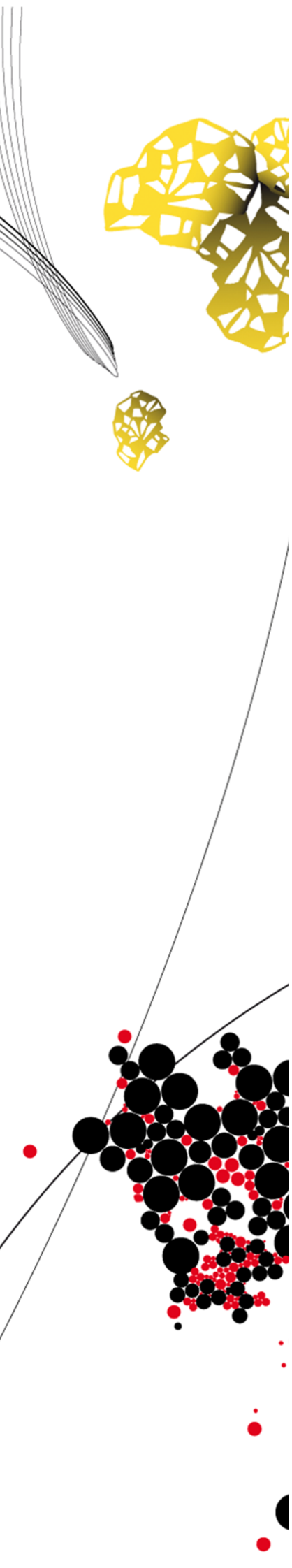


UNIVERSITY OF TWENTE.

Faculty of Engineering Technology

Design and Optimisation of Roller Coaster Elements using Reinforcement Learning

Wouter J. Schuttert

Master Thesis Mechanical Engineering
December 2022

Supervisors:
Prof. Dr. Ir. B. Rosic
Dr. Ir. J. P. Schilder

Chair of Applied Mechanics and Data Analysis
Faculty of Engineering Technology
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

The design of roller coasters in a layout phase can be done traditionally by use of geometric and force vector designs. However, these conventional approaches are often characterised by a lack of flexibility, the need for expert knowledge and difficulty in integrating detailed design. In this research, a novel view on the design problem is taken by the help of which the roller coaster track design is seen as a reinforcement learning (RL) example. The design of a few specific track elements such as a drop, a hill, a banked turn, a 2D looping and a pitched turn is thus mathematically modelled as a Markov Decision Process (MDP), the core of the RL algorithm. The track element is modelled as a parameterized spline, whereas the physics of the roller coasters is integrated in two different ways: either as a simplified point mass or as a more detailed multi-body model. The parameters of the track are then found by optimizing rewards, here to be understood as design goals, that promote for mechanical stability, thrill and passenger safety. The optimal design is then achieved with the help of the proximal policy optimisation RL algorithm.

Although the proposed automatic solution can successfully design some of the most important track elements, the RL solution is shown to struggle with the spatial awareness due to made simplified assumptions. Despite these shortcomings, the automatic RL design is perceived as a promising tool for the roller coaster application. The further focus thus can be on the achievement of the mechanical feasibility and improvement of the MDP model in terms of predefined actions and rewards.

Contents

Chapter 1

Introduction

Over the hundreds of years of roller coaster history, roller coasters have changed much. At the start of the 19th century, railway carts intended to transport coal were discovered to be popular amongst fair visitors. The railway carts were turned into wooden cars travelling down slow inclines at around 30 km/h. Roller coasters further evolved into the fibreglass trains being launched into inversions and spirals at over 100 km/h seen today [1]. The innovation in roller coaster design is sparked by the ever-growing industry of theme parks, where roller coasters are seen as the main attractions [2].

One of the main reasons for the popularity of roller coasters is the experience of thrill [3]. For roller coasters, thrill is described as a sensation of fear, increased heart rate, faster breathing and a spike in adrenaline [4]. These sensations are generated by exposure to variations in the bio-mechanical effects on the rider such as velocities, accelerations (G-Forces) and track elevations and by the special effects around the coasters, which can include light- and sound effects or optical stimulation [5].

In roller coaster design and operation, the safety of passengers and the structure are the most important criterion. For the passenger, exposure to G-forces for prolonged times can cause blurred vision, black-outs, loss of consciousness and even deaths [6, 7]. On the other hand, the structure consisting of the train, track and supports, has to be capable of resisting all applied loads without failure. To prevent failures, an investigation of the stability, strength and rigidity of the roller coaster structure is required. The improper design would lead to mechanical stresses or strains exceeding critical limits, which may cause serious consequences such as structural failure and human fatalities.

Figure 1.1: Corkscrew at Cedar Point. [8]

The process of designing a roller coaster consists of three phases [9]. The initial design phase consists of gathering all requirements necessary for the design specification. Examples are coaster type, area re-

strictions, local and national codes, customer's budget and special customer wishes. This is followed by the first design phase also known as conceptual design. In this phase, several hand sketches are made, showing possible solutions within the constraints. These sketches are shown to the customer and a conceptual design is chosen. At this stage, the track is already partitioned into functional elements, a piece of track with a specific function such as looping or lift-hill. In the second phase, one designs the track profile, by making the exact layout for each individual element. The layout is responsible for the variations in velocities, accelerations and track elevations experienced by the passenger. The definitive track layout is shown to the customer after a 3D render is made. After approval, the detailed design phase is started. In this stage, static and dynamic calculations are performed to determine the size and shape of all components. This step is achieved by the use of MBD, FEM and optimisation algorithms. If the loads are considered too high or expected manufacturing and maintenance costs become excessive, the whole process restarts at the track layout phase and approval from the customer is to be regained.

The biggest challenge in the track layout phase is the incorporation of the engineers' perception of thrill into a safe design of a specific element. The design of a safe element under a thrill criterion is further investigated in this thesis. To adhere to the wishes of the community, it is also important to attempt to integrate aspects of the detailed design phase into the track layout phase [9]. Examples of such aspects are the evaluation of the reach envelope or the dimensions of the roller coaster trains and carts. The reach envelope is the maximum reach of a cart occupant while riding a roller coaster, even the largest occupants should never be able to touch anything other than his/her own cart. When successfully integrated these aspects could prevent time expensive redesign of a roller coaster.

The first computer-aided method in the design of track layouts is geometric design [10]. In geometric design, the first step is to define the trajectory either by drawing, an energy-dependent function or by placing nodes to form a spline. A spline is a piece-wise polynomial which uses reference nodes to define a path in space. The second step is the evaluation of the forces at play. If the resultant forces are satisfactory, the track is within safety standards and fitting to the designer's definition of thrill, the track design is complete. However, if any of these conditions are not fulfilled the track design is to be revisited. Due to the complex nature of the roller coaster tracks, this results in an iterative and expert-intensive process between the two design steps.

Currently, force vector design is seen as the main method in track layout design [11]. This design method is based on the creation of a track geometry that is in accordance with a specified force vector. In this manner, the forces remain within safety standards and the definition of thrill in terms of G-Forces is incorporated more accurately. To reach a certain endpoint or to avoid obstacles in the environment a lot of iterations and expert knowledge are required.

Naturally, roller coaster engineers combine both methods in their work, lessening some of the drawbacks of the previously described methods. However, in both approaches, the integration of detailed design aspects remains difficult due to the complexity of methods like FEM and MBD. Next to this, the experience of the engineer heavily influences the time required for a successful design. This difficulty leads to a high number of iterations between design phases and the costly development of custom roller coasters. As a consequence, companies tend to stick to a design that has worked for them in the past, slowing down innovation. These shortcomings in current design methods signal the need for an improved design method for the layout of a roller coaster.

State of the art

To replace manual iterative design processes such as geometric or force vector design, design optimisation methods are applied. These methods are a tool to accelerate the design cycle and obtain better results. Also referred to as classical optimisation, this method requires a mathematical formulation of the optimisation problem, including design variables that are subject to change, the objective to be minimised, and

the constraints that need to be satisfied [12]. The search techniques for an optimum in classical optimisation are divided into two different algorithms, enumerative and deterministic.

Enumerative methods are the most straightforward search strategy. Enumerative methods compute and evaluate every possible solution to a given problem. However, determining all possible solutions is inefficient or even impossible for large search spaces [13]. The other option is deterministic methods, which are often based on gradients and derivatives. However, in [13, 14] deterministic and enumerative algorithms are described to have difficulty for design optimisation problems with: (1) discrete-valued design variables,(2) a large number of design variables; (3) multiple local minima, maxima, and saddle points; (4) no differentiable objectives and constraints; (5) discontinuities of functions or regions (6) convergence problems of analysis programs. For these reasons, deterministic or enumerative methods are not considered any further.

Today the main technique to optimise design problems are stochastic-based methods [15]. Stochastic-based methods consist of the initialisation of the optimisation process with a set of random candidate solutions for a given problem. The solutions are improved over a predefined number of steps by a chosen algorithm. Many of these algorithms are based on phenomena in nature, commonly called meta-heuristics. Evolutionary algorithms are one of the first meta-heuristic algorithms and exploit ideas such as mutation and a combination of the initial set of solutions to find an optimum. In [16], an evolutionary algorithm is used for the structural optimisation of aerofoils over a large number of decision variables. In [17], the optimisation of wind turbine blades is improved by not only considering the structural strength and stiffness of the blade but also the noise and power generation efficiency of the blade. Further, in [18], the best design of a sports car suspension system using simplified quarter-car models was presented to increase ride comfort. From the plentiful examples, it is concluded that meta-heuristics have a well-defined place in the scientific community. Based on the research reported in [19, 15] it can be stated that a meta-heuristic algorithm can find a good balance between exploration, exploitation, convergence, coverage and low computational cost for many engineering design problems. Here, exploration caters to the ability of the method in searching for new possibilities, whereas exploitation refers to the ability to exploit gathered knowledge about the problem. However, without the use of a large number of iterations or large population size, evolutionary algorithms struggle with complex real-world optimisation problems [20]. Meta-heuristics rely heavily on the definition of the initial solutions. The model that creates the initial solution drives the knowledge and these methods are thus considered model-driven.

Another way to find solutions to a design problem is the reinforcement learning (RL) approach, although barely seen in the field of design optimisation for engineering [15]. RL is a sequential decision-making process which has gained popularity ever since 2016, due to success in the board game Go [21] and more complicated games such as Starcraft [22] or the folding of protein structures [23]. Currently, RL is seen in a variety of fields and as a valid alternative for solving optimisation problems [24, 20]. Research has found that reinforcement learning can find non-intuitive solutions by making use of a combination of exploration and exploitation [25]. In mechanical engineering, RL is often encountered in the control of dynamical systems, due to the ability of the algorithm to deal with dissipation mechanics such as hysteresis, backlash and friction in a model-free setting [26, 27]. This method starts by collecting data about the environment, i.e. the state of the system, that has to be controlled. The optimal control is then derived by having the data-driven model of the corresponding environment. In fluid mechanics, RL is used due to its ability in dealing with a combination of non-linearity and high dimensionality in flow control and shape optimisation [28]. With regards to RL in structural design engineering, no mention is made in a recent state-of-the-art review [29], but novel approaches can be found. In [30], RL is used to optimise the design of nuclear cooling rods, a challenging problem where fuel designers' expert judgement has commonly prevailed over the use of stochastic optimisation. RL demonstrated superior performance to meta-heuristic methods by exploring the search space better and increasing the algorithm speed and efficiency. However, the results created by RL are often less insightful and it is more difficult to find the

reason for a specific design choice. This difficulty should be kept in mind when creating an RL model. The gap of knowledge with regard to RL in design optimisation combined with the successes in other fields makes RL a good research direction for design optimisation.

To apply any method of optimisation in the layout phase, one starts with building a model of the required mathematical and physical laws required for the problem, further followed by its analysis. For a roller coaster, with reasonable accuracy, point dynamics are used to describe forces acting on the heart-line of the passenger during a roller coaster ride in the track layout phase [10]. With regards to the track, a cubic spline function such as a basis or Bézier spline can be used for a continuous model [31]. In [32], the heart-line represented by a cubic B-spline is converted to a track layout using a procedural method which enables further analyses for the detailed design phase. A similar method is also applied in the roller coaster design program NoLimits 2 [33]. This program is used by roller coaster manufacturers for the layout, design, integration, visualisation and marketing. To optimise the design of a roller coaster element in the layout phase, a successful attempt is made in [34] by making use of the principle of conservation of energy, from which the time needed to travel the curve is solved and minimised. However, criteria other than minimisation of the travel time between point A and point B are not implemented. Instead of focusing on the layout phase, most current research considers the detailed design phase. In [35], a roller coaster design environment is developed for the proper evaluation of dynamic and loading conditions of roller coaster vehicles. In [36], multi-body modelling is used to not only obtain forces experienced by the vehicle and track but also the forces acting on the passenger as a result of the restraints. To enhance the lifespan of roller coasters and reduce safety factors a transient analysis method based on FEM and MBD models was developed [37]. These papers follow a similar method where MBD is used to obtain the load on the track and vehicle, which are then used in a FEM to analyse the integrity of the considered structures. These analyses are required for the full design of a roller coaster, but MBD and FEM require in-depth domain knowledge and long computation times, making the integration into the track layout phase more difficult [9]. The computational inefficiency is due to the explicit numerical integration schemes that are often used for the time integration of the system behaviour. To mitigate the computational times of numerical integration schemes, which are too expensive to implement in an optimisation problem, a surrogate model in a form of a neural network is used in several applications [38, 39, 40]. Neural networks are known as universal function approximators [41] and thus can be used to describe any physics-based system. In particular, for vehicle track systems, a deep learning model has been used to successfully predict low-frequency responses [42]. Based on the research a roller coaster can simply be described by a point mass and a spline in the layout phase but advances in computational efficiency for MBD and FEM methods implicate that a surrogate model can be developed to enhance the quality of the design in the layout phase.

1.1 Research Motivation

The current methods to find a solution in the layout phase such as geometric design and force vector design, are manual optimisation methods. These methods are characterised by a lack of flexibility, the need for expert knowledge and difficulty in integrating detailed design. Based on the current state of the art, two methods are considered for further improvement. Well-established meta-heuristics or reinforcement learning. The possibilities of meta-heuristics are well known and there are many applications which translate well into roller coaster design signalling that much of the required knowledge is already there. Furthermore, meta-heuristics still struggle with complex real-world optimisation problems and require a complete model of the solution possibilities as a model-driven approach. On the other hand, RL does not have an established place in the optimisation of engineering design but has shown great potential in other fields of mechanical engineering. Furthermore, a special subclass of RL methods also known as model-free RL is a data-driven approach that can incorporate unforeseen effects and it does not require a solution to be provided beforehand. In this thesis, the model-free approach is considered due to its

immediate adaptability in real-case industrial applications. Lastly, in an RL framework, a neural network as a surrogate model for an MBD model is more readily applied because of the knowledge overlap between RL and neural networks as surrogate models. This translates to easier integration of detailed design aspects into the layout phase to enhance the quality of the design of a roller coaster element. Due to the gap of knowledge in RL, the model adaptability of model-free RL and the possible advantages of a complete machine learning framework RL is chosen as a method. RL can create a more efficient roller coaster layout design process and is further researched in this thesis.

1.2 Research Objectives and outline

The goal of this research is to develop a reinforcement learning model which can optimise a functional element of a roller coaster. The model design process is started by investigating the translation of the engineering problem into a reinforcement learning problem, the main topic of this thesis. Then a solution algorithm is applied to find possible designs. To illustrate the capabilities and correctness of the RL model a layout for several common 2D elements is generated using a point mass to represent the train. This model is expanded to solve 3D problems. For both 2D and 3D, multiple cases are considered to attempt and create a more insightful application of the RL approach. Lastly, a 2D RL model is created using MBD to represent the train obtaining a more detailed layout design process. To represent the MBd a surrogate model is used. The secondary topic of this thesis is the development of this surrogate model for the cart position on the track. This surrogate model is developed to keep the computations times low. The surrogate model is kept simple by only considering the 2D kinematics. The surrogate models consists of a neural network as function approximator. Important to note is that the models used in this thesis do not constitute a real roller coaster. All data is approximate and used for demonstration purposes only.

In chapter 2 the problem description for a roller coaster element is given. In the chapter, design goals and all relevant details about the physical laws, regulations and assumptions of the problem are described. The chapter starts with the design goals followed by a description of the physics, i.e. spline mathematics, the point mass and multi-body modelling. Furthermore an example of the design process for a drop, a common roller coaster element, is given to illustrate the choices made in the design process. This is followed by a description for optimisation. Lastly, the use of a neural network as a surrogate model to represent the train for a computationally expensive MBD is presented and the required theory about neural networks is given.
The reinforcement learning methodology is explored in the first section of chapter 3 and applied to the creation of a roller coaster element. The chapter starts with an introduction to reinforcement learning followed by the formulation of the Markov decision process. Then the solution method and algorithm are discussed.
In chapter 4, the implementation, training and optimisation of the proposed methodology are discussed. The first section explains in detail the implementation process and iterations of the models that were made. In the second section, the parameters and creation of the surrogate model are discussed.
A case-study approach is taken in Chapter 5 to find a solution for several different roller coaster elements. The chapter starts with 2D elements, then two 3D elements and lastly a 2D MBD integrated element. Any problem-specific changes to the implementation as presented in chapter 4 are explained and reviewed per element.
Chapter 6 consists of a conclusion that summarises the work done in this thesis, followed by a discussion on the problem, method and results. Finally, recommendations are made for further research.

text

Chapter 2

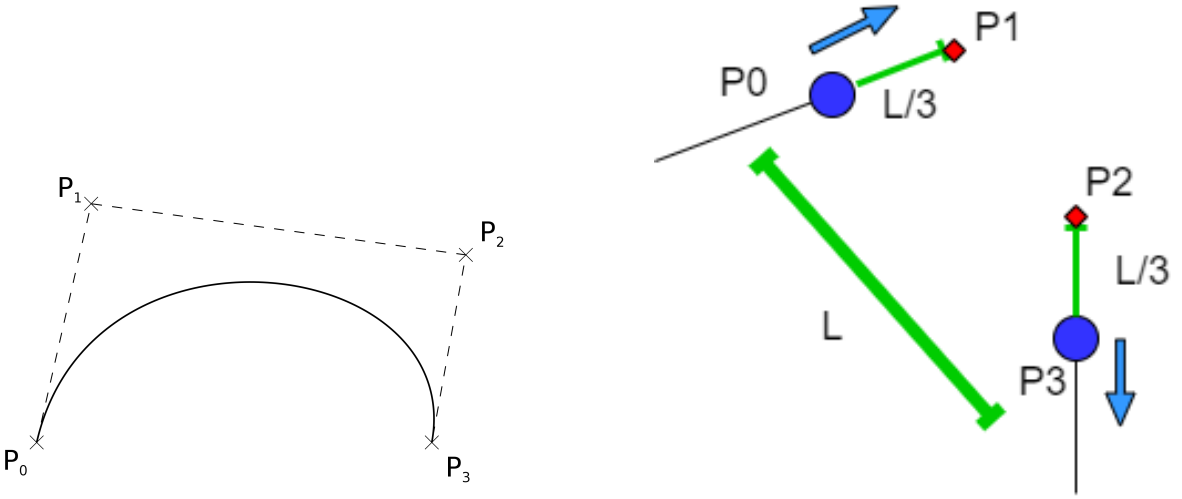
Problem Definition

In this chapter, the design problem of a roller coaster element is explained. The design goals, assumptions and considerations are presented in the first section. In the second and third sections the approximation for the track of a roller coaster is formulated with the help of Bézier splines and the local frame of reference. Then the physics of a roller coaster and two possible formulations for a roller coaster train, either as point mass or as a multi-body structure are given. In the fourth section, a roller coaster drop element is designed and global and local optimisation are discussed. To be able to integrate the multi-body structure into an optimisation problem a surrogate model is introduced in the fifth section followed by an explanation of neural networks.

2.1 Design goals

The first step in the design of a roller coaster is determining the design goals. Three design goals are considered in this thesis. The first goal is to design an element that fits within the assigned space and begins and ends at the specified points in space. The second goal is to design a safe element. A safe segment is when the forces experienced by the passenger are within safety limits. These limits are defined by accelerations and jerks. Accelerations are changes in velocity expressed in G. Acceleration is further divided into vertical acceleration, a change in up-to-down motion and lateral acceleration, a change in a side-to-side motion. Jerk is a change in acceleration expressed in G/s. The last goal is to design a thrilling element, which is only important if the other two design goals are achieved. Within these design goals a lot of variables can be defined but to lessen the scope of this thesis several considerations and assumptions are made.

The first assumption is that the track can take on any curvature. With regards to a specific element description, the start point, end point, properties of the cart, initial velocity and spatial boundaries are considered given. The safety standards are the limits for the accelerations and jerk, set at a maximum of $5G$ vertical acceleration, a minimum of $-1G$ vertical accelerations, an absolute maximum of $1.5G$ lateral acceleration and an absolute maximum of $15G/s$ Jerk [43]. Also, it is assumed that the thrill experienced in an element can be broken down into a single criterion. This criterion is chosen based on roller coasters that score high in the golden ticket awards, a ranking of roller coasters, and literature on enjoyment [44, 4]. Further, many practical considerations of design are out of the scope of this thesis, the main ones are budget costs, support structure feasibility, environmental conditions and material limitations. Lastly, elements that have an external influence on the movement of the train are not considered, for example, lift hills or brake runs. To evaluate the three design goals with these considerations and assumptions the mathematical and physical laws describing the roller coaster element have to be defined.



(a) Cubic Bézier curve with control points. [45]

(b) 1/3 rule in a cubic Bézier curve.

Figure 2.1: Example of Bézier Curve and the 1/3 rule.

