI-CRT that can be used directly in MATLAB to call the simulation routines with the modified data files. Following is the logic flow of the training process:

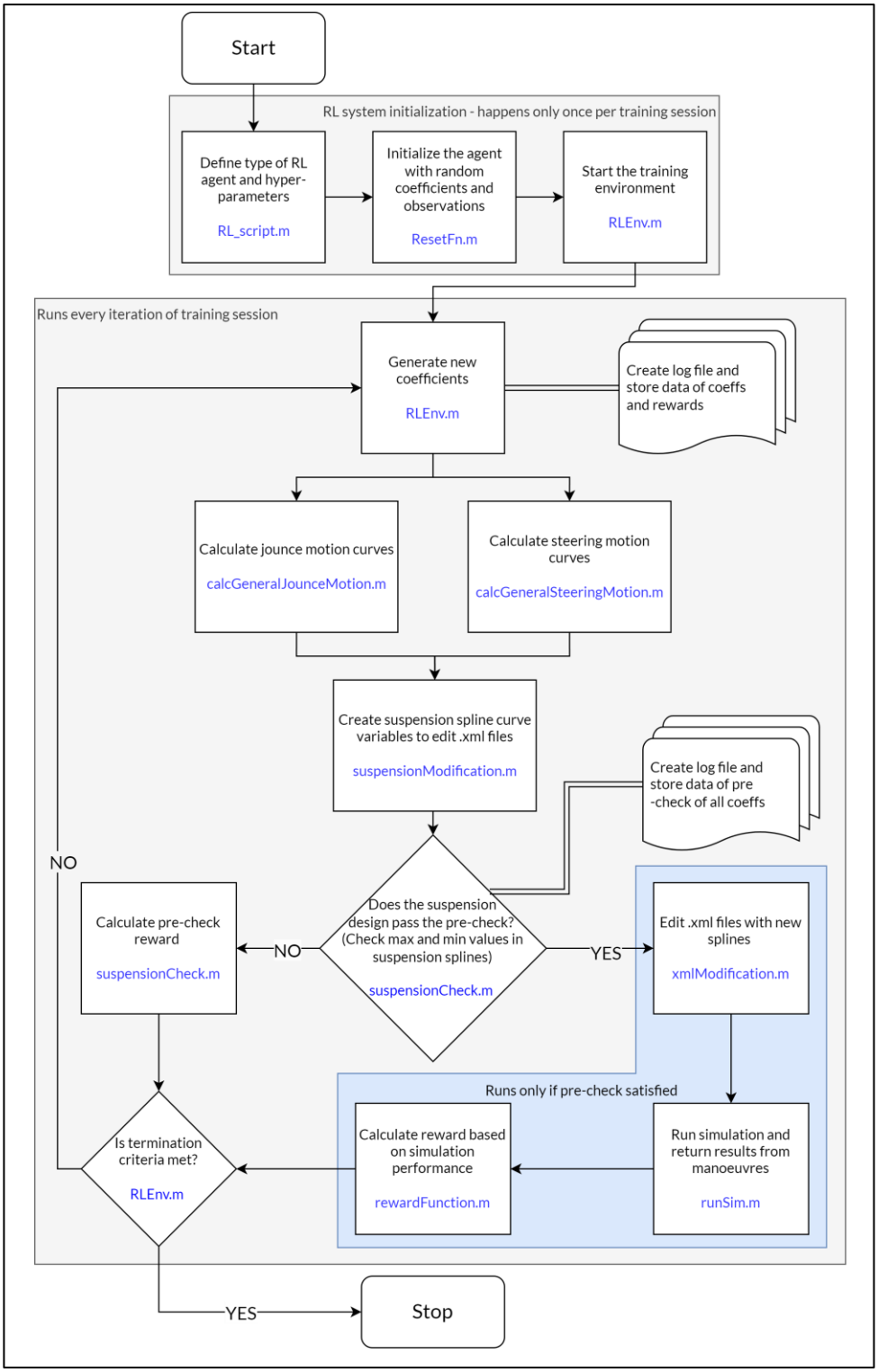

Figure 7 - Flow of the Reinforcement Algorithm in Detail

The purpose and working of the different MATLAB functions are explained in Section 6 - MATLAB Script Development and Description.

.

4.6.5 Validation of trained agent

A separate standalone script was developed to test the outputs of the trained agent. Since we were never able to fully train an agent, we used the logged data from the training sessions to validate the training process. The environment was set up in such a way that coefficients that performed well were logged into a test file so that they can later be tested out.

The validation script uses maneuvers that can test the vehicle behavior at extreme jounce and steering values to make sure the coefficients agree with physical constraints.

5. RL vs. Other Optimization Methods

Traditional optimization methods (gradient based) cannot be used in our problem as the system to be optimized cannot be represented as a mathematical function. Evolutionary optimization can be used, however they present the challenge of requiring large amount of data and iterations to reach the optimum value. Since the population of potential solutions in evolutionary optimization methods is high and each member of the population needs to tested, the total time taken for one iteration goes up proportional to the members in each generation.

Another issue with optimization (both traditional and evolutionary) is the need to mathematically rewrite and run the optimization of the objective function for every new requirement or constraint. With RL, one would simply have to add additional requirements to the reward function and constraints to the action space.

Another drawback to optimization is that the convergence of solution to the optimum depends considerably on the starting point. In a population, many starting points may not be feasible and need to be checked before using in the simulation.

These concerns are eliminated in a reinforced learning algorithm as the agent requires only one evaluation per iteration. Since it can develop policies on the solution, it can quickly learn to avoid infeasible points if rewarded and penalized correctly. RL algorithms also store previous experiences and build on that. Another advantage is that time to convergence to an optimal solution does not depend much on the starting point. As a result, the same algorithm can even be used to quickly find the optimal solution to similar problems. This allows a trained RL agent to be reused for different optimization problems as long as the actions and rewards remain the same.

6. MATLAB Script Development and Description

6.1. Development of MATLAB Script

The RL algorithm has undergone significant changes since the start of the project, as the team developed new insights on the working of the RL "black-box". The following table briefly summarizes the upgrades that have been implemented:

Table 3 - Updates in different versions of the RL Algorithm

Version	Changes
v1	Basic RL setup to verify error-free training
v2	Updated rewards function
v3	Errors in targets added as an observation to the agent
v4	Action space is the new coefficient, instead of change in coefficient which is [Δ coefficient = new coefficient – old coefficient]
v5.1	Pre-check implemented, error in pre-check added as observation to agent
v5.2	Updated rewards function (testing with the non-linear proportionality type, which is explained in Section 6.3 - Rewards Function). Evaluating effect of changing the balance between the exploration & exploitation by experimenting with greedy and exploratory policies.
v5.3	Suspension stiffness multiplier added as a new action

6.2. Description of Different RL Functions

The different functions mentioned in the logic flow in Figure 7 - Flow of the Reinforcement Algorithm in Detail are described as follows:

6.2.1 Main Script (RL_script.m)

This is the main script that serves the purpose of setting up and training a reinforcement learning (RL) agent for the project, and it includes the following:

1. Housekeeping and Log Files Removal:
    - Clears the console and workspace.
    - Deletes previous training log files if activated.
2. Definition of Observation and Action Spaces:
    - Defines observation space representing the coefficients, errors in precheck and errors in simulation.
    - Defines action space as coefficients for polynomial curves affected by varying jounce, as well as the spring stiffness multiplier in the later training.

3. Environment Setup:
    - Configures and initializes the RL environment using observation and action space definitions, as well as the reset function and the contents of environment file.
4. Reinforcement Learning Model Setup:
    - Describes the architecture of the actor network responsible for generating continuous actions.
    - Describes the critic network that evaluates the quality of actions.
        i. Defines a Deep Deterministic Policy Gradient (DDPG) agent using the actor and critic networks.
5. Hyperparameter Tuning:
    - Sets hyperparameters affecting RL learning and convergence.
6. Training the RL Agent:
    - Sets training options, including the number of episodes, steps per episode, and stopping criteria.
    - Initiates the iterative training process for the RL agent.

6.2.2 Reset Function (ResetFcn.m)

The reset function is designed to set the initial state of the reinforcement learning (RL) environment. It is executed only once during the first iteration of training, initializing key variables for the RL environment.
It initializes the observations for the first iteration as an array called init. The first 10 elements of the init array are the jounce related coefficients – considering a quadratic equation of the form: $a * x^2 + b * x + c = 0$, the odd elements of the array are the linear coefficients of the quadratic equation i.e., the b term and the even elements are the constant i.e., c term.

6.2.3 Environment Function (RLEnv.m)

This function acts as the external model, representing the "world" with which the RL agent interacts. It is akin to a "plant" in control systems. It manages interactions between the RL agent and the simulated environment. The working of the environment function can be described as follows:
1. Kinematic Parameter Calculation:
    - Calculates wheel center motion under jounce and steering based on coefficients inputted by the agent.
    - Calculates kinematic parameters (e.g., camber, toe) using wheel motion data.
2. Suspension Pre-check:
    - Certain sets of coefficients can give kinematic parameter values which are not possible considering the physical constraints of the vehicle. Simulating this in VICRT would be a waste of computation time and

resources since these sets of coefficients cannot be used in the vehicle development process. Therefore, it is beneficial to check whether the kinematic parameters are within certain ranges, and then proceed to simulation, which is what the pre-check does. This is explained in Section 6.2.6 - .

- Suspension Pre-Check.

3. System Modification and Simulation:
   - Modifies the vehicle subsystem XML file(s) (front-suspension and steering files) if the pre-check is passed.
   - Runs simulations for various load cases using VIRCT.
   - Evaluates the simulation results using a reward function.
4. Logging and Outputs:
   - Saves the best set of coefficients (with the highest reward) of the training session in a file.
   - Logs rewards, coefficients, pre-check pass/fails and iteration information in separate files.
   - Outputs the observations (current set of coefficients, errors in pre-check, and errors in targets).

6.2.4 Calculate Wheel Motion (calcJounceMotion.m & calcSteeringMotion.m)

These functions are derived from calcGeneralMotion.m file which was provided to the team at the start of the project. The purpose of these function is to simulate the effect of wheel jounce and rebound, as well as steering characteristics using coefficients to influence motion. The agent takes an action, which is to create a set of jounce coefficients, which is stored in the coeffs array. The nomenclature of the coefficients used is as follows:

```
Jounce Coefficients
coeffL1 = [0, coeffs(1), coeffs(2)];  % Bump Steer
coeffL2 = [0, coeffs(3), coeffs(4)];  % Bump Camber
coeffL3 = [0, coeffs(5), coeffs(6)];  % Anti-squat
coeffL4 = [0, coeffs(7), coeffs(8)];  % Anti-lift
coeffL5 = [0, coeffs(9), coeffs(10)]; % Roll centre height

Steering Coefficients - these are frozen and are unchanged by the RL
coeffL6 = [0, 18/10, 0];              % Hub trail
coeffL7 = [g2(1), g2(2), g2(3)];      % Kingpin offset
coeffL8 = [0, 8/10, -25];             % Caster trail
coeffL9 = [g1(1), g1(2), g1(3)];      % Scrub radius
coeffL10= [g3(1), g3(2), g3(3)];      % Wheel load lever arm
```

The working of the functions is as follows:

1. The function uses coefficients to extract kinematic characteristics. The first step is to split all the array of coefficients for each separate spline.