2.2 Spline and reference frame

The track of the roller coaster is described by a spline. Splines are able to describe a path through space with a low amount of variables making them a lot more convenient than specifying every centimetre of the track. In literature, cubic Bézier splines are deemed suitable for use in layout design. The reasoning for this is that they are twice differential which is required to obtain the direction and magnitude of the radius of curvature. A cubic Bézier spline is a parametric curve generated by a set of four control points. An example of the control points and its Bézier curve is seen in Fig. (2.1a). As shown, the dotted lines are the connections between control points and the curved line is the spline, which is described by:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3, 0 \leq t \leq 1 \tag{2.1}$$

In which, $P0 - P3$ represents the four control points and t represents the parametric Bézier variable that ranges from zero to one. P_0 and P_3 are the start and end points also known as join points. P_1 and P_2 are known as the intermediate points. With only the joint points, intermediate points can be computed by applying the 1/3 rule. This does require a known direction vector in the join points. The application of the 1/3 rule is shown in Fig. (2.1b). As depicted, the green lines represent distances between points, the blue point, with corresponding blue arrows are the join points and the red diamonds are intermediate control points. P_1 is found by adding 1/3 of the distance between P_0 and P_3 to P_0 in the direction of local forward facing vector at point P_0. By subtracting this distance from P_3 in the local direction of the forward-facing vector, P_2 is created. Mathematically P_1 and P_2 are defined by:

$$\begin{aligned} P_1^x &= P_0^x + \frac{1}{3}L_{seg}\cos P_1^p \\ P_1^z &= P_0^z + \frac{1}{3}L_{seg}\sin P_1^p \\ P_2^x &= P_3^x - \frac{1}{3}L_{seg}\cos P_3^p \\ P_2^z &= P_3^z - \frac{1}{3}L_{seg}\sin P_3^p \end{aligned} \tag{2.2}$$

where the superscript X or Z is the respective coordinate at the join point, L is the euclidean distance between the join points and the superscript p describes the local pitch at the join point. To compute the forward-facing vector, or tangent vector, at any point of the spline one requires the first derivative:

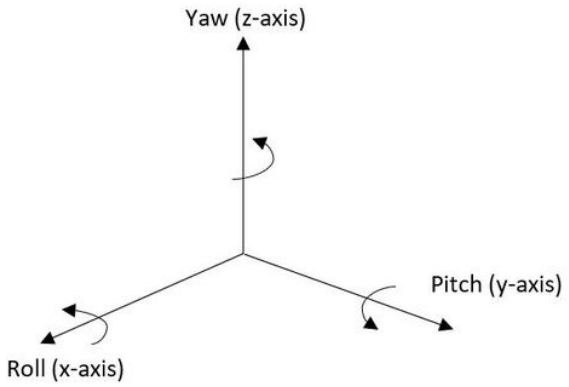


Figure 2.2: Coordinate axis with pitch, yaw and roll. [46]

$$\mathbf{B}'(t) = 3(1-t)^2(\mathbf{P}_1 - \mathbf{P}_0) + 6(1-t)t(\mathbf{P}_2 - \mathbf{P}_1) + 3t^2(\mathbf{P}_3 - \mathbf{P}_2). \tag{2.3}$$

The tangent vector $\mathbf{e}_1(t)$ and normal vector $\mathbf{e}_2(t)$ for the Bézier curve are then obtained using the following equation:

$$\mathbf{e}_1(t) = \frac{\mathbf{B}'(t)}{\|\mathbf{B}'(t)\|}, \quad \mathbf{e}_2(t) = \frac{\mathbf{e}_1'(t)}{\|\mathbf{e}_1'(t)\|} \tag{2.4}$$

The radius of curvature r is required for determining accelerations and is calculated using the double differentiable property of Bézier curves, i.e.

$$r = \frac{\left(x'^2 + y'^2 + z'^2\right)^{\frac{3}{2}}}{\sqrt{\left(z''y' - y''z'\right)^2 + \left(x''z' - z''x'\right)^2 + \left(y''x' - x''y'\right)^2}} \tag{2.5}$$

where the direction is defined by $\mathbf{e}_2(t)$.To know where the local positions in the spline are with respect to the global frame, the local frame of reference is needed. This is needed to integrate the effects of gravity, for instance. For the local frame of reference, the transformation (rotation) matrix is required. The coordinate frame used throughout the thesis is shown in Fig. (2.2). The respective rotation of pitch, yaw and roll is given by the 2D rotation matrix. Given the pitch β, yaw α and roll γ angles, one may define:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.6}$$

And for the complete 3D matrix:

$$R = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}. \tag{2.7}$$

In Eq. (2.7), the first column is the longitudinal direction vector. The second column is the lateral direction vector. The third column is the vertical direction vector. The spline together with the local direction vectors is able to define a position on a roller track at any point in space. To describe the roller coaster train and the effects on the passenger such as G-Forces, physical laws are required.

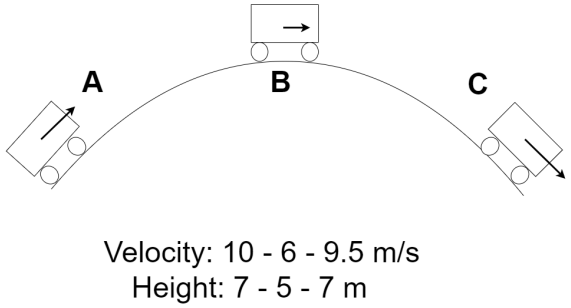


Figure 2.3: Cart rolling over a hill under effects of gravity and friction.

2.3 Roller coaster Physics

A classical roller coaster is powered by a straightforward concept: the conversion of potential energy to kinetic energy and back. Potential energy is the energy held by an object because of its relative height to the ground given by:

$$E_P = mgh. \tag{2.8}$$

Here, m is the mass of the body, g is the gravitational constant and h is the height above a reference level. The kinetic energy which is the energy because of the velocity of an object is given by:

$$E_k = E_t + E_r \quad E_k = \frac{1}{2}mv^2 \quad E_r = \frac{1}{2}\bar{\omega}^t \bar{\bar{I}} \bar{\omega} \tag{2.9}$$

In which E_t is the translational kinetic energy and v is the velocity, E_r is the rotational kinetic energy, $\bar{\bar{I}}$ is the bodies moment of inertia and $\bar{\omega}$ is the angular velocity. Here, the bar and double bar denote a 2D and 3D tensor respectively.

Although potential energy is directly changed into kinematic energy the roller coaster loses some energy to the environment by travelling on a path due to friction and air drag defined by:

$$E_{Loss} = (F_r + F_D)ds \quad F_r = C_{rr}N \quad F_D = \frac{1}{2}\rho v^2 C_D A \tag{2.10}$$

where ds is a segment of track, C_{rr} is the rolling resistance coefficient, N is the normal force on the body, ρ is the air density, v is the velocity of the body, C_D is the drag coefficient and A is the surface area of the frontal plane of the body.

In Fig. (2.3) a cart is drawn going up a hill to travel from point A to B and eventually to point C. At each point, the velocity and relative height are given. The energy at point A of the cart is described by the kinetic E_k^A and potential energy E_p^A. At the next point B along the track, the new kinetic energy E_k^B is found by calculating the new potential energy E_p^B and subtracting the energy losses between point A and B E_{Loss}^{AB} using the energy equation:

$$E_P^A + E_k^A = E_P^B + E_K^B - E_{Loss}^{AB} \tag{2.11}$$

From this equation, one determines the kinetic energy E_K^B as the change in height is defined by the path. With the kinetic energy, the velocity of the body is determined. In the same way, the velocity at point C is determined. Note that the velocity decreases in point C although the height is the same as in point A due to the energy losses from friction and drag. To further describe what is happening to the roller coaster train, the considered body is described by either a point mass or MBD model.

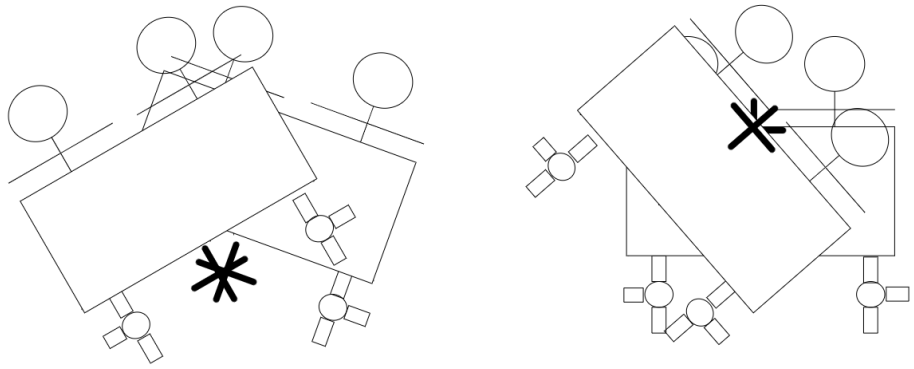


Figure 2.4: Rotation point for the spline at the centre or rails (left) and for the spline at the heartline (right).

2.3.1 Points mass cart model

The most simplified form of a roller coaster model is a point mass representing a cart following a spline. In this formulation, the spline is the heartline of the passenger as depicted in the right image of Fig. (2.4). The energy equation defines the velocity of the points mass, but accelerations and its derivative, i.e. jerk, are dependent on track geometry and velocity and read:

$$a_t = \frac{dv}{dt} \quad a_c = \frac{v^2}{r} \quad j = \frac{da_c}{dt}. \tag{2.12}$$

Here, a_l is the longitudinal acceleration, a_c is the centripetal acceleration, j is the centripetal jerk, v is the velocity, t is the time and r is the radius of curvature. To determine the orientation of the passenger a vertical $\bar{V_P}$ and lateral $\bar{L_P}$ vector are attached to the point mass defined by the direction of the track by Eq. (2.7). With the orientation of the passenger, one can divide the centripetal force which is in the direction of the normal vector $\bar{e}_2(t)$ up into a vertical and lateral acceleration experienced by the passenger. This is achieved by taking the dot product between the normal vector of the track and gravity and the vertical and lateral vector of the passenger such that for the vertical acceleration experienced by the passenger A_v:

$$A_v = a_c(\bar{V_P} \cdot \bar{e}_2(t)) + (\bar{G} \cdot \bar{V_P}) \tag{2.13}$$

Here, **G** is the vector for gravity which points in the negative Z direction and is of the magnitude of $-9.81m/s^2$ or 1G. And for lateral acceleration experienced by the passenger:

$$A_l = a_c(\bar{L_P} \cdot \bar{e}_2(t)) + (\bar{G} \cdot \bar{L_P}) \tag{2.14}$$

These accelerations are used to evaluate the second design goal. Note that a point mass has no rotational kinetic energy and does not take the shape of the roller coaster track or train into account. This provides very limited modelling that heavily approximates reality. However, such a model is often applied in the layout phase. For a more accurate description of the roller coaster behaviour in the detailed design, stage one is to substitute the point mass model with a physics-based model that accounts for a more detailed dynamics description. A multi-body model is one example of such a modelling procedure.

2.3.2 Multi-body cart model

Multi-body dynamics are concerned with the simulation of complex systems that are subject to large rotations. The idea is to describe the dynamics of a system by dividing it into several bodies that are mutually connected. This can be achieved in several different ways [47]. In this thesis, the floating

frame of reference formulation (FFRF) is used. The FFRF assigns a frame of reference to every body and connects any arbitrarily shaped body to another body or the fixed world. In the context of the roller coaster, the carts are the floating frames connected to each other and the track which represents the fixed world. The relations between the carts and track are described by the constraint equations in the form:

$$\mathcal{C}(\mathbf{q}) = \begin{bmatrix} \mathcal{C}_1(\mathbf{q}) \\ \vdots \\ \mathcal{C}_{N_c}(\mathbf{q}) \end{bmatrix} = \mathbf{0} \tag{2.15}$$

Here, C is the constraint equation and q is the generalised coordinate. A generalised coordinate can take the form of a spatial coordinate or angle with respect to a certain axis. As per the research objectives, only a 2D model is considered. A body in 2D is described by three generalised coordinates corresponding with three constraint equations. The fact that the bodies remain connected to the spline, introduces kinematics constraints. In this MBD formulation, the spline represents the centre of rails instead of the heartline as depicted in the left image in Fig. (2.4). The constraint equations are often highly non-linear due to geometrical nonlinearities, this means that a numerical estimator is used to generate a solution for Eq. (2.15).

With the derivations presented in the last two sections, it is possible to fully describe roller coaster dynamics along any given path using a point mass or multi-body model. Together with the design goals, a roller coaster element can be designed.

2.4 Designing a roller coaster element

To capture the essence of the design process an example problem is solved. The example problem considers the design of a 2D drop element. The start point is at X = 0m, Z = 20m and the tangent vector of the track is in the positive X direction, the endpoint is at X = 20m, Z = 0m and the tangent vector of the track is in the positive X direction. The track is not allowed to go below Z = 0 and the initial velocity of the train is 3 m/s. Modelling of the track is done in NoLimits 2. The program calculates information about the velocity, curvature and accelerations using a point mass. Nolimits 2 uses a Bézier curve for the creation of the spline. The problem together with a straightforward initial solution, a straight line from start to end, is shown in Fig. (2.5). The yellow line is the track. The almost instantaneous change of direction means a very low r, see Eq. (2.5). Due to the relation of the radius of curvature and acceleration in Eq. (2.12) G-Forces of up to 40 G are generated in this solution. Improvements are needed to create a safe element.

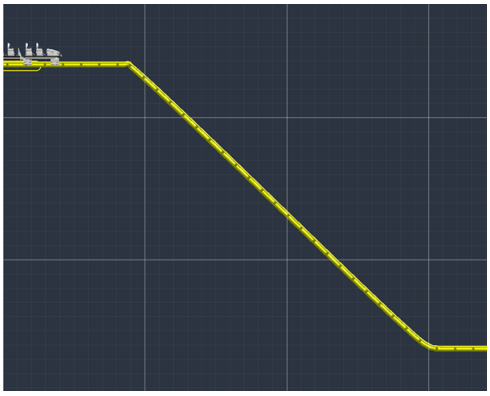
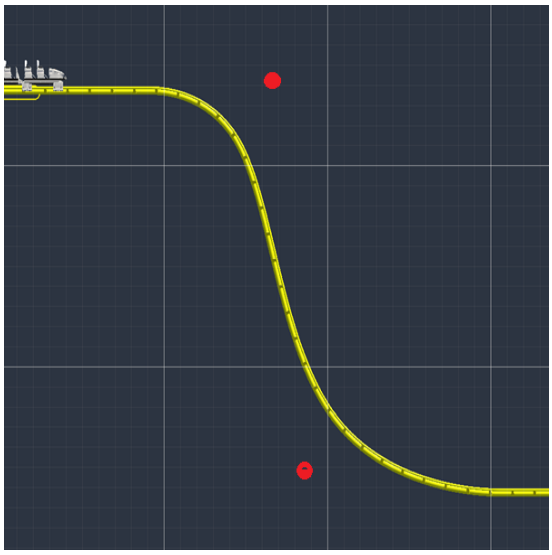
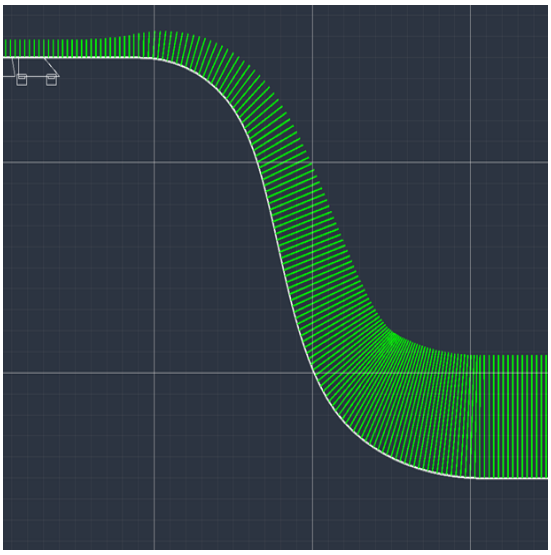


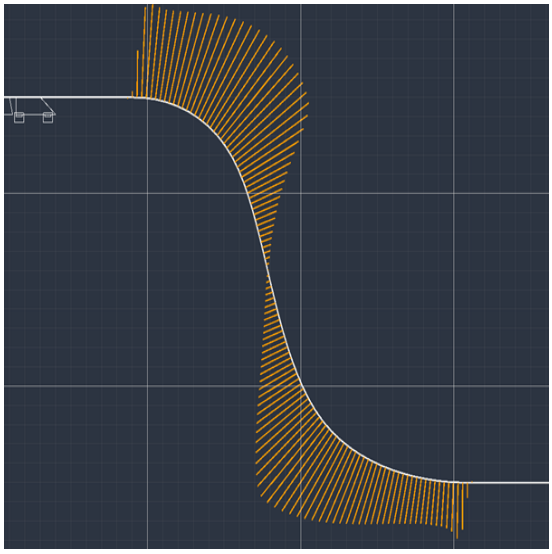
Figure 2.5: The problem with a simple solution, a straight piece of yellow track from start to end.



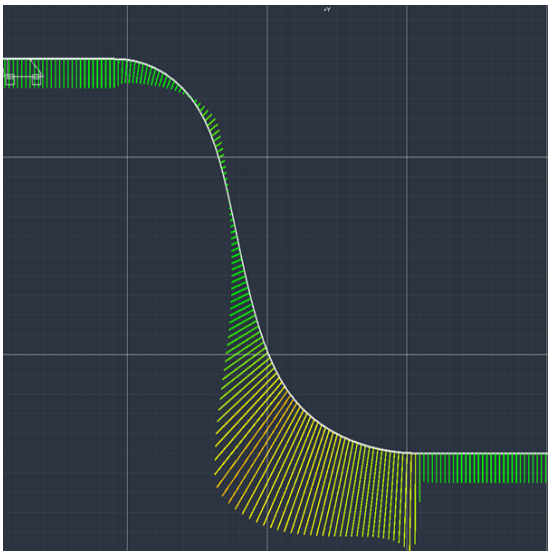
(a) Solution for the example problem, the red points are spline control points.



(b) The heart-line velocity for the roller coaster cart. Going from 10.8 km/h to 69 km/h.



(c) The heart-line curvature for the roller coaster cart.



(d) The heart-line acceleration for the roller coaster cart. Starting at 1 G, gravity, going up to -0.43 G and increasing to 4.91.

Figure 2.6: The analysis of a roller coaster element in NoLimits 2.

To improve the spline three options are given in NoLimits 2, placing a new node, moving an existing node or fixing the relationship between nodes, such as distance or relative height. To improve the design, two extra nodes are inserted and moved until G-forces met safety standards. The new nodes together with the start and end points create a cubic Bézier curve. The resulting track is seen in Fig. (2.6a), the red points are the newly placed nodes and again the yellow line is the track. The velocity, curvature and vertical acceleration for this solution are shown in Fig. (2.6b),Fig. (2.6c) and Fig. (2.6c) respectively. The lines coming from form the track in the figures represent the relative magnitude and for radius and vertical acceleration the direction of the associated value. Important to note that curvature is shown in Fig. (2.6c), not radius of curvature. Radius of curvature is one divided by curvature. This design provides an appropriate solution to the problem but it is manually generated and therefore is most probably one of the possible solutions, and not necessarily optimal.

2.4.1 Optimisation of design

To obtain an optimal solution a mathematical description of the possible optimisation is required. The optimisation of the element or curve can be done in three ways. The first is to increase the order of the curve or polynomial while keeping a single curve. This is the p-approach, a global approach. The second is to discretise the curve into smaller elements and optimise the local curves. This is the h-approach. Both options can also be utilised at the same time to optimise the curve, the h-p approach. The h-p approach presents too much freedom in the choices and is not further considered in this thesis. To make a choice between the other two one has to know the main difference, as both allow more accuracy in describing the desired track geometry. The p-approach makes a change in the governing equation for the curve which can be hard to deal with. The equation changes since a cubic Bézier curve would have to be changed to a quartic one. On the other hand, the h-approach discretises the curve, whereby a lot of extra variables are introduced. These variables arise from the local curves that have to be defined. The disadvantage of the large increase in variables in the h-approach can be overcome by employing a strictly sequential decision strategy. In this context, a sequential decision strategy describes the creation of an element by one local element at a time until the full element is described. The considered h-approach with sequential decisions allows an optimisation algorithm to focus on the creation of one local element at a time and on one set of the same variables. This is why, in this thesis, y a local sequential decision-making strategy is applied to optimise a roller coaster element. Before optimisation can be applied the discretisation of the track is explained.

The roller coaster element is discretised by dividing the element into multiple smaller elements, referred to as segments from this point onwards. Each segment with its own cubic Bézier curve. In this way, the cubic Bézier Eq. (2.1) remains valid. To control the extent of discretisation the variable N_{seg} is introduced, which is the number of segments in an element. Continuity between segments is required since each segment is described by a different equation. To ensure continuity, intermediate control points that originate from the same join point are locked. Locking means that the tangent vector remains continuous. This phenomenon is depicted in Fig. (2.7a). Here, two segments of an element are drawn, numbered one and two, each with its own Bézier curve. The join points of the segments are drawn as blue circles denoted by A, B and C and the intermediate control points are red diamonds labelled a_2, b_1, b_2 and c_1. For segment one this means that the P_0 , P_1 , P_2 and P_3 of Eq. (2.1) are represented by A, a_2, b_1 and B respectively. For segment two this is B, b_2, c_1 and C. The locking of intermediate control points in the figure is the identical tangent vector of b_1 and b_2 at B as is done in figure Fig. (2.7a). This is formally given by:

$$B = b_1 + (b_1 + b_2)\beta_1 \tag{2.16}$$

Here, β_1 is a freely chosen variable. To obtain the location of an intermediate control point, the 1/3 rule is used. The 1/3 rule automatically ensures locking. Consequently, only the position and tangent at the join point are required to describe the next segment. To describe a join point with a specified tangent the term node is chosen in this thesis. A node, in a 2D setting on an X-Z plane, is used to describe a point with an X, Z and Pitch coordinate. In an X-Y plane, a node has an X, Y, yaw and roll. In 3D a node has an X, Y, Z, pitch, yaw and roll.

The proposed concept of discretisation is depicted in Fig. (2.7b). Here a 2D drop element is discretised with N_{seg} is four. In the figure, nodes are the blue dots accompanied by the letters A to E, five in total. Each node is also accompanied by a red arrow showing the local tangent. In the figure, each segment is presented with a number from one to four. This discretisation increases the number of nodes from two to five, where nine extra variables are needed to specify the geometry but extra accuracy is obtained. The number of variables to consider at any one time is reduced by introducing the concept of sequential

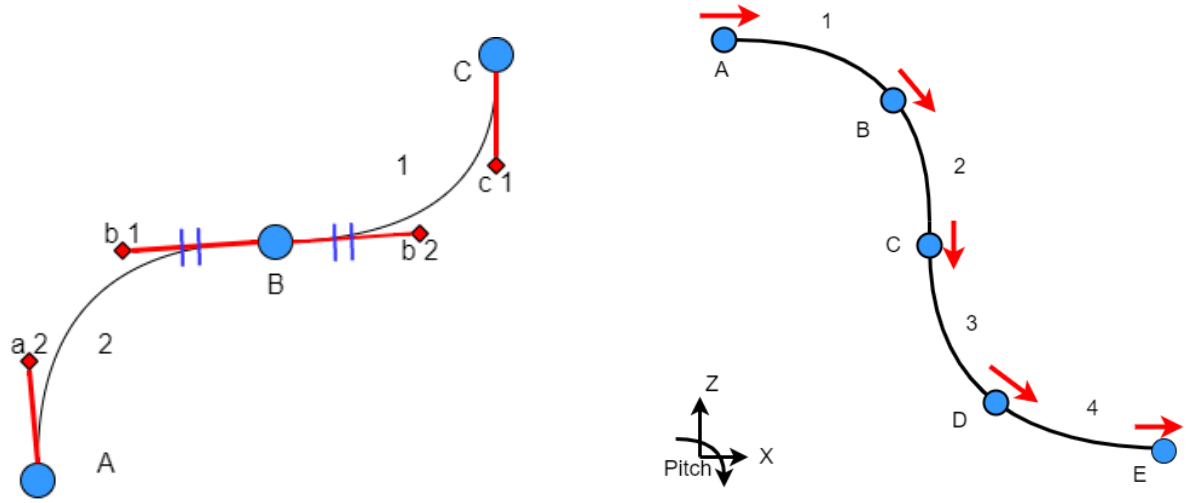
decisions.

Sequential decision-making reduces the variable to be chosen at any point to just the next node. Such that, starting from node A in Fig. (2.7b), only node B has to be specified. This is done by specifying the X, Z and pitch at node B. The coordinates Node B are most efficiently chosen by specifying the change in pitch ΔP between A and B, assuming a constant length between segments. This defines the X and Z coordinates in one variable. To further control the tangent at the node an adjustment of pitch δP is made. Thus, by specifying ΔP and δP, node B can be chosen as per Fig. (2.7b). Then again point C is chosen by specifying ΔP and δP and so on. For yaw, ΔW and δW are used to denote the change and adjustment of yaw. ΔR is introduced to specify a change in roll. The effect on the placement of node B from A is found with help of the 3D rotation matrix:

$$\begin{aligned} X_B &= X_A + L_{seg}\cos(P_t + \Delta^P)\cos(W_t + \Delta^W) \\ Y_B &= Y_A + L_{seg}\cos(P_t + \Delta^P)\sin(W_t + \Delta^W) \\ Z_B &= Z_A - L_{seg}\sin(P_t + \Delta^P) \\ P_B &= P_A + \Delta^P + \delta^P \\ W_B &= W_A + \Delta^W + \delta^W \\ R_B &= R_A + \Delta^R \end{aligned} \tag{2.17}$$

Here, L_{seg} is used to denote the distance between segments. P, W and R denote pitch, yaw and roll respectively. An element can now be created using a sequence of simple choices. However, this process can be cumbersome due to the number of choices available to the extent that even finding a possible solution is complicated. To make the design of an element automatic, and hence optimise the solution, one requires an automatic method to find the optimal design. The creation of a method which can find an optimum in the design of a roller coaster element is the main challenge tackled in this thesis.

Regardless of the optimisation method used one generally requires a huge amount of iterations of the layout to find the optimal design. This makes the MBD roller coaster train more difficult to integrate since it uses a numerical integrator which increases the time needed for each iteration drastically. A surrogate model representing the MBD of the roller coaster train is required for such an integration.



(a) Control point locking in a Bézier curve.

(b) Discretisation of a roller coaster element.

Figure 2.7

2.5 Surrogate multi-body dynamics model

The workflow to create a surrogate model for the roller coaster train is given in Fig. (2.8). First, possible elements are created using our optimisation method with point mass physics. These track function as the inputs for the multi-body model and the required dataset. Then a multi-body model has to be constructed such that the desired calculations can be executed in the proposed local sequential approach. In the next step, the multi-body model and rollercoaster track are used to generate an input-output data set from which a surrogate model can learn. This data set consists of inputs which are the track and initial position of the roller coaster train and outputs which are the new position of the roller coaster train. Then a neural network is trained using a supervised learning algorithm. Supervised learning relies on the simulation data created in the third step to learn a mapping between the inputs and outputs. If the performance of the surrogate model is adequate a multi-body roller coaster train can be used to solve the optimisation problem. To start, the multibody model is created.

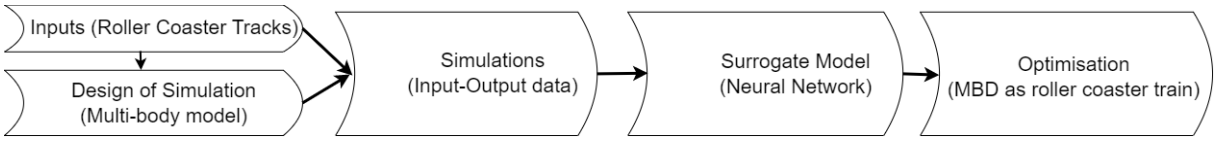


Figure 2.8: Workflow to create a surrogate model.

2.5.1 Multi-body roller coaster train

The multi-body model should be able to function in the proposed local sequential approach. Thus, it should be able to work in the local setting with a constantly changing track. The goal is to find a new position of the train for the current position of the train. The initial position of the roller coaster train is depicted in Fig. (2.9). The numbers describe each body, five in total, the lengths are used to define the constraint equations, the blue line is the track, the blue points are the nodes, the red circles are the wheels and the black lines form the cart. The train is placed on a straight track consisting of eight segments. As is done in the optimisation approach, one new segment is added onto the initial track. The multi-body model determines the new position at every desired point in the new segment. To introduce a second new segment, the most left segment is deleted in Fig. (2.9) and a new segment is placed in front. In this way, the amount of segments defined in the multi-body model remains constant. To keep the model simple, all elements are considered 2D rigid bodies. These bodies are connected by sets of kinematic joints. The roller coaster vehicle consists of a train with three cars interconnected by two linking bars, as represented in Fig. (2.9). The origin of the global coordinate system (x, z) is located at the first wheel axis. The initial height of the train is depended on the start position of the element.

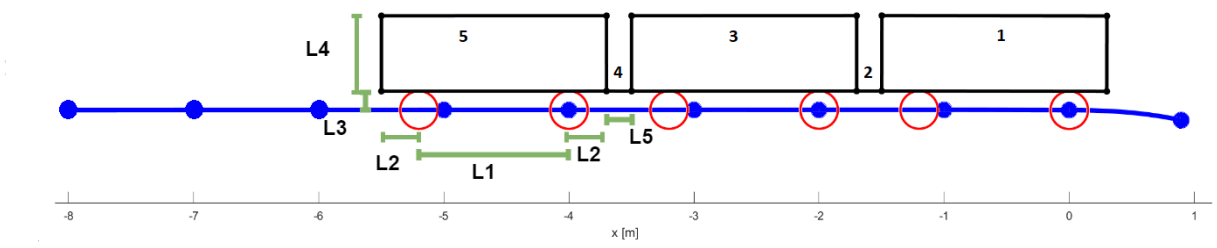


Figure 2.9: Drawing of the considered roller coaster train.

The multi-body model works by describing a roller coaster cart's position in terms of the parametric Bézier variable t. t represents the position of the first and second wheels of each cart in each segment. t

is put into the equation for the corresponding Bézier equation to generate an X and Z coordinate on the curve in which the wheel is present. The angle of the cart is the third generalised coordinate of a cart. Body one is given by q_1, q_2 and q_3 where q_1 is the right wheel, q_2 is the angle of the body and q_3 is the left wheel. This is the same for body three, given by q_7, q_8 and q_9 and body five given by q_{13}, q_{14} and q_{15}. To describe a linking bar, the X and Z position of the centre of mass and the angle of the bar is used. For body two this means q_4, q_5 and q_6 where q_4 is the X coordinate, q_5 is the Z coordinate and q_6 is the angle of the body. This is the same for body four given by q_{10}, q_{11} and q_{12}. The constraint equations $C(q)$ are formulated as in Eq. (2.15). The driving constraint equation determines the progress of the roller coaster cart in the new segment and is given by the position of the first wheel:

$$q_1 - t = 0 \tag{2.18}$$

where s is the constant for the corresponding Bézier curve. Examples of two constraints are:

$$\begin{aligned} B_x^{q_3} - B_x^{q_1} - L_1 \cos(q_2) = 0 \\ B_z^{q_3} - B_z^{q_1} - L_1 \sin(q_2) = 0 \end{aligned} \tag{2.19}$$

where the length L_1 should remain the same in both situations. In the first line, the distance between the X coordinate of wheel one and wheel two is compared. In the second line, the Z coordinates are compared. $B_x^{q_3}$ corresponds to the X coordinate of the Bézier curve for wheel two, as wheel two is described by q_3. $B_z^{q_3}$ corresponds to the Z coordinate of the Bézier curve for wheel two. $B_x^{q_1}$ corresponds to the X coordinate of the Bézier curve for wheel one, as wheel one is described by q_1. $B_z^{q_1}$ corresponds to the Z coordinate of the Bézier curve for wheel one. With Eq. (2.18) and Eq. (2.19) it is possible to solve for q_1, q_2 and q_3 since t is our input and there are three equations with three unknowns. However, to know which Bézier curve is used to determine the X and Z coordinates an extra parameter is added to the model. B_{wheel} is introduced to index each wheel's Bézier curve. The initial positions of wheels one to six are initially in the first, third, third, fifth, fifth and seventh segment respectively. Then at every step, it is checked whether the parametric Bézier constant exceeds one because the parametric Bézier constant is only valid between zero and one. If so, the Bézier function associated with the wheel one is moved to its neighbouring segment on the track. The rest of the constraint equations are found in appendix A.1. To solve the set of constraint equations, the numerical estimation method Newton–Raphson is used where the initial guess for the solution is based on the derivative of the considered set of equations:

$$q_1 = q_0 - \frac{C^q(q_0)}{C(q_0)}. \tag{2.20}$$

where q1 is the q vector at the next time step, and q_0 is the current vector of generalised coordinates. $C^q(q_0)$ is the derivative of the constraint equations at the current time step and $C(q_0)$ is the constraint equations at the current time step. It is checked whether the initial guess of the solution is an accurate answer to the set of constraint equations. If the error is larger than the defined solution tolerance I_{tol}, a new guess is made according to:

$$q_1^{i+1} = q_1^i - \frac{C^q(q_1^i)}{C(q_1^i)}. \tag{2.21}$$

Here, q_1^i initialises at q_0. New estimates are made until the solution tolerance is reached, the amount of estimates maximum defined iterations I_{max} or the new solution does not change enough anymore as defined by s_{tol},

$$s_{tol} = |q_1^{i+1} - q_1^i| \tag{2.22}$$

With the constraint equations, the Newton-Raphson solver, the track layout and the current cart position a new position can be solved for any segment of the track. The multi-body model repeats this process to find the position of the train for a complete element.

2.5.2 Simulations

The input needed to determine the cart position in a single segment is given by 46 values:

- The current position of the train is described by a parametric Bézier variable for each wheel t coordinate for each wheel, the angle of each cart and three coordinates for each linking bar, for a total of 15 values.
- Nodes that define the considered track, with an X, Z and pitch for each node. 10 nodes for a total of 30 values.
- New position of the front wheel of the train, which is given by t in Eq. (2.18)

The outputs are 14 values which is the new position of the rest of the train. To use the input and output as data for the surrogate model the parametric Bézier variables are converted to actual coordinates because a surrogate model does not have access to the Bézier functions. This is done by substituting the Bézier variable in its appropriate equation. The Z coordinate of wheel one is obtained such that $B_z^{q_1} = W_z^1$, the X coordinate of wheel one is obtained by $B_x^{q_1} = W_x^1$ and so on. The position of the centre of mass (CoM) is required for an optimisation algorithm because from there the reach envelope is most easily defined. The CoM is set in the middle of the black box as per Fig. (2.9). The relation between the angle of the cart and the coordinate is then given by:

$$CoM_\theta^1 = \tan^{-1}\left(\frac{W_y^2 - W_y^1}{W_x^2 - W_x^1}\right). \tag{2.23}$$

Here, CoM_θ^1 is the angle of cart one. The coordinate of the (CoM) is then found by the geometric relations with respect to the wheel:

$$\begin{aligned} CoM_x^1 &= W_x^1 - 0.5L_1\cos(CoM_\theta^1) - (0.5L_4 + L_2)\sin(CoM_\theta^1) \\ CoM_z^1 &= W_z^1 - 0.5L_1\sin(CoM_\theta^1) + (0.5L_4 + L_2)\cos(CoM_\theta^1) \end{aligned} \tag{2.24}$$

where CoM_x^1 and CoM_z^1 are the X and Z coordinate of the CoMs for cart one and the L refer to the lengths in Fig. (2.9). In the same way, the CoMs for cart two and three is formulated by the following four equations:

$$\begin{aligned} CoM_x^2 &= W_x^3 - 0.5L_1\cos(CoM_\theta^2) - (0.5L_4 + L_2)\sin(CoM_\theta^2) \\ CoM_z^2 &= W_z^3 - 0.5L_1\sin(CoM_\theta^2) + (0.5L_4 + L_2)\cos(CoM_\theta^2) \\ CoM_x^3 &= W_x^5 - 0.5L_1\cos(CoM_\theta^3) - (0.5L_4 + L_2)\sin(CoM_\theta^3) \\ CoM_z^3 &= W_z^5 - 0.5L_1\sin(CoM_\theta^3) + (0.5L_4 + L_2)\cos(CoM_\theta^3) \end{aligned} \tag{2.25}$$

With the CoMs defined the surrogate model outputs can be used in an optimisation algorithm using the local sequential approach. To surrogate the multi-body model a neural network is used.

2.5.3 Supervised Learning

A neural network is a function approximator, able to represent any type of relation between inputs and outputs. The building blocks of a neural network are neurons. An input vector x and a set of weights w together with a bias b are given to a neuron. The neuron calculates the weighted sum $g(x)$ given by:

$$g(x) = w \cdot x + b \tag{2.26}$$

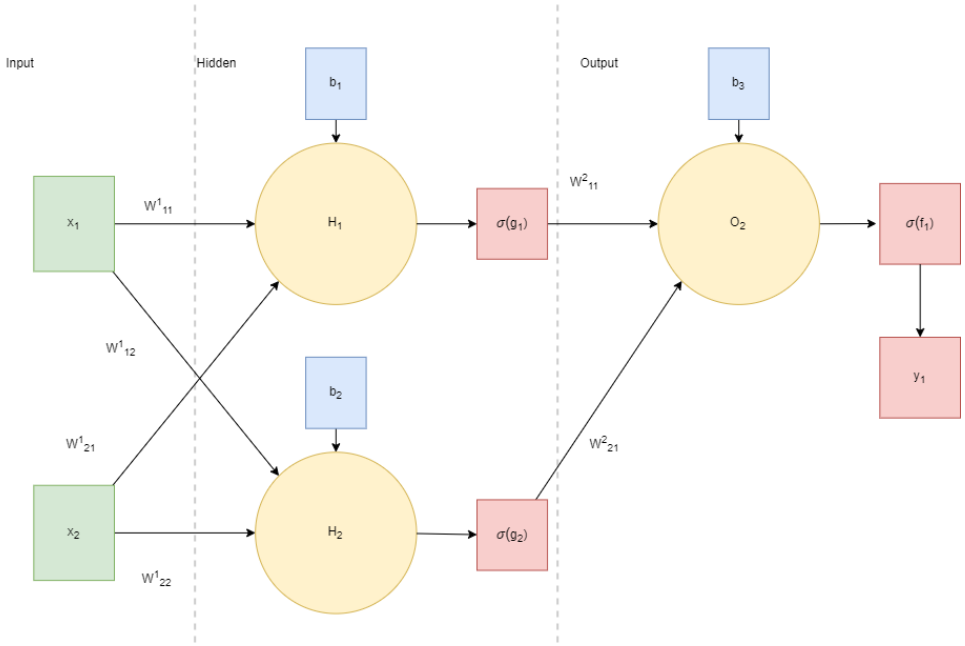


Figure 2.10: A small feedforward neural network.

An activation function σ determines the output of the neuron. Several neurons together can form a neural network. The simplest neural network is called a feedforward neural network (FNN) [48]. The network is fully connected (FC) if each neuron of each layer is connected to all the neurons of the following layer. A small FNN is shown in Fig. (2.10). This network consists of three layers, an input layer, a hidden layer and an output layer. The input is the vector x with size two. There are two hidden neurons which accept the input x and convert it to the vector g using Eq. (2.26). g is of size two. g is put through an activation function which creates the input for the output neuron. The output neuron again uses Eq. (2.26) to generate f. f gives the output y after going through the activation function. A neural network can have an arbitrary amount of inputs, outputs and hidden neurons. The width and the activation function are hyperparameters. The width is the number of neurons in a single layer, the depth is the number of hidden layers. The weight and biases are learned during the training of the neural network.

Neural networks learns the mapping between inputs and outputs by first giving all the weights and biases a random initial value. Then the forward pass is made, by using the input, weight and biases such that the predicted output is generated. This output is compared to the desired output, giving an error often called the loss. With the loss known, one uses the so-called backpropagation algorithm to update the weights and biases in network [49].

Supervised learning relies on labelled training data. Labelled training data consists of inputs and desired outputs to train the neural network. The inputs of the learning problem are the current position of the train, the nodes that define the track and the new position of the front wheel. The current position of the train is given by:

$$[W_x^1\ W_z^1\ W_x^2\ W_z^2\ X_{bar}^1\ Z_{bar}^1\ \theta_{bar}^1\ W_x^3\ W_z^3\ W_x^4\ W_z^4\ X_{bar}^2\ Z_{bar}^2\ \theta_{bar}^2\ W_x^5\ W_z^5\ W_x^6\ W_z^6]. \tag{2.27}$$

Here,W_x^1 is the X position of wheel one, W_z^1 is the Z coordinate of the node and so on. The subscript bar denotes the coordinates for the linking bars. The nodes that define the track are given by:

$$[N_x^0\ N_z^0\ N_\theta^0\ N_x^1\ N_z^1\ N_\theta^1\ .....\ N_x^9\ N_z^9\ N_\theta^9]. \tag{2.28}$$

The new position of the front wheel is then given by $W_x^1(n)$ and $W_z^1(n)$. Here, (n) means the value for the new position. The input consists of 50 total values. The output is the new position of the train which consists out of 16 values:

$$[W_x^2(n)\ W_z^2(n)\ X_{bar}^1(n)\ Z_{bar}^1(n)\ \theta_{bar}^1(n)\ W_x^3(n)\ W_z^3(n)\ W_x^4(n)\\ W_z^4(n)\ X_{bar}^2(n)\ Z_{bar}^2(n)\ \theta_{bar}^2(n)\ W_x^5(n)\ W_z^5(n)\ W_x^6(n)\ W_z^6(n)]. \tag{2.29}$$

The generated training data is shuffled and split into a training set and a validation set. The training set is used for training the neural network and the validation set is used to check how well the neural network approximates new data. The loss function in supervised learning evaluates variants of the error $y - \hat{y}$, where y is the desired output and $\hat{y}$ is the predicted output. The mean square error (MSE) is commonly used in regression [50]. The loss is the average square difference between true and predicted values. MSE is formulated by:

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2 \tag{2.30}$$

where N is the number of predictions. The mean square error loss generates a convex shape towards a global minimum, this means that gradient descent is easily applied. The main disadvantage is how MSE deals with large outliers in the data set, as the penalty for such an error is squared. However, the data generated has no outliers since each output-input pair is generated by the same set of equations and thus MSE is chosen as the loss function. This completes the description of the surrogate model and of our complete roller coaster design problem.

Chapter 3

Reinforcement Learning Methodology

The method chosen for the optimisation of the design of a roller coaster element is reinforcement learning. RL is a sequential decision-making process catering to the sequential decision problem of our design problem. The chapter starts with an introduction to RL. Followed by an explanation and formulation of the Markov Decision Process (MDP), the framework on which reinforcement learning is built. In the last section, to solve the MDP, the RL algorithm is presented.