2. Formulates equations of motion and evaluates derivatives for wheel position and orientation.
3. Integrates to obtain wheel position and orientation as a function of jounce levels and input steering.
4. Returns the wheel_center and wheel_orientation vectors with X, Y, and Z coordinates, camber, side view angle, and toe angle as functions of wheel travel (for jounce) and steer angle (for steering) respectively.

Currently the project scope is limited to optimizing the jounce coefficients. However, as seen in Section 7 - Results, considering the limited impact of selected coefficients, other parameters (e.g., suspension stiffness) can also be added as an action to make the optimization wider in scope.

6.2.5 Generating Kinematic Data for VI-CRT (suspensionModification.m)

This function converts suspension kinematics data from wheel jounce and steering simulations into the format required by VICRT – the software requires the kinematic parameters as curves, as a function of either steering wheel angle or rack travel. This is done as follows:

1. The function extracts X, Y, Z coordinates, camber, side-view angle, and toe angle against jounce and steering from the wheel_center and wheel_orientation vectors from the previous function.
2. It then transforms and adjusts coordinates to VICRT's coordinate system and units, and superimposes steering data for every jounce step, creating a pseudo-3D map of kinematic parameters.
3. The 3D map is reduced to steer sweep characteristics at intervals of 5 data points in jounce travel. This is due to VICRT's limitations on handling large datasets.
4. The output is a suspension_data structure which contains all the data which can be used in VICRT.

6.2.6 Suspension Pre-Check (suspensionCheck.m)

As described briefly in Section 6.2.3 -
Environment Function (RLEnv.m), the pre-check function is used to check if the static values of the suspension curves lie within the limits defined by physical constraints. After integrating the curves over the entire jounce and steering action, the extreme values in both directions (compression and extension for jounce and lock-to-lock steering) are compared to defined values. If the values are exceeded, the model is not simulated. This saves the computation time by avoiding simulation of the infeasible sets of coefficients.
The static limits considered are:

- Camber: ±15°
- Side-view angle/Caster: ±15°
- Toe: ±50°

- Wheelbase variation: ±75mm
- Track variation: ±75mm

The function works as follows:

1. The data from suspension_data (output of the previous function) is checked if it violates the static limit definitions for the minimum and maximum values (compression/extension for jounce, left/right for steering) for both the left and right front wheels. The pre-check outputs 1 if all kinematic parameter values are within range (this is pre-check pass condition) and 0 if any value is not in range (this indicates pre-check fail).
2. Also calculates the error in pre-check i.e., the difference between the boundary of the acceptable range of a kinematic parameter and its calculated value. It the pre-check is passed; the errors are set to 0.
3. The function also maintains a count of consecutive pre-check failures for potential RL penalty, which can be used to penalise the agent if required.

6.2.7 XML Modification (xmlModification.m)

VICRT requires the data to be in separate XML files for different subsystems, with a "master" XML file being used to reference the correct subsystem files. This function facilitates the replacement of kinematic curve data in the Steering subsystem XML and Front Suspension XML files for each iteration of the training, and it works as follows:

1. Data Import and systemStruct Generation:
    - Imports the standard vehicle model as a baseline using VICRT-MATLAB API and keeps the model as a structure called systemStruct.
    - The suspension_data from suspensionModification function is also taken in as an input.
2. Modifying Spline Data:
    - The code modifies data from the suspension_data struct into the systemStruct. This facilitates the replacement of data for each iteration of the training.
3. Modifying Other Data:
    - Sets steering type to Simplified.
    - Allows a switch for using either steering wheel angle or rack travel as an input.
    - Assumes constant compliance.
    - Modifies suspension stiffness using a multiplier, which can be influenced by RL agent actions.
4. Changing XML Files:
    - The function then finalizes the systemStruct and modifies the steering and front suspension subsystem XML files.

- It also updates the vehicle model XML file to reference the modified steering and front suspension subsystem XML files into the simulation.

6.2.8 Run Simulation & Post Process Targets (runSim.m)

This function initiates simulations in VI-CRT calculates relevant performance metrics for calculating the reward for the iteration.

1. Checking for Simulation Errors:
   - Reads simulation log files and outputs the variable sim_error as 1 if any log contains the string 'ERROR'. This flags simulations that fail to reach a steady-state equilibrium. The sim_error was used before the precheck was implemented to penalize the actions which cause physically infeasible vehicle wheel splines. However, since the precheck is added, the sim_error is obsolete, since iterations having infeasible set of coefficients are not simulated in VICRT. It is still retained, to have that functionality, in case any simulation error occurs even after precheck.
2. Constant Radius Cornering (CRC) Load case:
   - Calculates *Understeer Gradient* and *Roll Gradient*.
   - Evaluates standard deviation of lateral acceleration to check for oscillatory behavior, i.e., to check for roll oscillations. Flags iterations which cause roll oscillations, for penalizing later.
3. Frequency Response Sweep (SWE) Load case:
   - *Yaw Rate Gain*: Computes gain by analyzing peaks in yaw rate and steering rack displacement.
   - *Steering Response Delay*: Determines the phase delay between yaw rate and steering rack displacement.
   - *Yaw Rate Gain Margin*: Identifies frequency at which phase reaches $-\pi$.
   - *Yaw Rate Phase Margin*: Analyzes the phase at zero decibel gain.
4. Straight Line Braking (BRA) Load case:
   - Calculates *Pitch Gradient* during braking.