3.1 Introduction to Reinforcement Learning

The basic principle of the proposed RL model consists of the agent, a track builder, interacting with the environment, a roller coaster element. The RL process is shown in Fig. (3.1). Here, the agent takes an action based on the current state, which is sent to the environment. An observation is received and modified by the interpreter to a state and reward. Then a new action is chosen until a terminal state is reached. All states between the initial state and terminal state are called an episode. As an example, the RL model of the design of a roller coaster element using the sequential formulation is considered;

- The action is the placement of a new node
- The observation is all raw data obtained from the environment, including but not limited to the nodes of the track and the forces experienced by the passenger.
- The state is the relevant information needed to make a decision, derived from the observation i.e. the maximum acceleration in a segment.
- The rewards are given when something is done in line with the design goals such as getting closer to the desired endpoint.
- The episode is the creation of the element, by placing all available segments, defined by N_{seg} or until a terminal state is reached like the roller coaster running into the ground.
- Every turn, so every decision of the agent, is one segment of the element.
- The agent uses the state to decide the next action according to a policy, a decision-making strategy based on the current state and rewards

The goal is to let the agent accumulate as much reward as possible in a single episode. This is done by collecting data from the environment through experience. The agent is, at first, randomly picking actions and recording their impact on the environment and thus the rewards. If a certain action or a set of actions in a row gives a high reward the agent tries to replicate this experience. By doing this the agent exploits the current knowledge gained about the environment and rewards. However, the agent does not only want to repeat itself, it wants to improve itself. To improve itself it sometimes slightly changes an action such

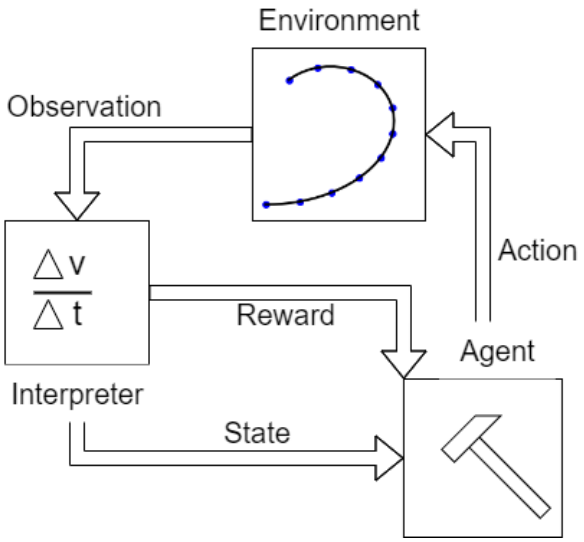


Figure 3.1: Visualisation of reinforcement learning.

that a different outcome is obtained, this is called exploration. If a change in action increases the reward the agent reinforces this slight change in that specific action. If a change in action lessens the reward it diminishes that choice of action. To formally define the states, rewards and actions a Markov decision process (MDP) is required.

3.2 Formulation of Markov Decision Process

An MDP is a mathematical framework for sequential decision-making. Formally an MDP consists of the four variables (S, A, r, P) and the decision making strategy $\pi(S)$ in which

- S is the set of states called the state space, the segment formed with a Bézier curve and the dynamics experienced by the point mass between the two most recently placed nodes. A state s is an element in the state space.
- A is the set of actions called the action space, the position of a new node. An action a is an element in the action space.
- r is the immediate reward after going from one state to the next if, for instance, the coaster is going in the desired direction.
- $P(s'|s, a)$ is the probability of getting to state s′ from state s following action a. Since model-free RL is considered $P(s'|s, a)$ is not learned and thus does not have to be defined [51].
- $\pi(S)$ is the policy, a decision-making strategy which depends on the state and potential rewards. An example is to always create a straight segment such that segments are within acceleration limits.

The MDP is initialised at the start point one with an initial state S_0, an action A_1 is chosen by a policy $\pi(S)$ depending on the current state. Consequently, a reward r_1 and new state S_1 are given. This creates a trajectory τ which looks like:

$$\tau = (S_0, A_1, r_1, S_1, A_2, r_2, S_2, ...). \tag{3.1}$$

The complete trajectory defines the placement of multiple nodes, defining multiple segments which together form an element. The goal of the MDP is to find a $\pi(S)$ which maximises the reward in a trajectory. The optimal policy is denoted by π^*. For this the discounted sum of rewards $R_(\tau)$ is used:

$$R(\tau)_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} ..... = \sum_{t=0}^{T} \gamma^t r_t. \tag{3.2}$$

where γ, a value between zero and one, is the discount factor, signalling the importance of future rewards. If γ is one, distant rewards are equally important to the immediate reward. If γ is zero you only care about immediate rewards. Tuning this parameter allows the policy to focus on either the next segment for a γ of zero or for instance the end position of the roller coaster for a γ of one.

The sequential formulation of the design of an element is transformed by describing the segments and nodes as the state and the placement of nodes as the actions. A possible situation for the MDP is drawn in Fig. (3.2). In the figure, the MDP is at State one. In the previous step Action one, the placement of a node was chosen from the initial state, then state one and a reward for creating a safe segment are given. From state one, a choice must now be made. The green circle represents the distance towards the next node L_{seg}. Two possible choices for Action two are sketched, one in purple and one in red. Important to note is that the choice of actions is only dependent on the current state, and not on previous states since they cannot be altered anymore. When the system is only dependent on the current state and not on past states it adheres to the Markov Property as is required for an MDP. To make an informed decision between the two actions a more mathematical description is needed.

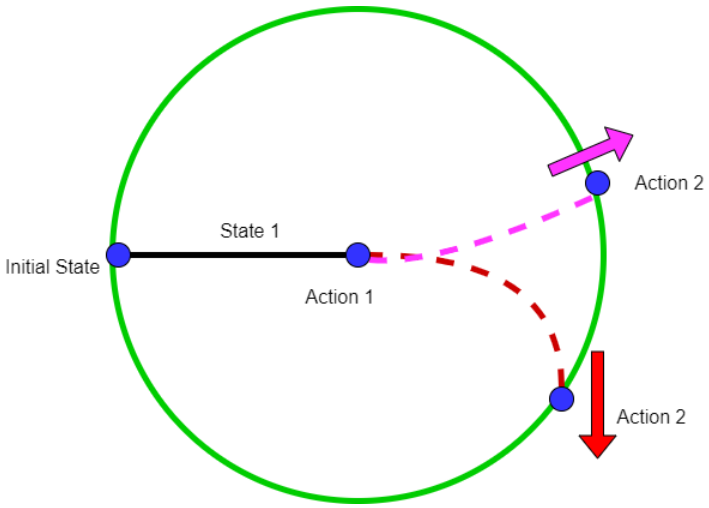


Figure 3.2: Visualisation of the MDP.

The current state provides the information in the MDP to make a decision. To give a more exact definition of the current state S and be able to make a good choice on the next action in Fig. (3.2), the example from section 2.4 is revisited. The example problem presents the variables that are used to make decisions. In the example, information about the velocity and accelerations are used to make decisions. The same variables define the choices for the MDP. Thus, in the state not only the current direction and position of the segment are present but also the effects experienced by the passenger. The state space then forms a selection of the appropriate nodal coordinates, location, velocity, energy, time, acceleration and jerk. To obtain the dynamics over a segment, the Bézier curve is first determined by Eq. (2.1). Then, the segment is divided into N_{int} pieces with a linear relationship between each point. Eq. (2.11), Eq. (2.6b), Eq. (2.12) and Eq. (2.12) are then applied to find the values of energy, velocity, time, acceleration and jerk. Since acceleration relies on the radius of curvature, which itself relies on a second derivative the first two terms and last two terms of the determined accelerations are considered invalid. For jerk these are the first three terms and last three terms. The process of obtaining the values for the dynamics experienced by the passenger is referred to as determining the track dynamics. To determine whether the current state

of the roller coaster is good or not rewards are needed.

Rewards allow one to express the design goals through simple statements. To repeat, the three design goals are positioning of the coaster, safety of the track dynamics and thrill experienced by the passenger. The reward related to reaching the end goal r_{goal} is defined by:

$$r_{goal} = N_{seg} - \sqrt{(X_{goal} - X_{end})^2 + (Y_{goal} - Y_{end})^2 + (Z_{goal} - Z_{end})^2} \tag{3.3}$$

Where the subscript $goal$ denotes the desired end position and the subscript end is the position of the last placed node. Furthermore, it is not only important that the end goal is reached. The track tangent at the end of the element is required to go in the desired direction. This desired direction is an input defined by the user. The reward r_{dir} is introduced that utilises the normal vector to determine the direction at the end of the element. The normal vector is formulated as in the third column of Eq. (2.7):

$$Normal(\beta, \alpha, \gamma) = [\cos\alpha \sin\beta \cos\gamma + \sin\alpha \sin\gamma, \sin\alpha \sin\beta \cos\gamma - \cos\alpha \sin\gamma, \cos\beta \cos\gamma] \tag{3.4}$$

And the reward is formulated by:

$$\begin{aligned} \bar{G_n} &= Normal(P_{goal}, W_{Goal}, R_{goal)} \\ \bar{E_n} &= Normal(P_{end}, W_{end}, R_{end)} \\ r_{dir} &= N_{seg}(1 - \sqrt{1 - \frac{(\bar{G_n} \cdot \bar{E_n}) + 1}{2}}) \end{aligned} \tag{3.5}$$

The dot product of the end normal vector $\bar{E_n}$ and goal vector $\bar{G_n}$ define the difference of the angle between the two vectors. This is a value between minus one and one. To obtain a usable measure out of the dot product, the dot product is converted to a value between zero and one. This conversion is done by first adding one and consequently dividing by two. One minus the value between zero and one creates a value which is one if the vectors are the complete opposite and zero if they are the same. One minus the square root is then utilised to emphasise small differences between the two vectors. Since r_{goal} and r_{dir} apply to a point in space which is unknown if only the current node is present in the state, the position, direction of the desired endpoint and the segments remaining S_{numb} are also added to the state. These are described by the list: $[X_{goal}\ Y_{goal}\ Z_{goal}\ P_{goal}\ W_{goal}\ R_{goal}\ S_{numb}]$.

The second design goal is to create a safe element. A reward r_{safe} of one is given for the placement of a safe segment. To further encourage the agent to place all segments a reward r_{place} is given as well. A reward of one is given for every segment placed. In the thesis, the four rewards r_{goal}, r_{dir}, r_{safe} and r_{place} are described as the general rewards. The general rewards are valid for every MDP thus also every element in the case study section.

The third design goal is to design a thrilling element. The associated reward depends on the element that is designed. The general rewards are considered more important due to them being strict goals. The reward associated with the thrill criterion is only given once 95% of the general rewards are met. Examples of a thrill criterion are the steepness of the slope and airtime in an element. With all three design goals expressed by a reward, four MDPs are formulated.

3.2.1 MDP with point mass physics

The first three MDPs that are defined feature the use of point mass physics.

MDP for an X-Z element

The state for a problem with only an X and a Z coordinate, such as a drop, is constructed by selecting the following state space:

$$S = [X\ Z\ P\ E\ v\ A_{vert}\ j\ t\ X_{goal}\ Z_{goal}\ P_{goal}\ S_{numb}] \tag{3.6}$$

where X and Z are the spatial coordinates, P is the pitch, E is the total energy in the system , v is the velocity, A_{vert} is the vertical acceleration, j is the jerk and S_{numb} is the amount of segments left. To describe the position and direction of a newly placed node, two actions a^1 and a^2 are needed in A. One action determines the position in X and Y, where all possible choices are defined by L_{seg} as per the green circle in Fig. (3.2). The other action determines the direction vector at the new node, the pink and red arrow in the aforementioned figure.

The first action is formulated by using a change of the local pitch ΔP and the second action by the adjustment of the local pitch δP such that:

$$A = [a^1\ a^2] = [\Delta^P\ \delta^P]. \tag{3.7}$$

The effect of the action on the placement of the new node is given with help of the rotation matrix:

$$\begin{aligned} X_{t+1} &= X_t + L_{seg}\cos(P_t + \Delta^P_{t+1}) \\ Z_{t+1} &= Z_t - L_{seg}\sin(P_t + \Delta^P_{t+1}) \\ P_{t+1} &= P_t + \Delta^P_{t+1} + \delta^P_{t+1} \end{aligned} \tag{3.8}$$

where X_{t+1}, Z_{t+1} and P_{t+1} are the coordinates and pitch at the new node, Δ^P_{t+1} and δ^P_{t+1} the actions chosen from state t and X_t, Z_t and P_t are the coordinates, yaw and roll at the previous node. The rest of the state space is determined by the track dynamics.

MDP for an X-Y element

Similarly, state space and action space are defined for a problem with only an X and Y coordinate such as a banked turn. Instead of pitch, yaw and roll are implemented for lateral movement and to lessen lateral G's. The state space then looks like this:

$$S = [X\ Y\ W\ R\ E\ v\ A_{vert}\ A_{lat}\ j\ t\ X_{goal}\ Y_{goal}\ W_{goal}\ R_{goal}\ S_{numb}]. \tag{3.9}$$

where Y is the spatial coordinates, W and R are the yaw and roll and A_{lat} is lateral accelerations. The action space is also similar but a third action is needed to describe the roll. The action space is then given by:

$$A = [a^1 \quad a^2 \quad a^3] = [\Delta^W \quad \delta^W \quad \delta^R]. \tag{3.10}$$

where Δ_W is the change of yaw, δ_P is the adjustment of yaw and δ_R is the adjustment of roll. The effect of the actions on the placement of the new node is given with help of the rotation matrix:

$$\begin{aligned} X_{t+1} &= X_t + L_{seg}\cos(W_t + \Delta^W_{t+1}). \\ Y_{t+1} &= Y_t + L_{seg}\sin(W_t + \Delta^W_{t+1}). \\ W_{t+1} &= W_t + \Delta^W_{t+1} + \delta^W_{t+1} \\ R_{t+1} &= R_t + \delta^R_{t+1} \end{aligned} \tag{3.11}$$

where X_{t+1}, Y_{t+1}, W_{t+1} and R_{t+1} are the coordinates, yaw and roll at the new node, Δ^W_{t+1}, δ^W_{t+1} and δ^R_{t+1} the actions chosen from state t and X_t, Y_t, W_t and R_t are the coordinates, yaw and roll at the previous node. The rest of the state space is determined by the track dynamics.

MDP for a 3D segment

The third MDP is a combination of the X-Z and X-Y elements to obtain a 3D formulation. The state spaces are combined into:

$$S = [X\ Y\ Z\ P\ W\ R\ E\ v\ A_{vert}\ A_{lat}\ j\ t\ X_{goal}\ Y_{goal}\ Z_{goal}\ P_{goal}\ W_{goal}\ R_{goal}\ S_{numb}]. \tag{3.12}$$

The action space is combined into:

$$A = [a^1 \quad a^2 \quad a^3 \quad a^4 \quad a^5] = [\Delta^P \quad \delta^P \quad \Delta^W \quad \delta^W \quad \delta^R]. \tag{3.13}$$

$$\begin{aligned} X_{t+1} &= X_a + L_{seg}\cos(P_t + \Delta^P_{t+1})\cos(W_t + \Delta^W_{t+1}) \\ Y_{t+1} &= Y_a + L_{seg}\cos(P_t + \Delta^P_{t+1})\sin(W_t + \Delta^W_{t+1}) \\ Z_{t+1} &= Z_a - L_{seg}\sin(P_t + \Delta^P) \\ P_{t+1} &= P_t + \Delta^P_{t+1} + \delta^P_{t+1} \\ W_{t+1} &= W_t + \Delta^W_{t+1} + \delta^W_{t+1} \\ R_{t+1} &= R_t + \Delta^R_{t+1} \end{aligned} \tag{3.14}$$

and the rest of the state space is determined by the track dynamics.

3.2.2 MDP with MBD physics

The fourth MDP utilises an MBD formulation of the cart instead of a point mass. The use of MBD in the RL model enables a more elaborate MDP, as more information is available. Instead of information about a single-point mass, the effects of the track on multiple carts can be used to formulate rewards and states. Obstacles are more easily taken into consideration since the reach envelope can be found with the position of the carts. This MDP considers a roller coaster train of three carts.

MDP for an X-Z element

The MDP model uses the centre of mass (CoM) to keep track of the roller coaster carts. The state space is then adapted. First, values that are valid for the complete train such as track, velocity, energy and end goal remain the same. Positions and angles are that of the centre of masses (CoM). The state space is defined by the following two equations:

For the whole cart:

$$S = [X\ Z\ Pitch\ E\ v\ t\ X_{goal}\ Z_{goal}\ P_{goal}\ S_{numb}]. \tag{3.15}$$

And the separate coordinates for each cart:

$$S = [CoM_x^1\ CoM_z^1\ CoM_\theta^1\ CoM_x^2\ CoM_z^2\ CoM_\theta^2\ CoM_x^3\ CoM_z^3\ CoM_\theta^3]. \tag{3.16}$$

Here, CoM_x^1, CoM_z^1 and CoM_θ^1 are the X, Z and Pitch coordinates of the CoMs for cart one. The contents of the action space remain the same as for the point mass model as seen in Eq. (3.7:

$$A = [a^1\ a^2] = [\Delta^P\ \delta^P]. \tag{3.17}$$

The actions now define the exact position of the track rather than the heartline as shown in Fig. (2.4). The average height of the three CoMs is used to determine the potential energy and velocity. The rewards are reduced to; r_{goal}, r_{dir} and r_{place}. To keep computation times low the surrogate model is used to approximate a multi-body model of the roller coaster train and obtain the CoMs.

These four MDPs give a progressively more dimensional, thus often more difficult to solve problem. This allows for a gradual increase in difficulty during implementation. To optimise and find solutions for the MDPs a reinforcement learning algorithm is used. The goal of reinforcement learning is to maximise the expected reward of Eq. (3.2) over the episode by finding the optimal policy.

3.3 Solution Algorithms

In RL, the decision strategy must be learned. This learning can be done in two ways, learning every possible state-action pair or learning a decision strategy function. Learning every state-action pair is not feasible with large sizes of state and action space [52] such as the roller coaster. Thus a decision strategy function is needed. An FNN is used for parameterisation, representing a function for the decision strategy given a state as input and outputting an action. To update the parameters of a FNN and find an optimal policy, a loss function is required.

Methods to find the optimal policy are divided into value-based methods and policy-based methods. Both methods aim to maximise the expected reward. Policy-based methods do so by directly optimising the decision policy. Value-based methods estimate the value of being in a specific state and taking a particular action.

3.3.1 Value-based methods

In RL, value defines how good it is to be in a certain state ahead of the long-term result. For instance, if the goal of a roller coaster element is to turn right, then most likely all states that turn in the left direction are of lower value. This allows an agent to make decisions without knowing anything about the model of the environment. The value function is related to the specified rewards and allows the agent to analyse the quality of the current state and the possible actions. The state value gives the expected reward for being a given state s. The state value is given by:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau) \mid s] \tag{3.18}$$

in which $\mathbb{E}$ denotes expectation with respect to the policy from state s. When a value of a state is known the best possible action dictates the policy by selecting the action with the highest value. The value of a being in a state s and choosing a certain action a is given by the state-action value:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau) \mid s,a] \tag{3.19}$$

in which $\mathbb{E}$ denotes expectation with respect to the policy from state s and taking action a. The state value and state-action value functions are linked through the formula:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi}[Q^{\pi}(s,a)] \tag{3.20}$$

This means that $V^{\pi}(s)$ is the weighted average of $Q^{\pi}(s,a)$ over all possible actions by the probability of each action. To help the agent figure out which action is the best given the current state and possible actions the state-action advantage function is defined:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3.21}$$

Here, the action with the largest increase in value can be chosen as an optimal action.

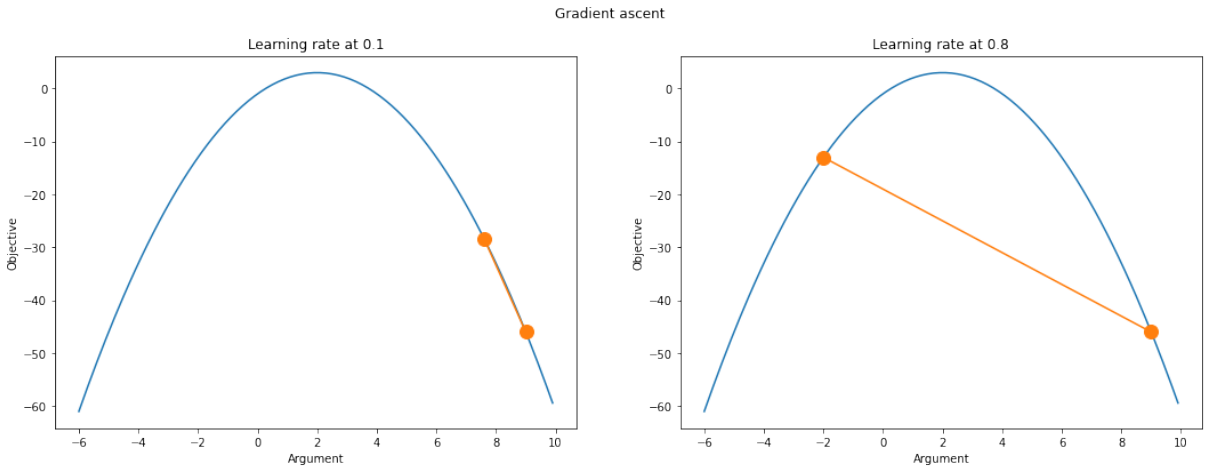


Figure 3.3: Effect of different learning rates on gradient ascent.

3.3.2 Policy-based methods

Instead of a learned value function with every state, it is possible to directly influence the policy itself. For instance, if the goal of a roller coaster element is to turn right, the action of going left is directly changed by the action of going right. This requires the need for a parameterised policy θ, in which the action for every state visited by the policy is remembered. The parameterised policy is given by $\pi_\theta(a|s)$. This means that $\pi(a|s)$ is mapped to θ and kept in memory until the policy is updated.

Policy-based functions optimise θ by introducing an objective function, which aims to maximise the expected reward. The objective function is given by:

$$J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)]. \tag{3.22}$$

where the optimal parameterised policy θ^* is obtained by choosing the θ that leads to the highest expected reward,

$$\theta^* = \arg\max_\theta \mathbb{E}[R(\tau)]. \tag{3.23}$$

The change in θ directly affects action such that a larger reward is generated in the next episode. To find the optimal parameters for Eq. (3.23), one can use the gradient-based approach. This is done by applying the log-probability method [53] expressing the gradient as an expected value:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_0}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi_\theta\left(a_t \mid s_t\right)\right) R(\tau)\right] \tag{3.24}$$

Then θ is updated through gradient ascent through the following equation:

$$\theta \leftarrow \theta + \lambda \nabla_\theta J(\theta) \tag{3.25}$$

where λ is the learning rate defining the width of the updating step. The learning rate is an important parameter and determines how much the policy can be updated in a single step. The effect of a different λ on the gradient ascent algorithm is shown in Fig. (3.3). Here, the smaller learning rate of 0.1 undershoots the maximum and the high learning rate of 0.8 overshoots the maximum. In RL if the learning rate is set too high, the policy is updated towards only optimising the last evaluated state-action pair. The policy then changes a lot at every time step, causing a very unstable model. If set too low, learning can be too slow such that convergence is not reached within a reasonable time frame. In Eq. (3.25) $\nabla_\theta J(\theta)$ takes the form of an expected value and is computed at the end of an episode in basic policy-based algorithms. This can result in bad actions cancelling out good actions, such that both remain undetected. This can be

prevented by using an actor-critic method.

3.3.3 Actor-Critic methods

In Eq. (3.22), $R(\tau)$ is used to find the gradient of $J(\theta)$. However, using $R(\tau)$ means that the full set of actions and states are taken into account, although current actions have no influence on rewards from the previous time states. To reformulate $R(\tau)$ in Eq. (3.24) the advantage function as shown in Eq. (3.21) is chosen and thus the loss function becomes:

$$L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\log\left(\pi_\theta\left(a_t \mid s_t\right)\right)A^{\pi_\theta}\left(s_t, a_t\right)\right] \tag{3.26}$$

Here, L is used to indicate a loss for learning a neural network, replacing $\nabla_\theta J(\theta)$. This is required for the learning of the neural network. The gradient computation of Eq. (3.24) is handled by the backpropagation algorithm. Actor-critic methods use the value function to optimise the policy function. The actor $\pi_\theta(s, a)$ controls the actions of the agent. The critic A_π evaluates the quality of the action taken by the actor. To avoid having to evaluate two separate value functions, the advantage function Eq. (3.21) is estimated by approximation. This approximation uses the immediate reward after an action and the estimated value for the next state, such only a value function for $\hat{V}$ is needed to formulate the advantage function:

$$\hat{A}(s_t, a_t) = (r(s_t, a_t) + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t). \tag{3.27}$$

Eq. (3.27) is called the temporal difference (TD) advantage estimate. However, the advantage relies heavily on the accuracy of the predicted value function $\hat{V}(s_t)$. Instead, a generalised advantage estimate (GAE) can be used:

$$\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma^{GAE},\lambda)} :=& (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \ldots\right)\\
=& (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \lambda^2\left(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V\right) + \ldots\right)\\
=& (1-\lambda)\left(\delta_t^V\left(1 + \lambda + \lambda^2 + \ldots\right) + \gamma\delta_{t+1}^V\left(\lambda + \lambda^2 + \lambda^3 + \ldots\right)\right.\\
&\left. + \gamma^2\delta_{t+2}^V\left(\lambda^2 + \lambda^3 + \lambda^4 + \ldots\right) + \ldots\right)\\
=& (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \ldots\right)\\
=& \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V
\end{aligned} \tag{3.28}$$

in which the exponential average of the TD advantage is used [54]. Here, $\delta_t = \hat{A}_t(s_t, a_t)$ and λ is the exponential weight discount. If we set this to 0, then we are left with the TD advantage estimate.

To determine the policy function, proximal policy optimisation (PPO) is used in this thesis [55]. PPO is a model-free, policy-gradient reinforcement learning method and is developed specifically for stability, ease of implementation and easier-to-tune hyperparameters. The ease of use of this algorithm makes this choice very suitable since the primary goal is to validate the formulation of the MDP. Also, PPO is able to use discrete or continuous actions, which is advantageous when building or learning from environments [56]. Discrete actions are described by a finite set of options, while continuous actions are described by a probability distribution. Discrete actions are easier to learn and thus are used in the first iterations of the model, then later replaced by continuous actions which are able to describe a more accurate action and better capture the uncertainty in the expectation of Eq. (3.26).

3.3.4 Proximal policy optimisation

Building on policy gradient methods, PPO aims to improve the parameterised policy function at the next step θ_{k+1} by looking at the loss of the last used policy θ_k using the following equations:

$$\theta_{k+1} = \arg\max_{\theta} L(\theta_k, \theta) \tag{3.29}$$

with the loss being defined by:

$$L\left(\theta_k, \theta\right) = \underset{(s,a)\sim\pi_{\theta_k}}{\mathbb{E}} \left[\min\left(\Pi\left(s, a, \theta, \theta_k\right) \hat{A}_t^{\mathrm{GAE}}(s,a), g\left(\varepsilon, \hat{A}_t^{\mathrm{GAE}}(s,a)\right)\right)\right] \tag{3.30}$$

where

$$\Pi\left(s, a, \theta, \theta_k\right) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k(a|s)}} \tag{3.31}$$

is the ratio of the policy for θ and θ_k, and

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon)A & A > 0 \\ (1-\varepsilon)A & A < 0 \end{cases} \tag{3.32}$$

is a clipping function where ϵ is a hyperparameter, determining the range of the clipping. $L\left(\theta_k, \theta\right)$ measures the quality of θ is compared to θ_k. If the estimate of the advantage function is positive, taking action a is preferable over the average of all actions available in that state. The policy is then updated to make such an action more likely. However, due to the possibility of very large errors in training data of the policy algorithm, the ratio $\Pi\left(s, a, \theta, \theta_k\right)$ can be very large. The error in training data arises because the state action pairs from which it learns are self-generated. Large updates are often harmful to the performance of the policy, for this reason, PPO uses the clipping function $g(\varepsilon, A)$. For a very good action $g(\varepsilon, A)$ then clips the update to $1+\epsilon$. For a very bad action $g(\varepsilon, A)$ then clips the update to $1-\epsilon$. The clipping creates a more stable learning algorithm. The defined loss function is used to update the FNN representing the policy functions. The value function is updated by Eq. (2.30) through supervised learning since the actual discounted reward is always calculated after estimating the value function. The neural nets are used to find the optimal policy of the proposed MDPS.

Chapter 4

Implementation and Training

This chapter covers all details needed for the implementation of the proposed methods. The first section starts with the implementation of PPO, such that the self-made reinforcement learning environments can be tested, or custom environment for short. Following the implementation of PPO, the options for creating a custom environment are explored and the development process of the custom environment is presented. In the last part, the tuning of the MDP, custom environment and explanation of the pseudo-code are presented. In the second section implementation of the MBD, generation of data and learning of the surrogate model are discussed.

4.1 Reinforcement learning implementation

For the creation of the custom reinforcement learning environments, the Gym python package is chosen [57]. Gym offers a simple Python-based template which is capable of representing general RL problems. From the documentation of Gym several guidelines are followed: (1) use a normalised observation space, (2) normalise your action space to a symmetric array when continuous. Then reshape actions inside the environment and(2) start with a simplification of the model. Normalisation means that an associated value remains between minus one and one. Simplification of the model means that not all state's actions and rewards are implemented in a single try.

4.1.1 Learning of the environment

Building on the Gym custom environment are the implementations of several RL algorithms by stable baselines 3 (SB3). SB3 is a community implementation of various algorithms based on the OpenAI gym framework. The name SB3 originates from the numerical stability of the featured algorithms. The same implementation of an RL algorithm often varies in performance as is mentioned in [58]. This is why the SB3 python package is utilised and not a self-made implementation. SB3 is chosen since it is often used in literature and offers efficient and numerical stable performance for the PPO algorithm [59].

To understand the implementation of a machine learning algorithm first the concepts of rollouts, epochs and batches are explained. A rollout in reinforcement learning is a collection of experiences from the agent interacting with the environment and following a certain policy. The amount of experiences in a rollout is given by the parameter n_{steps}. Epochs are used to define the number of times the data in the rollout is used to update the policy. The amount of epochs is given by n_{epochs}. In an epoch, thus a single use of the rollout, the experiences are divided into batches. Each batch computes an average loss using the loss equations and updates the policy. A batch is identified by the parameter n_{batch} indicating the number of experiences in the batch. A machine learning algorithm utilises batches to update the loss function instead of single data points to generate a smoother learning experience. A single experience is prone to error compared to a collection of experiences. To define how many experiences are collected over the full learning period, n_{total} is defined also referred to as training steps. With the introduction

of the concepts in this paragraph, the pseudo-code of the stable baseline 3 implementations of PPO is given in Alg. (1). n_{cur} denotes the current amount of completed learning steps in the pseudo-code. The pseudo-code starts with initialisation followed by the creation of a rollout. The rollout is cycled through n_{epoch} times, divided into batches updating the policy and value neural network. This process is repeated until all the training steps are executed. The training steps is an input for the learning algorithm.

Algorithm 1 PPO stable baselines 3 implementation

Initialise Environment and learning parameters
while $n_{cur} < n_{total}$ **do**
  Let the agent (policy network) generate n_{steps} experiences ▷ 1 rollout
  for $k = 1$, $k{+}{+}$, while $k < n_{epochs}$ **do** ▷ Use rollout n_{epoch} times
    for $m = 1$, $m{+}{+}$, while $m < \frac{n_{step}}{n_{batch}}$ **do**
      Evaluate actions by computing values $V(s_{t+1})$ and $V(s_t)$ using the experience in Eq. (3.18)
      Compute advantages with predicted value for $\hat{V}(s_{t+1})$ given in Eq. (3.27)
      Compute the policy loss given in Eq. (3.30)
      Compute the value loss by calculating MSE loss in Eq. (2.30)
      Generate gradients and update the agent
    end for
  end for
  $n_{cur} =+ n_{steps}$
end while

The relevant parameters and default values are as follows: the learning rate $\lambda = 3e^{-4}$, used in Eq. (3.25). $n_{step} = 2048$, the number of experiences generated per rollout. $n_{batch} = 64$ the number of experiences in a single batch, from which a single update on the policy and value function is calculated. The total amount of batch in a rollout is $\frac{n_{step}}{n_{batch}} = 32$. $n_{epochs} = 10$ is the number of times a single rollout is used. $\gamma = 0.99$ the discount factor used in Eq. (3.18) and Eq. (3.28). $\lambda_{GAE} = 0.95$. $\epsilon = 0.2$ is the clip range used in Eq. (3.32). For the neural network, which is used to represent the policy and value function a structure of 2 hidden layers each with 64 neurons is chosen. The activation function is tanh and the optimiser is Adam. The policy and value network or actor and critic both have a separate FFN which is trained, although shared layers are possible but not recommended for novel applications [59]. All the values come from the default settings of the SB3 implementation. This algorithm is proven in solving many RL problems and is also suitable for use in the creation of a custom environment [57].

4.1.2 Custom environment

With a working learning algorithm, the MDPs formulated in the previous chapter are implemented in a step-by-step fashion as per rule three of Gym custom environments. Normalisation is not applied from the start, because direct insight into state and action space is found to be helpful to have when bug fixing. Normalisation is applied once the environment works well to decrease training times. First, the selection possibilities, restricted by the use of SB3, for the action space, state space and rewards are presented. Then, the roller coaster problem is simplified to a path finder, with a procedural introduction of velocity, accelerations and jerk into actions, states and rewards. The algorithm used to find a solution is the PPO implementation as described in the previous section. All parameters of PPO remained unchanged in this section and only the amount of training steps is specified for every learning problem. When a change to the states, actions or rewards is not mentioned, it can be assumed that it has not changed in the subsequent version of the custom environment.

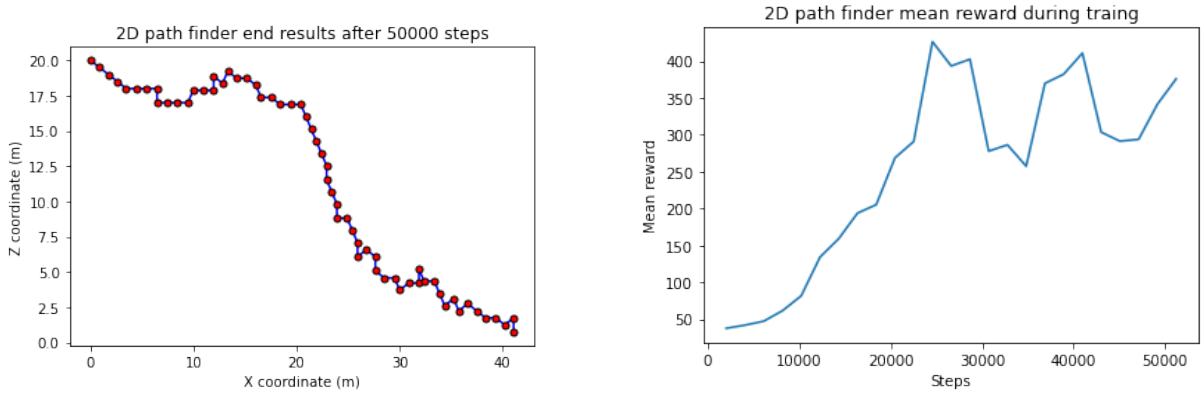
To represent actions in a custom environment two choices are given. Either a discrete set of actions where for each action several choices are given such as Δ^P is either -15, 0 or 15 degrees. Here Δ^P is divided into three choices or bins, each with a 15-degree difference. Δ^P is described by a categorical

distribution, which outputs the probability for each action separately. An example of such a distribution is a 10% probability of -15, 70% probability of 0 and a 20% probability of 15 degrees. The other option is a continuous distribution where Δ^P is represented by a Gaussian distribution, with a specific mean and standard deviation [60]. Boundaries of the distribution are created using an invertible tanh squashing function [61]. This enables one to specify an upper bound and lower bound, i.e. an upper bound of 15 degrees and a lower bound of -15 degrees. Regardless of action type, when an action needs to be picked a sample from the distribution is taken. A sample is taken by drawing a random number between zero and one and seeing where it falls in the distribution. This sampling promotes the exploration of the agent.

Each state is described with an upper and lower bound, which describe the valid values decreasing the search space. For instance, for the X coordinate, the upper bound is 1000 and the lower bound is -1000, a valid value is anything in between. 1000 is chosen since a spatial coordinate is not expected to fall outside of that range. The upper and lower bound in the state space represent a sanity check such that when a state value falls outside of its bound one knowns something is wrong with the custom environment.

In this implementation, a sum of separately defined rewards is utilised. This creates a single reward function with multiple design goals. Rewards can be sparse or continuously given. If a reward is given sparsely for our roller coaster, it means that it is given after the placement of the last segment. If it is set as continuously, the reward is given at every segment. Termination of an episode is also considered a reward since this prevents acquiring additional rewards in the episode.

With these possibilities for actions, states and rewards the first custom environment, a 2D pathfinder in the X-Z plane is created. The action space is discrete. The action space consists of one action Δ^P, with 7 bins. The bins are 90 degrees up, 60 degrees up, 30 degrees up, straight forward, 30 degrees down, 60 degrees down and 90 degrees down. The state consists only of the current X and Z coordinate and the reward r_{goal} of Eq. (3.3) is given at every time step. The start position is X = 0m, Z = 20m and the goal position is X = 40m, Z = 0m. N_{seg} is 60 and L_{seg} is 1m. The model is trained with 50 000 training steps. The end result is seen in Fig. (4.1a). Here, the red dots are the join points and the blue line is the track in between, a total reward of 398 was obtained by this simulation. The agent is able to relatively quickly find a path towards the end goal. The learning curve is plotted in Fig. (4.1b). The learning curve shows the number of steps versus the mean reward over an episode during the training process. In this specific figure, a steep rise in mean reward is seen in the first 25 000 steps, afterwards, the model starts to overtrain resulting in a volatile mean reward.



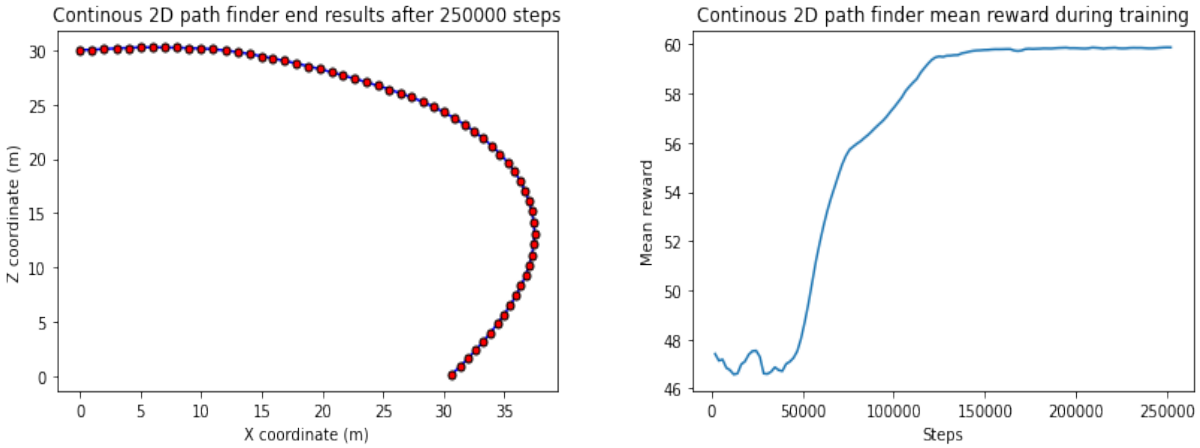
(a) (b)

Figure 4.1: Discrete path finder result and training.

The next version of the custom environment features a continuous action space, with Δ^P taking values in a [-45,45] interval. The state space features the X, Z, Pitch and S_{numb}. The r_{goal} is only rewarded at the

end instead of at every segment, this allows the agent to focus more on the creation of the path itself instead of the end goal. Further, a penalty for going out of geometric boundaries is given. A penalty p_{bound} of minus one is given at every step where an X of Z value exceeds the limit. A similar start position of X = 0m, Z = 30m and the goal position of X = 30m, Z = 0m are the inputs. N_{seg} is 60 and L_{seg} is 1m. The model is trained with 250 000 training steps. More steps are needed since the r_{goal} is made sparse. The lower limit for X is set at minus one which ensures that the path finder does not go backward. The lower limit of the Z coordinate is set at minus one, to ensure the path finder does not go below ground. These limits are chosen instead of zero since both the initial position lie directly on a zero value. The agent would otherwise be immediately penalised if it only overshoots the goal position by a single centimetre. In Fig. (4.2a), the end result is plotted. Here, it is apparent that the agent creates a path that stays as far away as possible from the boundaries. Also, due to the continuous action space, the element trajectory appears smoother than in Fig. (4.1a), although the same effect is possible when more bins are used in a discrete action space. To evaluate the convergence, the maximum possible reward is determined. The maximum possible reward is N_{seg} since r_{goal} is only presented at the end. In Fig. (4.2b) the mean reward is plotted, here the effect of the automatic scaling of the maximum reward is visible. Furthermore, the policy is already close to convergence at 150 000 steps, but the slightest increase in reward is still noticeable.

At this stage, a 2D (X Y) path finder is also created and performed similarly to the 2D (X Z) path finder. To move around in the X Y plane, Δ^W is replaced by Δ^P in the action space and pitch is replaced by yaw in the state space.



(a) (b)

Figure 4.2: Continious path finder result and training.

The third version of the custom environment focuses on the introduction of velocity and total energy into the state space, computed by equation 2.11. For this calculation, a point mass of 250 kg is used and energy losses are ignored. Now the reward r_{place} is added as a continuous reward which is a reward of one for a total of N_{seg} times. This provides incentive through Eq. (3.2) for the agent to place all segments. Also, the episode is terminated if the velocity goes below zero, such that only roller coasters that keep a positive velocity are designed. This third custom environment produced results similar to figure 4.2a, but, the solution is not a realistic one. The negative G's induced by the downward turn are dangerous for riders. This expresses the need for accelerations in the state space and accelerations-based rewards.