These help to analyze the handling, stability, and ride comfort characteristics of the vehicle setup in an objective manner.

6.3. Rewards Function

The reward function is arguably one of the most important functions inside the reinforcement learning environment. The reason for this is that this is the only signal to the reinforcement learning agent if the actions it is taking leads towards desirable outcomes. The agent calibrates the weights and biases inside the neural networks depending on the correlation between the observations and the rewards. Thus, the rewards have to be made such that is has obvious (to the agent) associations with the observations amongst other constraints set by the agent and the rest of the environment.
One of the constraints is that the reward function must be smooth and continuous, the benefits of which are described below:

1. Gradient Continuity: DDPG is a gradient based algorithm, which means that it updates the policy according to the gradient of the actions and their consequent rewards of the current iteration in relation to the previous iterations. Having a smooth reward functions lead to continuous gradients, which allows the algorithm to have well-defined gradients throughout the state and action spaces. This in turn helps the algorithm to converge faster than if the rewards were discontinuous. [11]

2. Gradient Descent Stability: Smooth rewards reduce the likelihood of sudden changes or discontinuities in the gradient landscape, which improves the stability of the algorithm. [12]

3. Improved Exploration-Exploitation Tradeoff: A smooth reward function provides a more gradual transition between good and bad actions. This encourages the learning agent to explore a broader range of actions, aiding in the exploration-exploitation tradeoff. [13]

4. Better Generalization: Smooth rewards contribute to better generalization. The learned agent is more likely to generalize well to unseen states, as the smoothness helps the model to understand the underlying structure of the environment.

The other constraint is that the agent should be rewarded/penalized in such a way that it is easy for it to recognize the consequence of its actions, i.e., the reward should help the agent easily acknowledge whether it is moving forward in the correct direction or not.

Considering these constraints, two different types of rewards functions were tested. The reward for each vehicle level target being achieved is described by the functions below. After that is the description of how the reward for an iteration is calculated.

Both functions are smooth and continuous, but use different mathematical equations to describe them:

- Non-linear proportionality: The proportionality, in this case, is described by the hyperbolic tan function, with a negative penalty being applied if the result is outside the acceptable range.

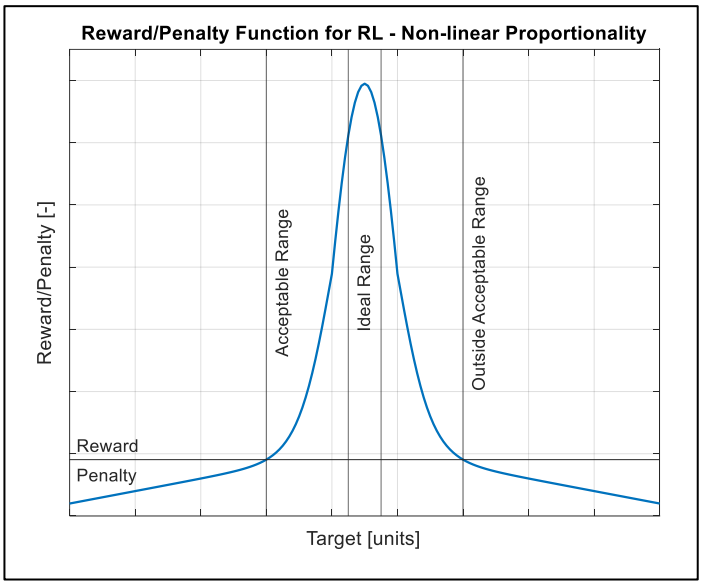

Figure 8 - Reward/Penalty Function - Non-linear proportionality

- Linear proportionality: The proportionality, in this case, is described by the slopes of a straight line.

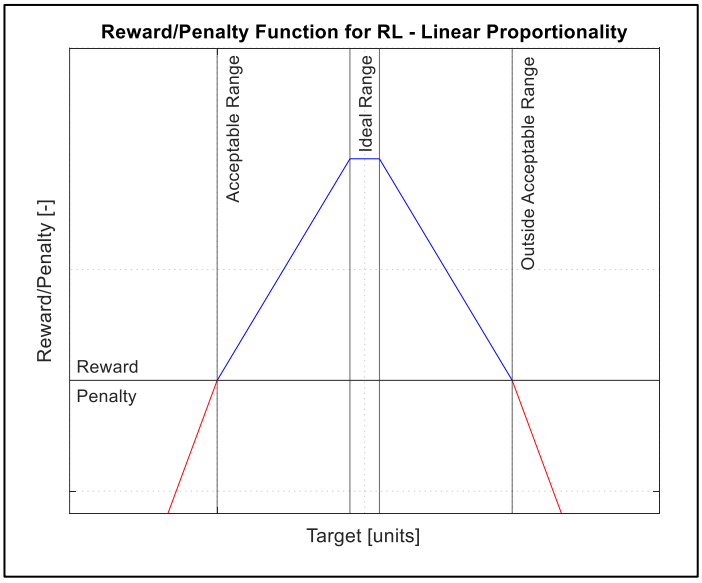

Figure 9 - Reward/Penalty Function - Linear proportionality

The different rewards functions were set up differently to evaluate the impact of proportionality on how the agent trains. The scaling (actual numeric value) of the reward and penalty, as well as the proportionality (slope) was modified for different training sessions.