To add roller coaster dynamics into the path finder the creation of a Bézier curve at each segment is needed. The action space is expanded such that δ^P is included, the adjustment of the pitch as in Eq. (3.6). The actions are implemented as independent continuous actions, with an upper and lower bound of 30 and -30 degrees respectively. The values for accelerations are then computed through the track

dynamics using a point mass cart as discussed in the previous chapter:

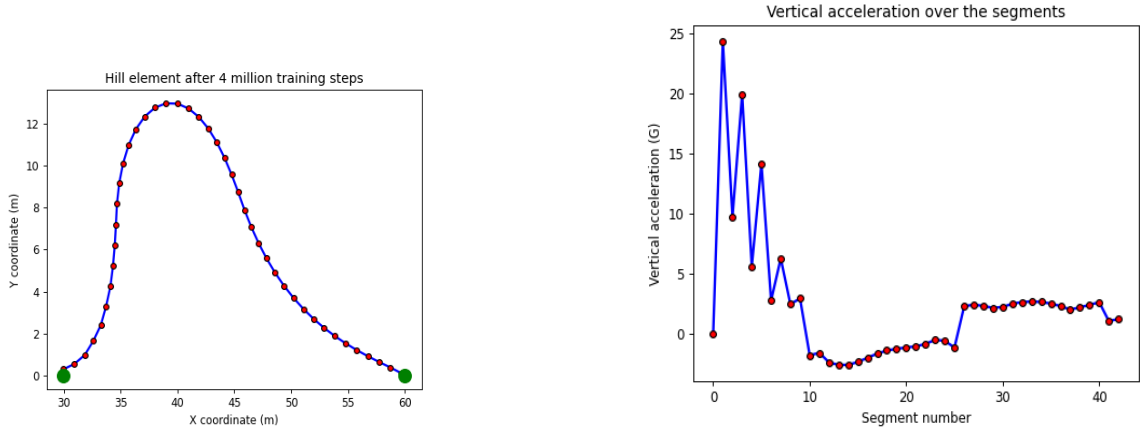
$$\text{Observerved Dynamics} = [E^0\ v^0\ A_v^0\ E^1\ v^1\ A_v^1\ ....\ E^{N_{int}}\ v^{N_{int}}\ A_v^{N_{int}}] \tag{4.1}$$

Here, the superscript denotes the interpolated point of a segment. As the segment is interpolated with N_{int} points, there are N_{seg} sets of values in the observation. The observation is interpreted into a state by only providing the values of velocity $v^{N_{int}}$ and energy $E^{N_{int}}$ at the new node. For accelerations and later jerk, only the highest value in the observation is provided to the state space. I.e. when three values of 11, 14 and 12 are provided as vertical acceleration in the segment then only 14 is provided to the state space. This interpretation of the observed dynamics is then given by:

$$\text{Interpretation} = [E^{N_{int}}\ v^{N_{int}}\ A_v^{max}] \tag{4.2}$$

Here, A_v^{max} denotes the highest value for the acceleration in the observation Eq. (4.1). The interpretation reduces the variables in the state space from three times N_{int} to just three. In this thesis, all figures concerning the acceleration and jerk of the environment plot values from the state space which is why figures of acceleration and jerk can appear discontinuous. r_{safe} is added as a reward, adding to r_{place} and r_{goal}. r_{safe} is a continuous reward of one given if the segment adheres to dynamic limits. r_{safe} incentivises the agent in the creation of an element with acceptable accelerations. The reward r_{goal} is changed in this version. When the endpoint is within 5 meters an extra $\frac{1}{2}N_{seg}$ is rewarded and if the endpoint is within 1 meter an extra N_{seg} is rewarded. The boost in reward is needed to amplify the importance of reaching the end goal. Together these rewards make the total possible reward of 4.5 N_{seg}

To benchmark the proposed theory, a hill element is chosen as the next example of the environment. Hill elements were seen to converge faster than drop elements which makes evaluation of the environment easier. The initial position for the problem is X = 30m, Z = 0m and the goal position is X = 60m, Z = 0m with an initial velocity of 20 m/s. N_{seg} is 42 and L_{seg} is 1m. This makes the total possible reward 189. The cart or point mass properties are M is $250kg$, C_{rr} is 0.03, A = $1.25m^2$ and C_d is 0.1, see Eq. (2.9), Eq. (2.8) and Eq. (2.10). These properties are retained throughout the thesis. The number of total training steps is 500 000 and the simulation converged to a reward of 182. In Fig. (4.3) two plots show the end results for the designed hill element. In the left figure, the track is plotted, and the model generates a hill that reaches the end goal as intended. In the right figure, the maximum acceleration in each segment is plotted. The acceleration exceeded the prescribed maximum of $5G$ vertical acceleration 6 times and the agent finds this acceptable. However, in real-life engineering, this is unacceptable and requires an altercation.



(a) (b)

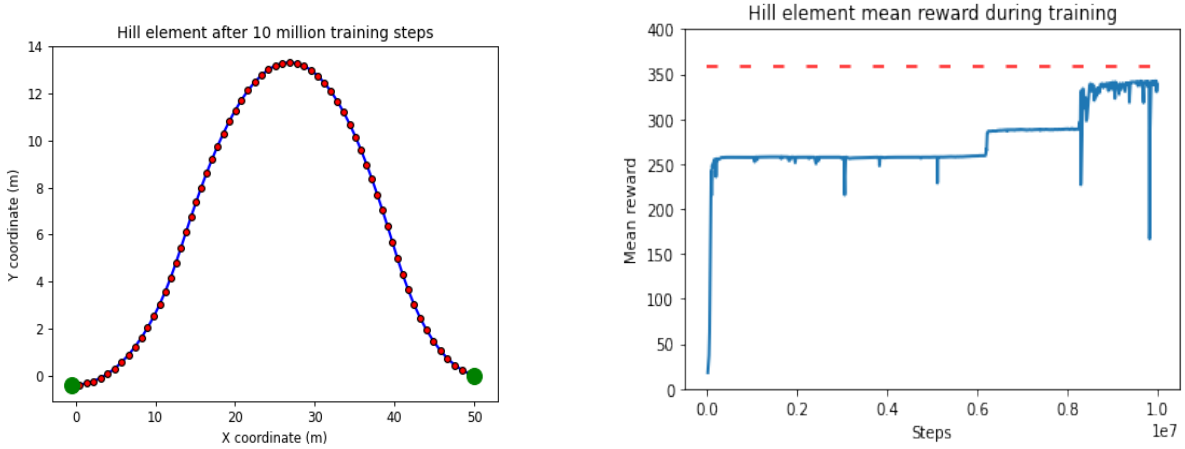
Figure 4.3: Hill element results and learning curve.

To remedy the maximum acceleration issue the termination of the complete episode is utilised as a reward. If accelerations are out of bounds it is possible to immediately terminate the episode. However, this completely hinders the exploration of the agent. Another option to make sure the agent creates an element with acceptable accelerations is obtained from a RL racing game example [52]. Here a penalty is given for every training step of the episode that the car is outside of the track, and eventually, if enough steps are spent outside of the track, the racing game is terminated. A force check variable F_c is introduced into the model, which increases by one every step of the episode where a segment is unsafe. A force check maximum F_c^m is a parameter which dictates how many unsafe segments can be placed before the episode is terminated. In the same way, the geometric check and the geometric maximum G_c and G_c^m are implemented to make sure the agent stays within the geometric boundaries given at the initialisation. The penalty p_{bound} is deleted. F_c^m and G_c^m are set at three for a balance between exploration and the importance of constraints.

A second issue which is seen in Fig. (4.3b) is the large changes in acceleration between segments. A large change in acceleration causes a large jerk. To prevent this, jerk is inserted into the state space. A limit of $15G/s$ jerk is instated and a transgression of this limit adds to the variable F_c, terminating the state if it happens F_c^m times.

As shown in Fig. (4.2a) and Fig. (4.3a) the tangent of the track is headed into the ground at the end of the element. To mitigate this, r_{dir} as in Eq. (3.5) is introduced as a sparse reward. With the rewards of r_{goal}, r_{place}, r_{safe} and r_{dir}, the maximum reward possible for a element is $5\frac{1}{2}N_{seg}$. $2\frac{1}{2}N_{seg}$ is given for reaching the goal within one meter, one N_{seg} is given for matching the tangent and at the end of the element, one is given N_{seg} times for placing a segment and one is given N_{seg} times for placing a segment within safe limits. With all the general rewards in place, a hill element is again designed.

For the hill element, the initial position is X = 0m, Y = 0m and Pitch = 0 degrees and the end goal is X = 50m, Y = 0m and Pitch = 0 degrees. N_{seg} is 65, L_{seg} is 1m and the initial velocity is $20m/s$. The maximum reward possible is 357,5. The result is seen in Fig. (4.4). In the left figure a side view is plotted, where a smooth trajectory is visible. In the right figure, the training steps are plotted against the mean reward over an episode. The red dotted line is the maximum reward and the blue line is the mean reward during training. The curve looks smooth indicating stable learning, the jumps in the curve are caused by the boost given for r_{safe}. However, 10 million training steps were needed to find a good solution. This is a large increase from previous iterations of the custom environment and it calls for a more efficient environment.



(a) (b)

Figure 4.4: Hill element result and vertical acceleration.

The first step in making the environment more efficient is to implement rules 1 and 2 of the Gym documentation. This means normalisation of the state and action space. At this point, Eq. (3.12) and Eq. (3.13) represent the state and action space respectively. The state is represented as a normalised space between minus one and one. A normalisation constant is chosen for every type of state: velocity is divided by 100, degrees, distances, time and accelerations are divided by 1000, jerk is divided by 10000 and the total energy of the roller coaster by 100000.

In the action space, the normalised actions are abbreviated as A^1, A^2, A^3, A^4 and A^5 and are reshaped in the environment. All normalised actions are represented by a bounded normal distribution. Furthermore, determining L_{seg} is introduced as a sixth action A^6 to give the agent more control over the total length of the roller coaster. As the total length is L_{seg}times N_{seg} a less accurate estimate of the total length is needed to reduce possible input bias by giving more control over to the RL model. The reshaping of the sixth action is given by:

$$L_{seg} = (A^6 \cdot \delta_{seg}) + L_{avg} \tag{4.3}$$

To further guide the agent into creating a good roller coaster δ^P and δ^w are made dependent on Δ^P and Δ^W. The dependency comes from the fact that if δ^P equals Δ^P, the Bézier curve is symmetric and the radius of curvature is at its highest. A maximal radius of curvature means minimal centripetal acceleration as per Eq. (2.12). To reshape A^1 the parameter Δ_{MaxP} is introduced which controls the bounds for Δ^P:

$$\Delta^P = A^1 \cdot \Delta_{MaxP} \tag{4.4}$$

Then to create the desired depenency on Δ^P, δ^P is computed by:

$$\delta^P = (A^2 \cdot (\Delta^P * C)) + \Delta^P \tag{4.5}$$

Here, the value of Δ^P when A^2 is zero. C determines the bounds of δ^P using Δ^P. For example if C = 0.5, then the bounds for δ^P are $\frac{1}{2}\Delta^P$ and $\frac{3}{2}\Delta^P$. In the same way, the change and the adjustment of yaw are reshaped using Δ_{MaxW} and C:

$$\begin{aligned} \Delta^W &= A^3 \cdot \Delta_{MaxW} \\ \delta^W &= (A^4 \cdot (\Delta^W / C)) + \Delta^W \end{aligned} \tag{4.6}$$

And lastly, the bounds of the roll are defined by the parameter Δ_{MaxR}:

$$\delta^R = A^5 \cdot \Delta_{MaxR} \tag{4.7}$$

The default values of the reshape parameters Δ_{Max^P}, Δ_{Max^W} and Δ_{MaxR} are 35, 35 and 45 degrees respectively. C is set at 1.8 because at this value no sudden change in the relative direction of the normal of the curve occurs. This means that A_v determined by Eq. (2.13) does not suddenly change direction within a segment limiting sharp increases in jerk. The parameter L_{avg} is set as an input parameter and $\delta_{seg} = 0.1$. Changes in L_{seg} impair velocity continuity, there is only tangent continuity because L_{seg} is no longer constant. this change in continuity definition is not seen to impact the continuity

This concludes the final version of the environment and the performance of the custom environment is evaluated in the next chapter where several 2D and 3D elements are created. The pseudo-code for the implementation is given in Alg. 2. This code is executed when an element is created. In the code, for the initialisation of an element, the following inputs are needed: the specification of whether an XZ, XY or 3D element is needed for the appropriate state and action space, whether point mass physics or the surrogate model is used and the startpoint, endpoint, L_{avg}, N_{seg}, initial velocity and thrill criterion. The thrill criterion is only given if 95% of the general rewards are achieved. In the pseudo-code, the custom environment continues placing segments as long as the total segments S_{numb} is under N_{seg}.

Algorithm 2 Custom Environment

Initialise Environment parameters and set $S_{numb} = 0$ see Eq. (3.13, 3.12)
while $S_{numb} < N_{seg}$ **do**
  Pick actions $\mathbf{A}_{t+1}$ based on current state $\mathbf{S}_t$ using Eq. (3.29) ▷ FNN
  Reshape actions with Eq. (4.3 - 4.7)
  Compute new node from actions by Eq. (3.14)
  Compute Bézier curve given by Eq.2.1
  for $k = 1$, $k{+}{+}$, while $k < N_{int}$ **do** ▷ Cycle through N_int points on Bézier curve
    Compute Velocity with Energy by Eq. (2.11)
    Compute Acceleration and Jerk Eq. 2.12
    Compute Energy Losses and new total Energy Eq. (2.10)
  end for
  S_{numb} is $S_{numb} + 1$
  Compute new state S_{t+1}
  Check F_c and G_c
  Compute rewards r_{safe}, r_{place} ▷ Continous rewards
  if S_{numb} is N_{seg} **then**
    Compute rewards r_{goal} and r_{dir} ▷ Sparse rewards
    if General rewards$> 0.95 \cdot (5.5N_{seg})$ **then**
      Compute thrill criterion reward
      Done
    else
      Not Done
    end if
  end if
end while

All the custom environment default parameters and settings used for the case study are repeated for clarity in the table in the appendix A.2. Both parameters of the learning algorithm and custom environment are investigated. For the learning algorithm, PPO, changes in the learning rate, clip range, discount factor and neural network size and shape were tested. These changes did affect the learning problem but for the worse. The learning algorithm converged much slower or not at all. A change in these parameters might be needed once the learning problem becomes more difficult but they are not further experimented with in the thesis.

With regard to the custom environment parameters changes in L_{avg} and δ_{seg} are especially interesting. These impact L_{seg} directly and the total length of the coaster. To showcase the impact, a simulation is done for a drop element with initial position X = 0m, Z= 20m and P = 0 degrees and endpoint X = 20m, Z = 0m and P = 0 degrees. The initial velocity is 3 m/s and the thrill criterion is steepness. For the first simulation L_{avg} is set at 1m and N_{seg} at 35. For the second simulation, L_{avg} is set at 2m and N_{seg} at 17. The learning curve is plotted in Fig. (4.5). Here, the red line is the first simulation and the blue line is the second simulation. The dotted lines represent 95% of the total general rewards possible for their respective colour. The blue line converges a lot faster thus L_{avg} set to a higher value makes the learning problem easier. However, if the end result is viewed for the blue line as in Fig. (4.6) and Fig. (4.6b) it is seen that the blue curve produced an invalid result. The learning problem is so easy to optimise that the agent learned to cheat very hard on exactly two segments. This is possible due to F_c^m being 3. Afterwards, the agent return to valid values for the accelerations. Due to this phenomenon, the value for L_{avg} remained at 1. Other parameters were also tested but yielded no results worth mentioning.

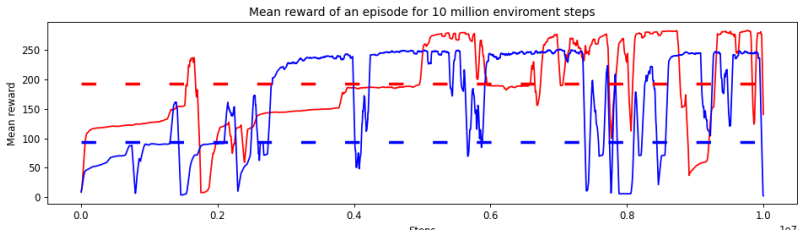
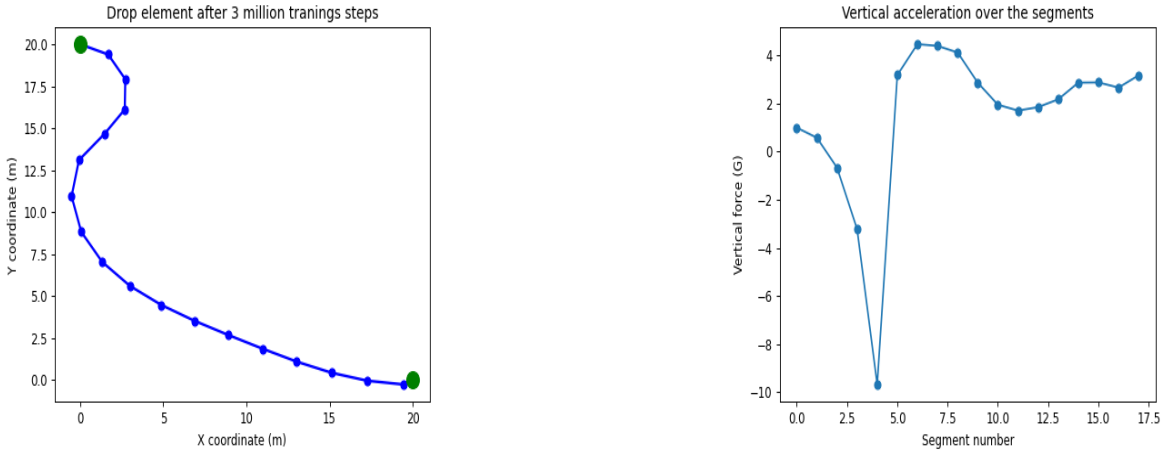


Figure 4.5: The training of the model for two drop elements.



(a) (b)

Figure 4.6: Results for a drop with N_{seg} at 2.

4.2 Surrogate model

The surrogate model for kinematics is used in the design of a 2D drop element. For the training of the model, several intermediate designs for a drop element are obtained using the RL points mass model. The input for the RL model is X = 0m, Z = 10m and P = 0 degrees and an end position of X = 20m, Z = 0m and P = 0 degrees. v_i is 3 m/s, L_{seg} is one and N_{seg} is 31. A trajectory is taken every 100000 training steps. The resulting trajectories are plotted in Fig. (4.7a). Each track has 31 segments, except the first track which has 26 segments for a total of 212 placed segments and are used to generate output data for the surrogate model using the multi-body model.

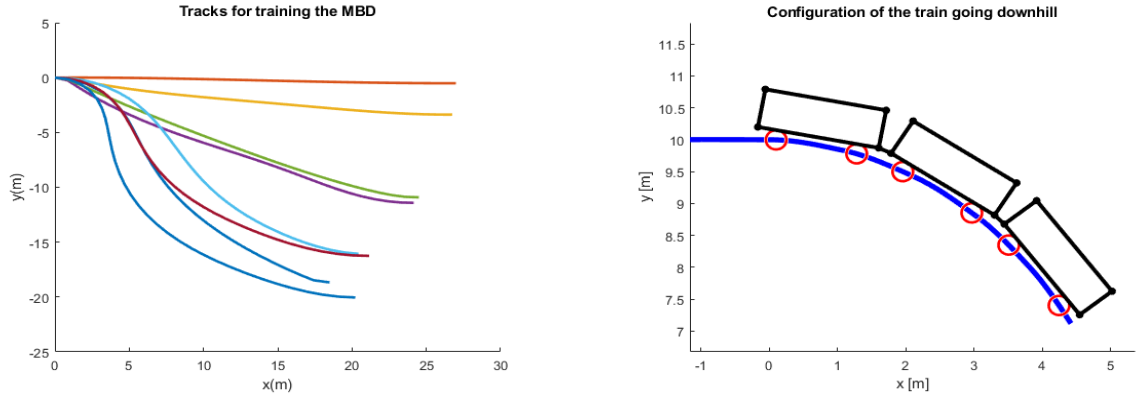
The multi-body equations of Appendix (A.1) are implemented in Matlab. The inputs for a single run of the script are ten nodes, which represent nine segments, the dimensions of the cart and the Newton-Raphson parameters. The initial position is obtained by determining the position of the train on the initial straight track as in Fig. (2.9). The newly generated segment is added to make a track of nine segments. Once the train reaches the end of the ninth segment, the last segment is deleted and a new segment is obtained from the tracks generated by the RL model. The pseudo-code for this script is found in Alg. (3). The code starts by initialising the cart and adding the first segment from the generated python tracks to the initial track. Then for those nine segments, the Bézier curve, constraint equations and derivative of the constraint equation are computed. Then a solution is found by Newton-Raphson iteration. If the solution causes any of the wheels to fall outside of their respective segment, the parameter which keeps track of the wheels is adjusted and again a solution has to be found. The process is repeated by adding a new segment and deleting the last segment until all segments are used.

Algorithm 3 MBD data generation

Load Python track
Initialise position of carts, q_0 and B_{wheel}
for $k = 1, k{+}{+}$, while $k < N_{seg}$ **do** ▷ For every segment in the track once
  Add k segment, remove the last segment
  Compute Bézier Functions Eq. (2.1) for the considered segments, $C^q(q)$ and $C(q)$ by Eq. (2.19)
  for $i = t_i 1, i{+}{+}$, while $i < t_s$ **do** ▷ Determine cart position in the segment
    I = 0, $q_{i+1} = q_i$ ▷ reset
    while any($C_{q_{i+1}} > I_{tol}$, $|q1 - qo| > s_{tol}$ and I < I_{max}) **do** ▷ Newton-Raphson Iteration
      q_{i+1}^{I+1} = q_{i+1}^{I} - $C^q(q_{i+1}^{I})/C(q_{i+1}^{I})$ of Eq. (2.21)
      $I = I + 1$
      q_{i+1}^{I} = q_{i+1}^{I+1}
    end while
    if $|q1 - q0|$ < s_{tol} or I > I_{max} **then**
      End simulation and give error message
    end if
    i = i + Δt
    if any(0 > B_{wheel} > 1 **then** ▷ Check if wheels are still in their assigned segment
      Adapt B_{wheel}
      Compute Bézier Functions Eq. (2.1), $C^q(q)$ and $C(q)$ by Eq. (2.19)
      Restart at reset
    end if
  end for
end for

The cart dimensions, as the length shown in Fig. (2.9) are L_1 is 1.2, L_2 is 0.3, L_3 is 0.15, L_4 is 0.6 and L_5 is 0.2 all in meters. The parameters in a segment are the start point $t_i = 0$, the time step size $\Delta t = 0.001$ which defines the length travelled over the segment and the stopping point at the end of

the segment $t_s = 1$. This means a total of 1000 cart positions are determined in every segment. The Newton Raphson parameters are the new solution tolerance $I_{tol} = 1e-4$ of the multi-body equations, the minimal step changes $s_{tol} = 1e-4$ per time step and the maximum amount of Newton Raphson iterations $I_{max} = 50$. The amount of input segments is 212. For every segment the position is calculated 100 times, this generates a total of 212000 data sets, but for every element, the output data is shifted by 50 data points such that input from i.e. $\frac{3}{20}L_{seg}$ is paired with an output for $\frac{4}{20}\ L_{seg}$. In this way, because of the specified shift, the MBD data is only usable if L_{seg} is one and N_{int} is 20. Due to the shift, the total is reduced to 211650 input-output for supervised learning. Due to these choices, the surrogate model is only viable for the design of a hill element, with the same exact inputs.



(a) (b)

Figure 4.7: Data generation with the MBD model.

4.2.1 Learning and Optimisation of surrogate model

With the data of cart position for the elements collected, a supervised learning algorithm is implemented. For the neural network used as a surrogate model, six different configurations were tested. The learning parameters are 100 epochs, where the data is split into batches of 32 with the MSE loss function, the tanh activation function and Adam optimiser. The results are given in Table (4.1). From the table, it is concluded that the 128-layer width and six hidden layers are the best-performing neural network. The learning curve is given in Fig. (4.8). A somewhat poor convergence on the validation data as compared to the training data is visible but is the best of the considered networks. The sigmoid activation function was also tested but gave worse results than a tanh activation function.

To test the increase in computational efficiency the difference in training times for the surrogate model and the multi-body model in the steps executed per second were determined. The multi-body model computation time is an average obtained by running the multi-body model for an element with 31 segments three times with $\Delta t = 0.05$. This means a total of 630 sets of constraint equations were solved. The computation times were 152, 154 and 159 seconds for an average of 155 seconds. 630 in 155 seconds is a total of 4.1 steps per second. The surrogate model is steps per second is determined by calculating the cart position for the same element, here 0.6 seconds were required to generate an output from the neural network 620 times for a total of 1033 steps per second. With these computation times, the surrogate model is useable in the reinforcement learning environment.

	Layer Width	Hidden Layers	Loss of best model (e-4)
1	32	6	11.10
2	64	6	5.26
3	128	6	4.39
4	256	6	6.10
5	64	8	6.73
6	128	8	5.08

Table 4.1: Result of different surrogate neural networks.

Figure 4.8: Supervised learning of the surrogate model.

Chapter 5

Case Study

In this chapter several elements with the points model are designed, each design also receives one thrill criterion. The 2D point mass element are a hill, a drop, a banked turn and a 2D version of a looping. For the 3D point mass, a turn with a pitch and a narrow 180-degree turn are designed. Lastly, a drop element is generated using the surrogate multi-body model.

The analysis of the result and training for the design of a roller coaster retains the same format for every element. First, the element is introduced followed by the initial conditions. Then 3 intermediate results and the highest rewarded track geometry are plotted. In such a plot the green dot represents the start and end points. This is, followed by two plots of a dynamic value for every element, which originates from the best design. Examples of dynamic values are lateral acceleration or velocity, the chosen dynamic value depends on the element. Lastly, the learning curve. The learning curve consists of the mean reward of an episode versus the training steps. In every learning curve, four green dots represent the four plotted results and a red dashed line indicates the maximum reward of the general rewards. The green dots deviate from the learning curve since a deterministic optimal action is taken instead of a sample from a distribution. These seven plots give a complete picture of the effectiveness of the model for the design of the element.

Results were simulated on a personal laptop of which the specifics are given in appendix A.2. To quantify the difference in training times for the 2D points mass model, 3D point mass model and the 2D surrogate model the steps executed per second were determined. The data is obtained by running the appropriate RL environment for 500 000 training steps. The training steps per second for the 2D points mass, 3D point mass and 2D surrogate model were on average 399, 385 and 79 respectively. From these numbers, it is concluded that the training steps per second slow down as state complexity and computational difficulty increase.

5.1 Hill

A hill is a section in which you start from a low height, at high speed go up and back down again. Fitting a hill within a certain section is challenging due to changes in height, thus velocity and acceleration. The thrill criteria for the hill is airtime, a beloved sensation in which the net accelerations experienced by the body are close to 0 G, this is also known as weightlessness.

The initial conditions for the environment are:

- Start $= [0, 0, 0]$(X Z P), End goal $= [50, 0, 0]$(X Z P)
- $L_{avg} = 1$, $N_{seg} = 65$
- $V_{initial} = 20m/s$
- Action Space $= [\Delta^P, \delta^P, L_{seg}]$
- State space $= [X, Z, P, E, V, A_{Vert}, j, t, X_{end}, Z_{end}, P_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Z is minus one and at X is minus one.
- Thrill criterion is translated to vertical G, where +1 is given for every segment with a maximum absolute acceleration of 0.25 G. This reward is only given if 95% of other rewards are achieved.

To design the hill element 7.5 million training steps are performed to train the agent. The three intermediate and the final result are plotted in Fig. (5.1). Fig. (5.1) shows the result after 250 000 training steps, here the agent just learned to draw a straight line and is going up to get closer to the the end goal. The total reward is 249. In Fig. (5.1b) a hill is formed and the end goal is almost reached for a total reward of 289. Both in Fig. (5.1c) and Fig. (5.1d) at least 95% of the total possible reward is reached however in Fig. (5.1c) only 20 segments satisfy the thrill criterion. For a total reward of 376. The total reward of Fig. (5.1d) is 394, where 38 segments satisfy the thrill criterion. The vertical acceleration with two red lines indicates the limits for the thrill criterion plotted in figure Fig. (5.2a). Here, the ability of the agent to balance vertical acceleration in between 0.25G is illustrated. Furthermore, the safety criteria of 5G are pushed to their limits as well. The velocity of the point mass is shown in Fig. (5.2a). The dependency of the velocity on height is clearly visible from this figure, as it resembles the generated element but is upside down.

In Fig. (5.3) the learning curve is plotted together with the redline that indicates the maximum of the general rewards and the four green dots each matching the figures in Fig. (5.1). This maximum is 357.5. The agent manages to quickly utilise all segments, at roughly 100 000 steps. Then it takes the agent 3.4 million steps to form a proper hill such that the end goal can be reached. Between 3.5 million and 6.5 million the agent is optimising the thrill criteria reaching the best result at 6 million.

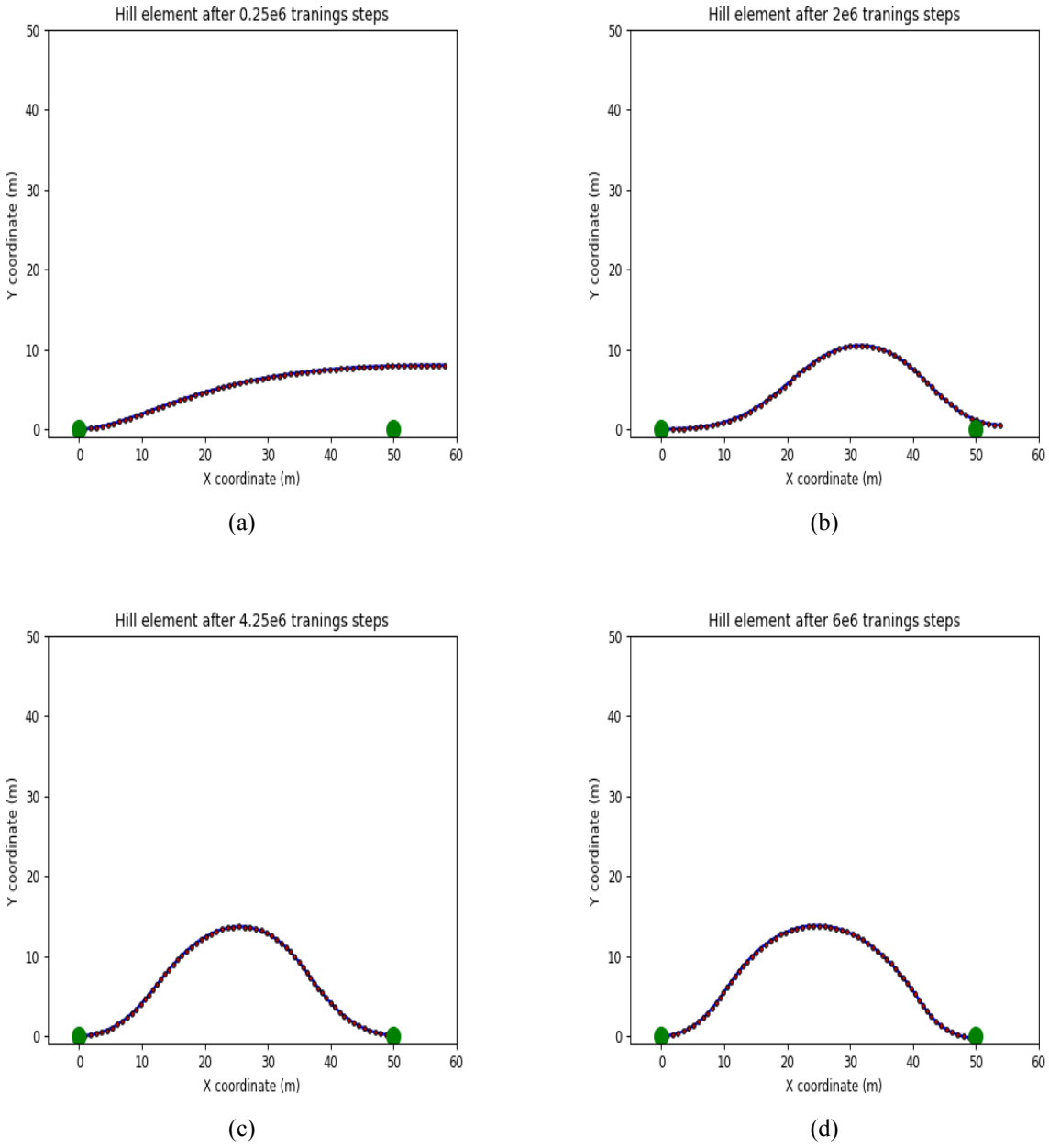
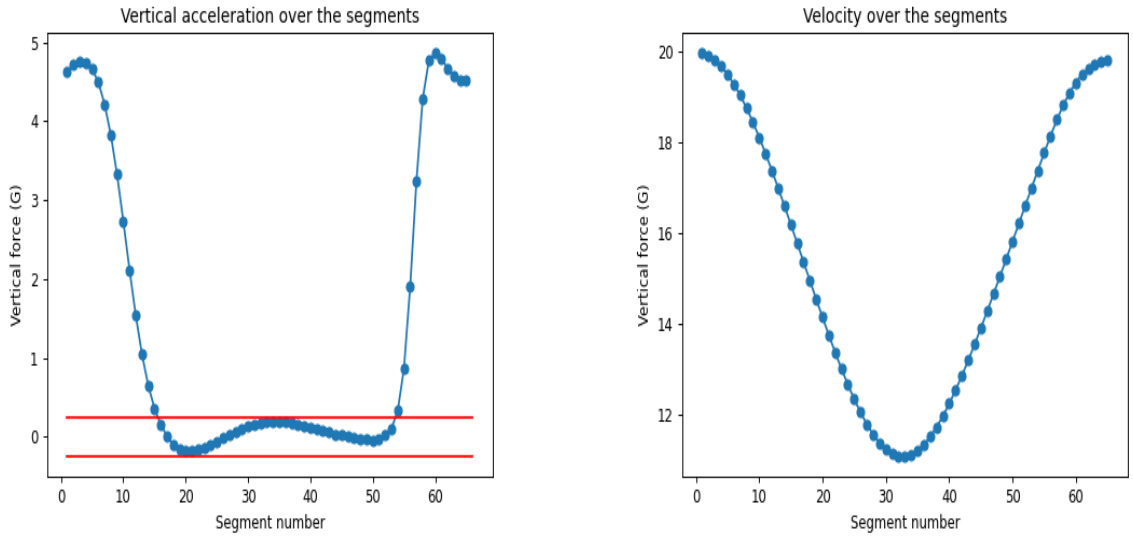


Figure 5.1: Training progression of the hill element.

.



(a) (b)

Figure 5.2: Dynamic values of the best-generated hill element.

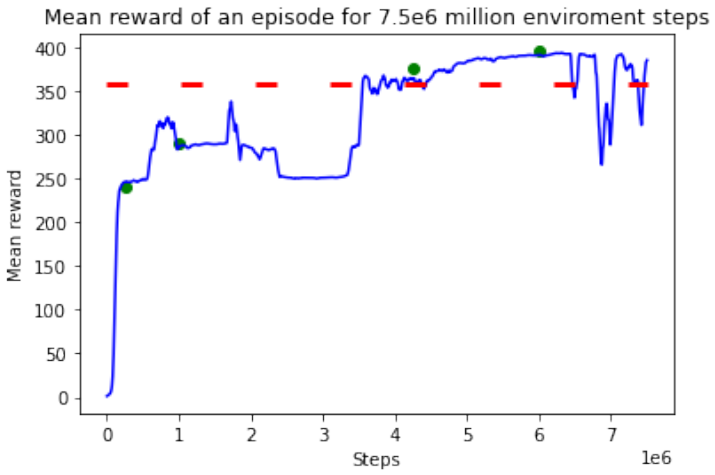


Figure 5.3: Mean reward curve for the hill element.

5.2 Drop

A drop is a section of track where you start at a slow speed after a lift hill and drop down generating high speed due to the conversion from potential to kinetic energy. The challenge here lies in generating a smooth transition towards a steep slope downward and levelling the train back out again.
The initial conditions for the environment are:

- Start $= [0, 20, 0]$(X Z P), End goal $= [20, 0, 0]$(X Z P)
- $L_{avg} = 1$, $N_{seg} = 35$
- $V_{initial} = 3m/s$
- Action Space $= [\Delta^P, \delta^P, L_{seg}]$
- State space $= [X, Z, P, E, V, A_{Vert}, j, t, X_{end}, Z_{end}, P_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Z is minus one and at X is minus one.
- Thrill criteria are steepness, where +1 is given for every degree of the steepest node. This reward is only given if 95% of other rewards are achieved.

To design the drop element 10 million training are performed to train the agent. The three intermediate and the final result are plotted in Fig. (5.4). Fig. (5.4a) shows the result after 250 000 training steps, here the agent created a straight line, similar to the hill element only accumulating a reward of 116. The agent comes close to the end goal by creating a steeper drop in Fig. (5.4b). In Fig. (5.4c) the end goal is reached but the maximum pitch is only 78 degrees, for a total reward of 269. Fig. (5.4d) shows the optimised result with a maximum pitch of 98 degrees and a total reward of 289. The vertical acceleration and pitch for the segments are plotted in Fig. (2.6a) and Fig. (5.5b). Here, the agent reaches the limits for both the low end and the high end of vertical acceleration .

In Fig. (5.6) the learning curve is plotted together with the red line that indicates the maximum of the general rewards and the four green dots each matching the figures in Fig. (5.4). This maximum is 192.5. The agent manages to quickly utilise all segments, at roughly 50 000 steps. Then it takes the agent 3 million steps to form a drop which reaches the end goal. The optimisation of the drop shows a drop in mean rewards caused by the transgression of the safety limits. this is due to the fact that it has to learn about an increasing velocity and an increasing acceleration at the same time in contrast to the hill element. In the hill element, the agent was immediately confronted with a high speed, mainly only learning about vertical acceleration. This likely made the hill element converge faster.

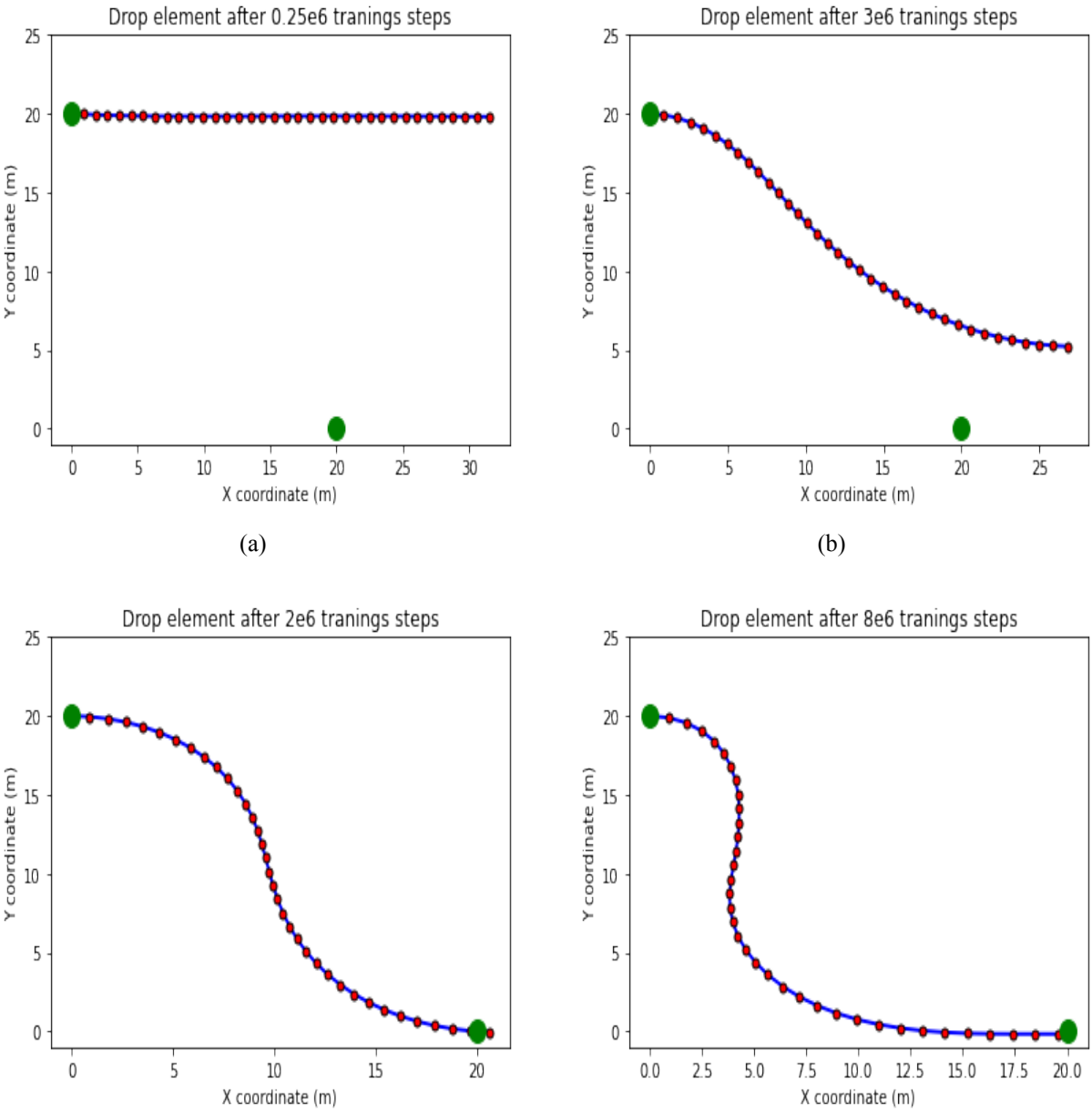
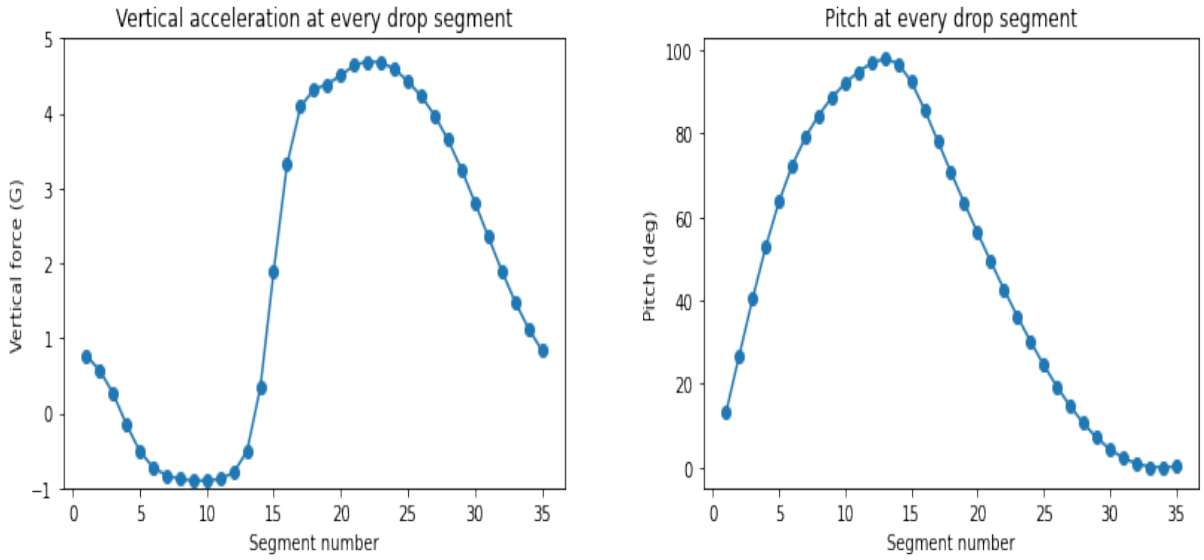


Figure 5.4: Training progression of the drop element.

.



(a) (b)

Figure 5.5: Dynamic values of the generated drop element.

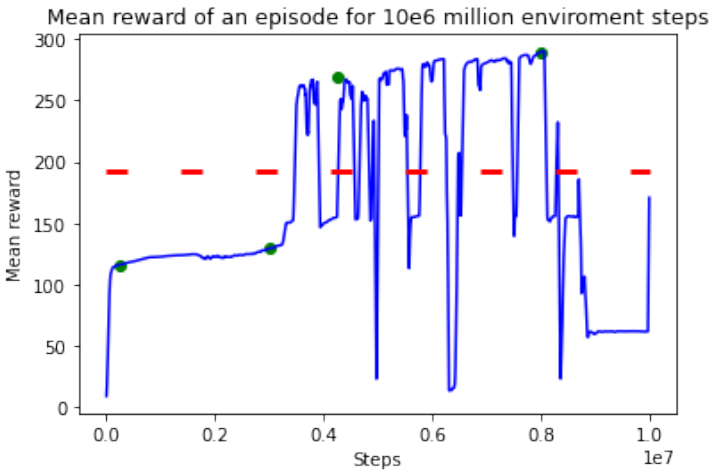


Figure 5.6: Mean reward curve for the drop element.