A brief description of the how the total reward is calculated is as follows:

- Calculating the baseline reward: The reward "curves" described in Figure 8 and Figure 9 are set for only one target, and their outputs would be a single scalar value in relation to how well a target has been achieved, with the highest reward being awarded for being within the ideal bounds. This value is computed for all the five targets set, the addition of which gives the baseline reward.
- Checking for oscillatory behaviour: In the earlier training sessions, it was found that some set of coefficients resulted in getting all targets within the acceptable (or even ideal) range but resulted in the vehicle experiencing roll oscillations. To avoid this behaviour being rewarded, the reward function was updated to check if the standard deviation of lateral acceleration is high, in which case, a penalty is applied instead of a reward.
- Calculating Target In-Range Bonus Rewards: The objective of the RL is to get a set of coefficients that gets most (if not all) targets inside the ideal range. To incentivize this, a bonus reward is granted based on two conditions related to the counts of achieving targets within the acceptable and ideal ranges:
    - The function count's how many targets are within the ideal and acceptable ranges each.
    - If the count of acceptable targets or ideal targets is greater than 1, then the count is multiplied by a bonus value.
    - This is then added to total reward.
- Calculating Kinematic Parameter In-Range Bonus Rewards: The acceptable range for the kinematic parameters (KPs) such as camber, SVA, toe, etc., in the pre-check is quite large (this can be termed as acceptable KP range) to get the agent to explore a bigger larger space. However, it is more beneficial if the KPs are within tighter bounds (these can be called ideal KP range). To adhere to both constraints, there is no bonus or penalty for acceptable KP range, but there is a bonus reward for each KP being inside the ideal range. This is done by counting the number of KPs in ideal range and multiplying the count with a bonus value.
- Calculating Kinematic Parameter Out-of-Bound Penalty: In contrast to the previous case, it is important for the agent to know if the action it takes shift the KP outside the acceptable range. Thus, if pre-check is failed, then the number of KPs outside the acceptable range is counted and multiplied by a penalty value.
- Computation Expense Penalty: This is an optional penalty added to help the agent understand that each cumulative time it fails the pre-check, it incurs a penalty. The purpose is to convey to the agent that computation time needs to be shortened. If the penalty continues to increase, it indicates that the agent should explore a different part of the action space.

All the above points give a single scalar value each, which is added up to get the total reward.

7. Results

Multiple training sessions were done with the agent, each time updating the inputs and reward function as per the findings from the previous training. The setups and their results are described:

7.1. Training Agent 1 (v5.1)

- Training Setup:
    - Observations:
        - Coefficients
        - Error in simulation
        - Error in precheck
    - Actions: New coefficients
    - Reward: Nonlinear proportionality (tanh)

- Takeaways
    - Agent explores only limited range in the action space: By checking the logs, it was observed that the agent moved back and forth between the same set of coefficients and was unable to explore the action space in its entirety.
    - No positive rewards: The agent was unable to get any set of coefficients that could get any target within the ideal range. It could, however, get one or two targets in the acceptable range, but this is not enough to get a net positive total reward. Moreover, it is evident from Figure 10 that it could pass pre-check successfully for multiple iterations – *PP indicates pre-check pass* and *PF indicates pre-check fail.*
    - RG and Ku targets were not met with any set of coefficients: While the RL agent could find coefficient sets that could satisfy the pitch gradient, yaw rate gain and phase margin targets (into its acceptable range at least), it could not find any coefficients that could satisfy the roll gradient and understeer gradient targets.

- Learnings
    - Reward function (the non-proportional kind) may not be giving agent the expected feedback/motivation.

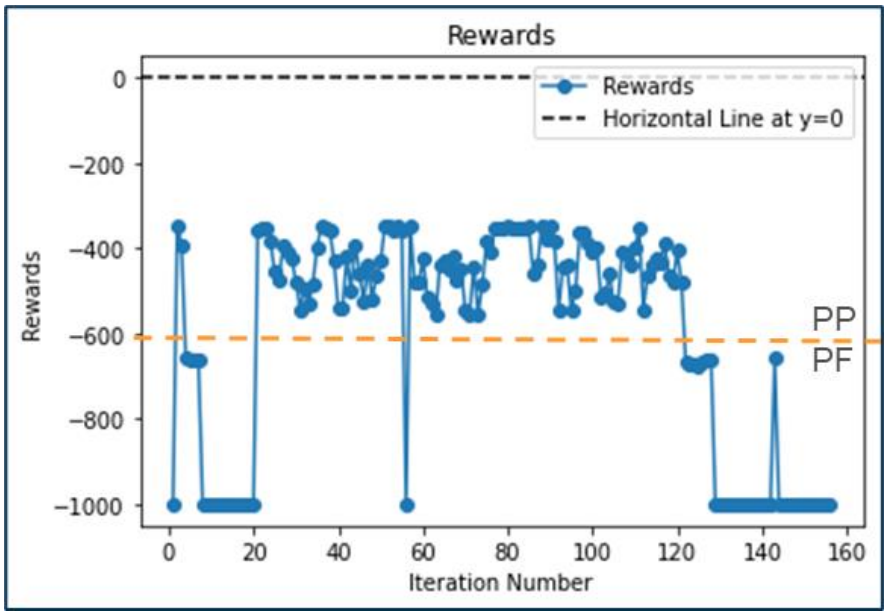

Figure 10 - Training of RL Agent 1 (v5.1)

7.2. Training Agent 2 (v5.1)

- Training Setup:

The agent used is the exact same as the previous one, but the reward function has been changed, which is why the version number is same.

  - Observations:
    - Coefficients
    - Error in simulation
    - Error in precheck
  - Actions: New coefficients
  - Reward: Linear proportionality
  - Changes: The type of reward has been changed.

- Takeaways
  - Agent explores the action space: By changing the reward function to a linear proportionality kind, it was observed that the agent moved through the action space quite more as compared to before.
  - Still no positive rewards: The agent showed similar results as last training, which is seen in Figure 11.
  - Agent revisits 'safe point' after failing to pass precheck: A key takeaway from this training was that the agent was able to backtrack to its "safe-point", i.e., the point in the action space which it knows passes the pre-check, and it can explore a different part of the action space from that safe point. This is seen in the logs of the RL training.
  - RG and Ku targets were not met with any set of coefficients: Shows similar takeaway as the last training session.

- Learnings
  - Reward function may work as expected but tuning needed to push the agent in the right direction.
  - Agent retains memory of past rewards and backtracks.

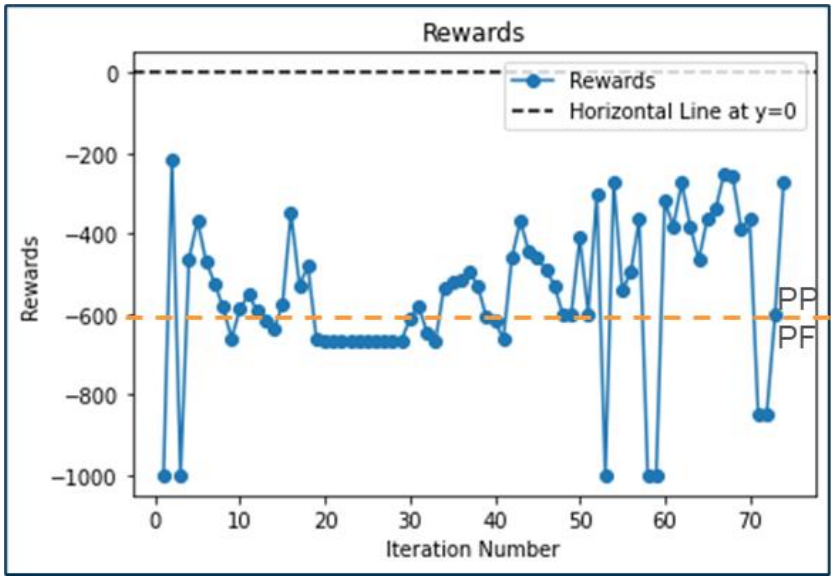

Figure 11 - Training of RL Agent 2 (v5.1)

7.3. Training Agent 3 (v5.2)

- Training Setup
  - Observations:
    - Coefficients
    - Error in simulation
    - Error in precheck
  - Actions: New coefficients
  - Reward: Linear Proportionality (the scaling and the way the penalties are applied is changed).
  - Policy: Changed to Greedy/Exploitation Policy: The greedy approach selects the action with the highest estimated reward most of the time. [14]

- Takeaways
  - Decent number of iterations with high rewards: As seen in Figure 12 (L), the agent can explore the action space progressively and is able to find sets of coefficients that can satisfy multiple targets.
  - Ku and RG values still not satisfied: The logs reveal that none of the coefficient sets are able to satisfy the roll gradient and understeer gradient targets.
- Learnings
  - Reward tuning is in the right direction as agent explores high reward points

- Since the "good" sets of coefficients are not able to make significant changes to the understeer gradient and roll gradient, it leads to the conclusion that some other suspension constants may have to be changed (stiffness/compliance) to affect those targets.
- Even though it seems like the agent is on a positive trend, it can be seen in Figure 12 (R) that the agent starts giving bad sets of coefficients and is unable to recover. Thus, maybe stopping and retraining the agent might be better than keep it training for a longer duration.

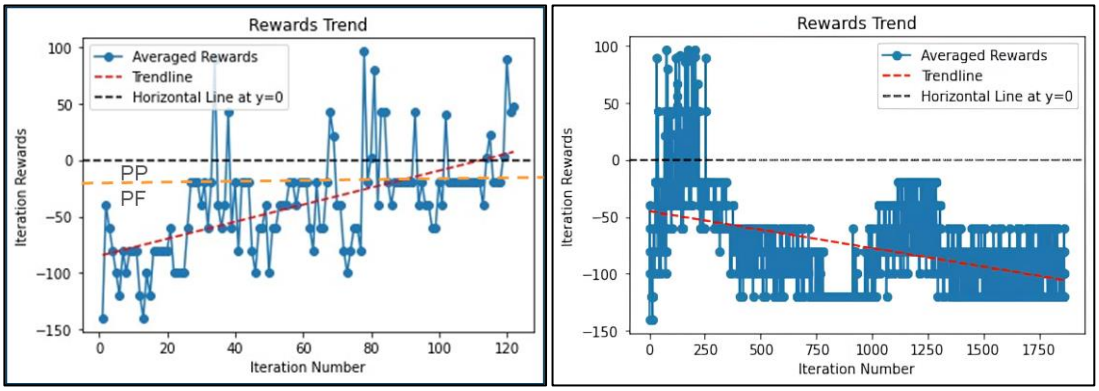

Figure 12 - Training of RL Agent 3 (v5.2) for ~120 iterations (L); Same agent run coninuously for ~1 800 iterations (R)

7.4. Training Agent 4 (v5.3)

- Training Setup
  - Observations:
    - Coefficients
    - Error in simulation
    - Error in precheck
  - Actions: New coefficients, spring stiffness multiplier
  - Reward: Linear Proportionality (the way penalty is applied is modified again)

Note: The spring stiffness is not directly changed for each iteration. In VI-CRT, there exists an option to change a multiplier for the spring stiffness. The multiplier is changed for each iteration, varying up to $\pm 30\%$ of the initial stiffness.

- Takeaways
  - Agent explores the action space without 'oscillating': It does depend on the initial seed in the action space; however, it is seen in Figure 13 that there is little oscillation in the rewards, which means that the agent finds the part of the action space that can successfully satisfy multiple targets.

- RG and Ku targets can be met by changing stiffness: The logs show that changing the stiffness gets the roll gradient and understeer gradient targets in the acceptable range, which is a leap forward in the right direction.

- Learnings
  - This reward function is more in line with what the agent expects, and further tuning may improve it. However, tuning further that this would just be getting closer to a point of diminishing returns, where the result may not be significantly impacted.
  - The agent starts giving bad sets of coefficients and is unable to recover. Tuning hyperparameters such as exploration decay parameter influences when the agent can explore no more, and what actions it can take at that point is important in further study of the RL.

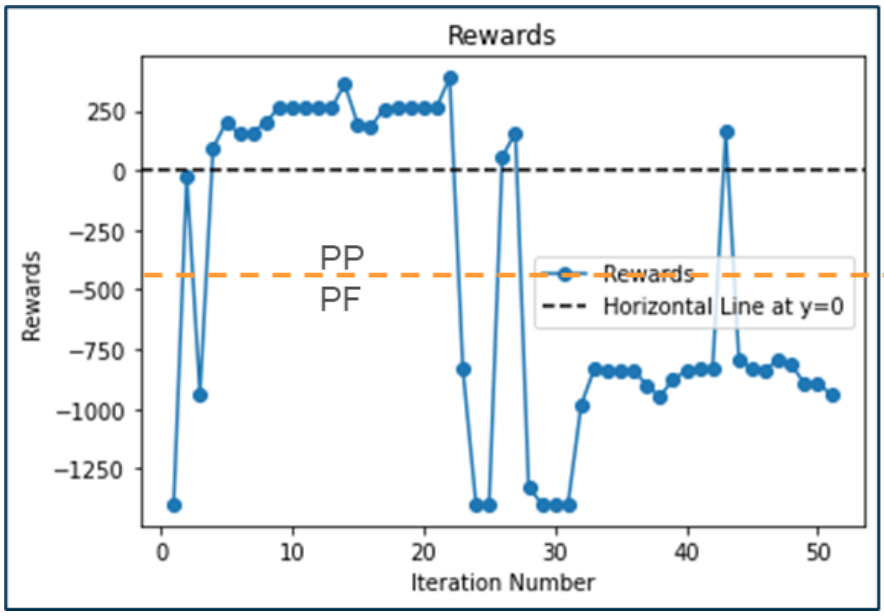

Figure 13 - Training of RL Agent 4 (v5.3)

8. Session at the car simulator centre at Volvo AB

The preceding chapters primarily present the theoretical development and simulation using the MATLAB and VI-CarRealTime programs. In addition to these theoretical components, a comprehensive development involves practical verification using real simulators. This allows establishing the connection between subjective driving perception and objective measurement parameters. The practical application of the previously programmed vehicle parameters makes them tangible and quantifiable.

During the implementation of a reinforcement learning (RL) method to enhance a suspension design, it was possible to visit and test the physical simulator at Volvo Car Corporation. The task involved equipping a predetermined vehicle model with different suspension parameters and subsequently perceiving these differences on the simulator through a track to be driven.

Figure 14: Vehicle Dynamics simualtor at Volvo Car Corporation [15]

The use of a vehicle dynamics simulator provided valuable insights, highlighting a critical aspect: constructing a car based on simulation results does not inherently ensure drivability or subjective excellence. The experience emphasized that the subjective feel of a car transcends mere kinematic curves. It involves a complex interplay of various elements, including springs, dampers, anti-roll bars, and numerous intricate factors. The simulator facilitated an exploration of vehicle dynamics intricacies, demonstrating that achieving a harmonious and subjectively satisfying driving experience necessitates a comprehensive understanding of the intricate relationships among diverse mechanical components. This underscores the significance of integrating theoretical insights from simulations with practical, hands-on exploration to engineer vehicles that excel not only in performance metrics but also in providing an enjoyable driving experience.

9. Observation

The performance of agents in reinforcement learning is a new complex phenomenon in which the starting point and the random decision play an important role. The initial condition plays a crucial role in the agent's decision making and learning process, influencing its ability to adapt and optimise outcomes. Furthermore, the variations in the random initial conditions lead to a significant degree of diversity in the learning process, resulting in different outcomes and behaviours.

In an attempt to improve the agent's performance, changing and interacting with the basic hyperparameters proves to be a practical and effective strategy. Manipulating these parameters helps to fine-tune the learning process and enables improved efficiency and effectiveness.

A particular observation with different agents is the presence of "oscillating" behaviour. This behaviour manifests itself in the form of fluctuations between two sets of coefficients and illustrates the dynamic nature of the learning process and the continuous adaptation of the agent to its environment. Eliminating these fluctuations can contribute to the success of the learning process and improve computing time.

Despite attempts to optimise performance, agents, even those that are considered promising, may experience "crashes". These crashes occur when the agent abandons exploration and chooses one of the boundaries of the action space as the final action. Such occurrences hinder further learning and require intervention; to counteract crashes and maintain the learning process, a practical approach is to stop, save and restart the training in time. This intervention interrupts a possible stagnation and enables the agent to resume the exploration and avoid a premature cancellation. To summarise, the success of the learning system is determined by a strong understanding of the learning process and the decisive factors are important for programming a new RL agent.

10. Conclusion

This project represents a first draft that forms the basis for a fully developed and multi-dimensional program. The main objective of this project is to improve suspension development. The Reinforcement Learning (RL) program is used to identify and analyze possible configurations. As there is currently no agent designed for this problem on the market, the code is programmed using an agent originally developed for the control of robots.

As research and development in this area is still in its infancy, it is difficult to solve problems straight away as there is virtually no literature available. Therefore, the programming of the RL code is more time-consuming than previously thought. The best solution to drive development is to design an RL algorithm from scratch to improve the configurations. This allows the interface of the programs to be changed at will and any problems that arise to be solved more easily.

The multidimensional nature of suspension design presents a challenge that involves both objective (numerical) measures and subjective (feeling-based) considerations. These dimensions defy accurate modelling by conventional neural networks. Therefore, the use of RL, specifically developed for the particular requirements of suspension design, is the logical consequence. This approach recognises the inherent complexity of the problem and attempts to bridge the gap between numerical measurements and subjective assessments, paving the way for a more comprehensive and effective suspension design process.

11. Future Scope

Ensuring the drivability of the RL model's output stands as a critical phase in this project. Validating the model's performance in real-world driving scenarios is essential to affirm its practical applicability and reliability. This validation process serves as a pivotal step in gauging the effectiveness of the RL-based suspension design and its ability to translate into a drivable and responsive system.

The exploration of various RL agent types adds a layer of adaptability to the project. Experimentation with different agent architectures enables the identification of the most suitable one for the task at hand. This adaptive approach acknowledges the diversity in suspension design challenges and aims to tailor the RL model to the specific requirements of optimizing drivability. To enhance the comprehensiveness of the RL model, the incorporation of coefficients for wheel motion under steering as additional RL actions is proposed. This addition recognises the complex relationship between steering dynamics and wheel motion and aims to refine the ability of the RL model to handle the complexity of real driving scenarios and proposes to investigate input parameters such as suspension damping, stabiliser stiffness and others, in addition to manipulating spline coefficients. This broader focus on input parameters recognises the multi-layered nature of suspension design and aims to improve the ability of the RL model to deal with a wide range of optimisation variables. Furthermore, the inclusion of coefficients related to rear wheel dynamics in the RL action space represents a more sophisticated approach to account for the holistic nature of suspensions. Taking into account the combination of front and rear wheel dynamics ensures a more comprehensive optimisation process and thus contributes to an improved and more complex suspension design. A specialised agent aims to exploit the unique challenges and intricacies of suspension design, to better match the RL model to the requirements of the task and maximise its efficiency in achieving optimum handling.

12. References

[1] Graphical model; 2023 [cited 2024 July 1] Available from: URL: https://en.wikipedia.org/w/index.php?title=Graphical_model&oldid=1188076796.

[2] Holländer B. Introduction to Probabilistic Graphical Models - Towards Data Science. Towards Data Science 2020 Feb 23.

[3] Douglas B. What Is Reinforcement Learning?; 2019.

[4] PDF The Hundred-Page Machine Learning Book Andriy Burkov - pdf download free book.

[5] Doya K. Reinforcement learning: Computational theory and biological mechanisms. HFSP J 2007; 1(1): 30 [https://doi.org/10.2976/1.2732246][PMID: 19404458]

[6] Kaelbling LP, Littman ML, Moore AW. Reinforcement Learning: A Survey. jair 1996; 4: 237–85 [https://doi.org/10.1613/jair.301]

[7] Sutton RS, Barto A. Reinforcement learning: An introduction. Second edition. Cambridge, Massachusetts, London, England: The MIT Press 2020.

[8] Reinforcement Learning Using Deep Neural Networks - MATLAB & Simulink - MathWorks Nordic; 2024 [cited 2024 January 7] Available from: URL: https://se.mathworks.com/help/deeplearning/ug/reinforcement-learning-using-deep-neural-networks.html.

[9] Reinforcement Learning Toolbox Documentation - MathWorks Nordic; 2024 [cited 2024 January 7] Available from: URL: https://se.mathworks.com/help/reinforcement-learning/index.html?s_tid=CRUX_lftnav.

[10] VI-CarRealTime | VI-grade; 2024 [cited 2024 January 7] Available from: URL: https://www.vi-grade.com/en/products/vi-carrealtime/.

[11] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. arXiv; 2015.

[12] Mnih V, Kavukcuoglu K, Silver D, *et al.* Human-level control through deep reinforcement learning. Nature 2015; 518(7540): 529–33 [https://doi.org/10.1038/nature14236][PMID: 25719670]

[13] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv; 2018.

[14] Epsilon-Greedy Q-learning. Baeldung on Computer Science 2020 Dec 18.

[15] Boerboom M. internal figures/pictures/diagrams/media of Volvo AB.