5.3 Turn

A turn is a straightforward element, without changing height the cart has to go left or right. Banking is crucial for a good turn because by rolling the cart the otherwise uncomfortable lateral G's are converted to vertical G's.

- Start $= [0, 0, 0, 0]$(X Y W R), End goal $= [0, 30, 180, 0]$(X Y W R)
- $L_{avg} = 1$, $N_{seg} = 50$
- $V_{initial} = 20$ m/s
- Action Space $= [\Delta^W, \delta^W, \Delta^R, L_{seg}]$
- State space $= [X, Z, P, E, V, t, A_{vert}, A_{lat}, j, X_{end}, Z_{end}, P_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Y is minus one.
- Thrill criterion is lateral acceleration, where +1 is given for every segment where the absolute lateral force is smaller than 0.2 G. This reward is only given if 95% of other rewards are achieved.

The turn element is designed with a total of 2 million training steps to train the agent. the three intermediate and the final result are plotted in Fig. (5.7). In Fig. (5.7a) the result is plotted after 0.1 million steps, the reward totalled 140. All segments are placed but the end position and direction are still very far from the goal. A straight track is the first design which manages to define all the segments because there are no lateral accelerations if there is no radius of curvature. Fig. (5.7b) and Fig. (5.7c) show the gradual bending of the direction of the track, which goes together with an increase in roll. These figures accumulated a reward of 185 and 221 respectively. Fig. (5.7d) show the end result, with a total reward of 295, and the roll and lateral acceleration of each segment is given in Fig. (5.8a) and Fig. (5.8b). In Fig. (5.7d) the consequence of the use of a normal vector as direction indication is visible. The normal is the same at this point, but the yaw is still different. This signals that a different measure for r_{dir} is required. In Fig. (5.8b) the struggle of the agent in retaining a small lateral G force is visible and the agent succeeds with 20 segments.

In the learning curve of Fig. (5.15) the learning process is visualised. The red line, indicating the maximum reward for r_{place}, r_{safe}, r_{goal} and r_{dir} is at 275. The green dot for Fig. (5.7c) and Fig. (5.7d) differ a lot from the mean at that point, this indicates that the agent is still fairly unsure about its actions.

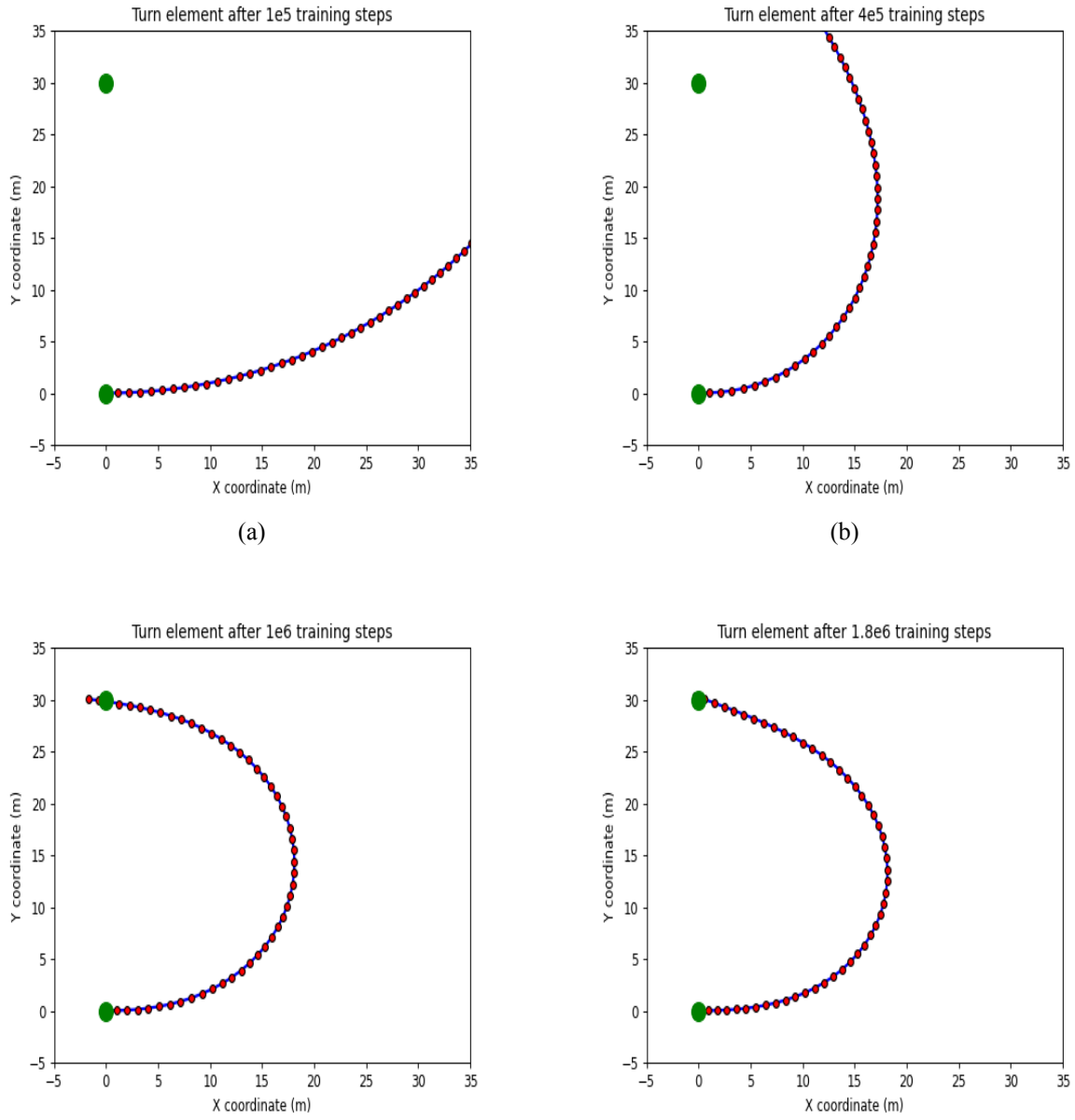
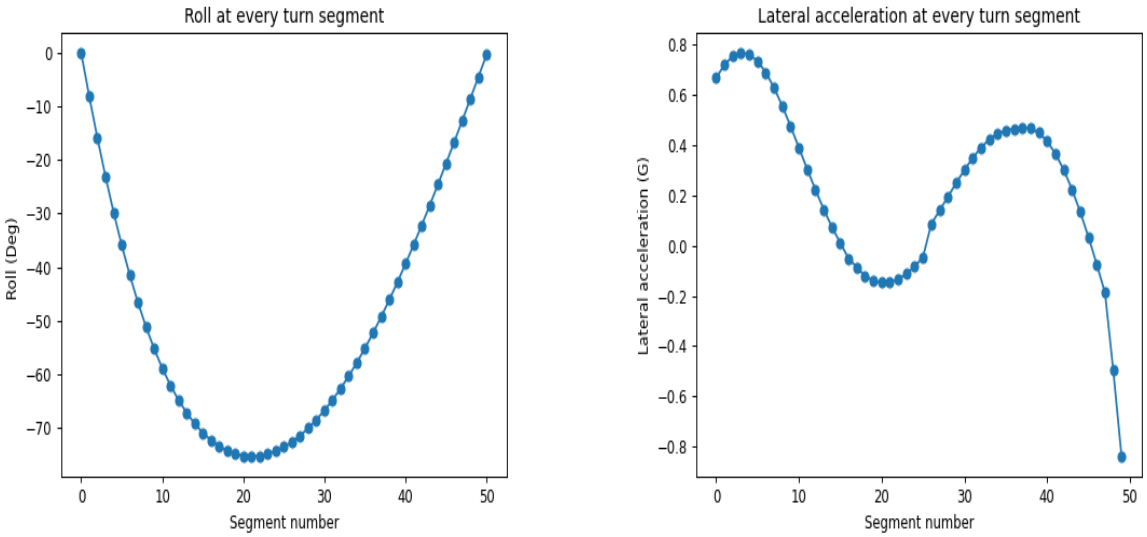


Figure 5.7: Training progression of the turn element.

.



(a) (b)

Figure 5.8: Dynamic values of the best-generated turn element.

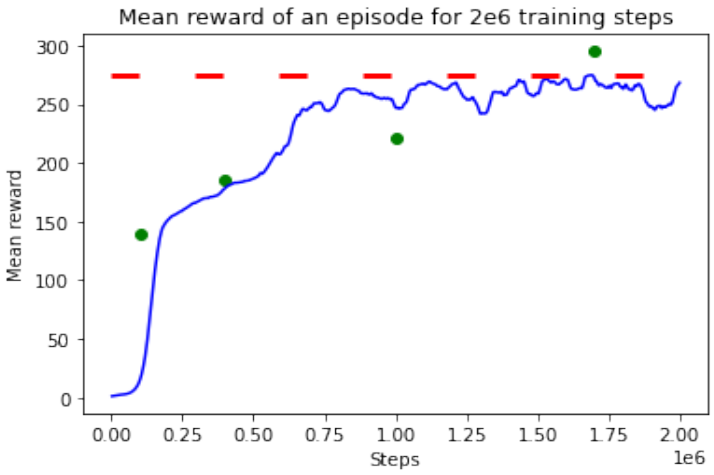


Figure 5.9: Mean reward curve for the turn element.

5.4 Looping

A looping is a track element which completely inverts the rider until they are straight again. Inverting the rider completely calls for a small radius of curvature, but a sudden change in the radius of curvature causes intolerable vertical G's and jerk. A smooth change is needed. Modern loopings often feature a clothoid shape, which is seen as an upside-down water droplet and this is used to validate the designed looping.

- Start $= [0, 0, 0]$(X Z P), End goal $= [20, 0, 0]$(X Z P)
- $L_{avg} = 1$, $N_{seg} = 75$
- $V_{initial} = 20m/s$
- Action Space $= [\Delta^P, \delta^P, L_{seg}]$
- State space $= [X, Z, P, E, V, A_{Vert}, j, t, X_{end}, Z_{end}, P_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Z is minus one, at X is minus one and at X is 21. The boundary at X 21 causes the coaster to perform an inversion since it is the only possible direction.
- Thrill criteria is smoothness measured by the jerk, where +1 is given for every segment where the absolute jerk is lower than 2 G/s. This reward is only given if 95% of other rewards are achieved.

For the looping, a total of 6.25 million training steps are performed to train the agent. The three intermediate and the final result are plotted in Fig. (5.10). In Fig. (5.10a) the result is plotted after 250 000 million steps, the reward totalled 124. This reward comes from the place of 64 segments of which 60 were within safety limits. The environment terminated because F_c^m is violated 4 times. In Fig. (5.10b) the agent manages to place all pieces within safe limits but the end direction and position are still not good, the total reward is 311. In Fig. (5.10c) all general rewards are met within 95% and a total reared of 438 is achieved. However only 28 segments meet the thrill criteria of a maximum of two (G/s) jerk, this is caused by the relatively fast change of curvature in the beginning and end sections of the looping. Fig. (5.10d) is the best solution that is obtained after 6.25 million training steps. Here, the element nicely resembles an upside-down water droplet. The total reward is 482 with a total of 71 segments that full fill the thrill criterion. The loop is placed almost precisely in the middle allowing for a gradual change in curvature and resulting in a smooth ride. The acceleration and jerk are plotted in Fig. (5.11), and in Fig. (5.11a) it is visible that the vertical acceleration is pushed towards its safety limits of minus one G vertical acceleration by the agent. Fig. (5.11b) shows the optimised jerk where two of the peaks are just shy of two (G/s) jerk. The agent prefers to violate the thrill criterion rather than the safety limits since violation of safety limits causes the termination of an episode.

In the learning curve of Fig. (5.12) the learning process is visualised. The red line, indicating the maximum reward for r_{place}, r_{safe}, r_{goal} and r_{dir} is at 375. The path towards 375 is fairly smooth and the bonuses for r_{goal} are visible at 0.5 million and 2.5 million. Optimisation of the thrill criterion starts at roughly 2.9 million, with a gradual improvement until 5 million. After 5 million the agent tries to lower the jerk by increasing the total height of the loop, but this leads to a termination of the episode since the velocity reaches zero.

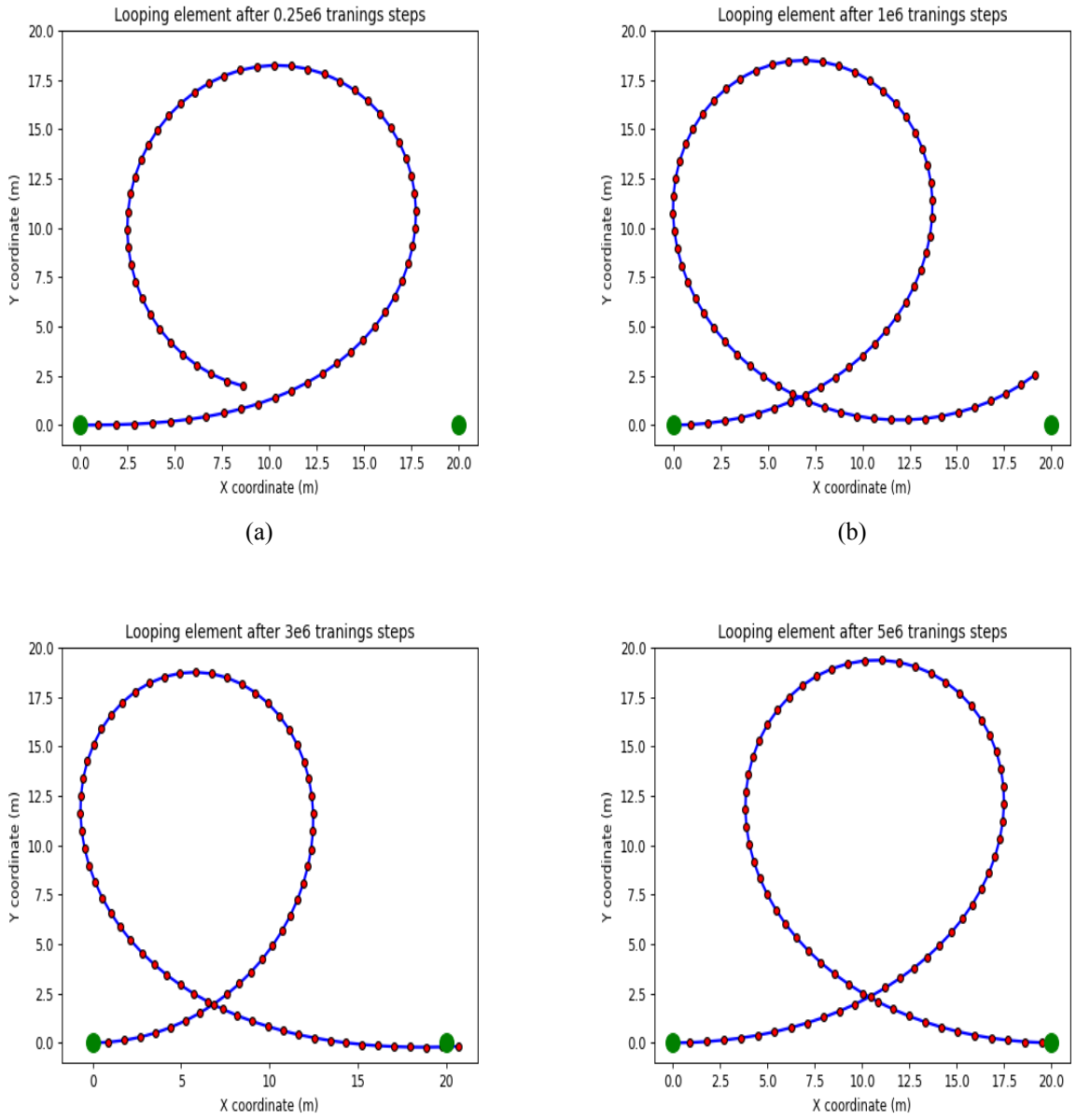
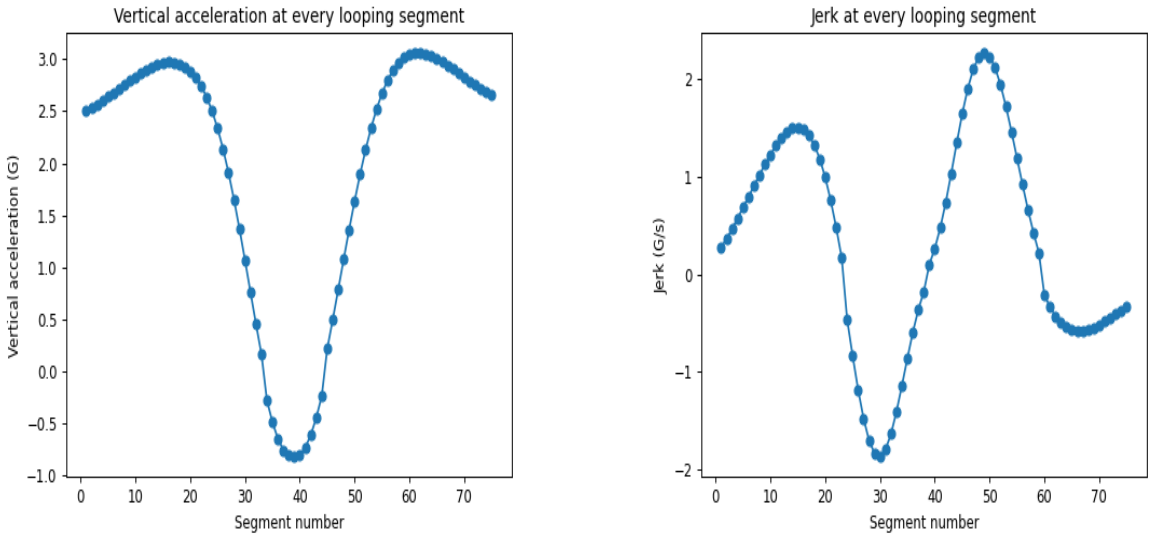


Figure 5.10: Training progression of the looping element.

.



(a) (b)

Figure 5.11: Dynamic values of the best-generated looping element.

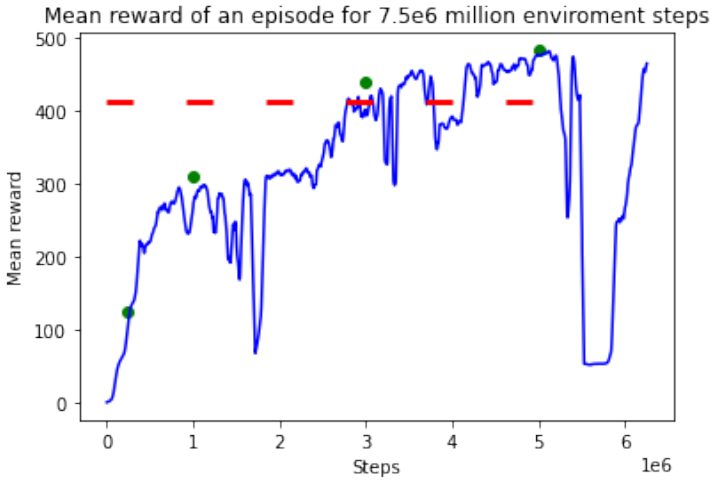


Figure 5.12: Learning curve for the looping element.

.

5.5 Pitched Turn

A pitched turn is a turn which also goes uphill. The complexity here lies in the limited space given for the turn but by going uphill the roller coaster slows down making a sharp turn easier on the lateral and vertical forces at play. To achieve a high average speed through the turn is difficult but does increases the amount of thrill in the element as an average speed does [4].

- Start $= [0, 0, 0, o, 0, 0]$(X Y Z P W R), End goal $= [10, 10, 10, 0, 90, 0]$(X Y Z P W R)
- $L_{avg} = 1$, $N_{seg} = 18$
- $V_{initial} = 15m/s$
- Action Space $= [\Delta^P, \delta^P, \Delta^W, \delta^W, \Delta^R, L_{seg}]$
- State space $= [X, Y, Z, P, W, R, E, V, t, A_{Vert}, A_{Lat}, j, X_{end}, Y_{end}, Z_{end}, P_{end}, W_{end}, R_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Y is minus one, at X minus 1, at X 11, at Z minus 1 and at Z 11.
- Thrill criterion is speed, where +1 is given for every m/s of average speed. This reward is only given if 95% of other rewards are achieved.

For the pitched turn a total of 1 million training steps were performed to train the agent The three intermediate and the final result are plotted in Fig. (5.13). In Fig. (5.13a) the result is plotted after 100 000 million steps, the reward total is 28. The episode was terminated because F_c^m is violated 4 times. In Fig. (5.13b) the agent manages to place all pieces within safe limits but the end direction and position are still not good, the total reward is 61. In Fig. (5.13c) the agent manages to create an element which reaches the endpoint but the normal is too steep to reach the thrill criterion. The thrill criterion is reached in Fig. (5.13d), for an average speed of 9.68m/s and reward of 108. The agent first decreases the speed of the cart by going up in height, deciding to only turn at close to maximum height. The vertical and lateral acceleration are given in Fig. (5.14), and the values are staying well within the allowable limits. From these results is concluded that the agent is able to efficiently manage the pitch, yaw and roll at the same time.

In the learning curve of Fig. (5.15) the learning process is visualised. The red line, indicating the maximum reward for r_{place}, r_{safe}, r_{goal} and r_{dir} is at 99. The main aspect to notice is that only 1 million steps were needed to obtain the design. This is because N_{seg} is a lot lower than in other elements. The learning process also appears a lot smoother.

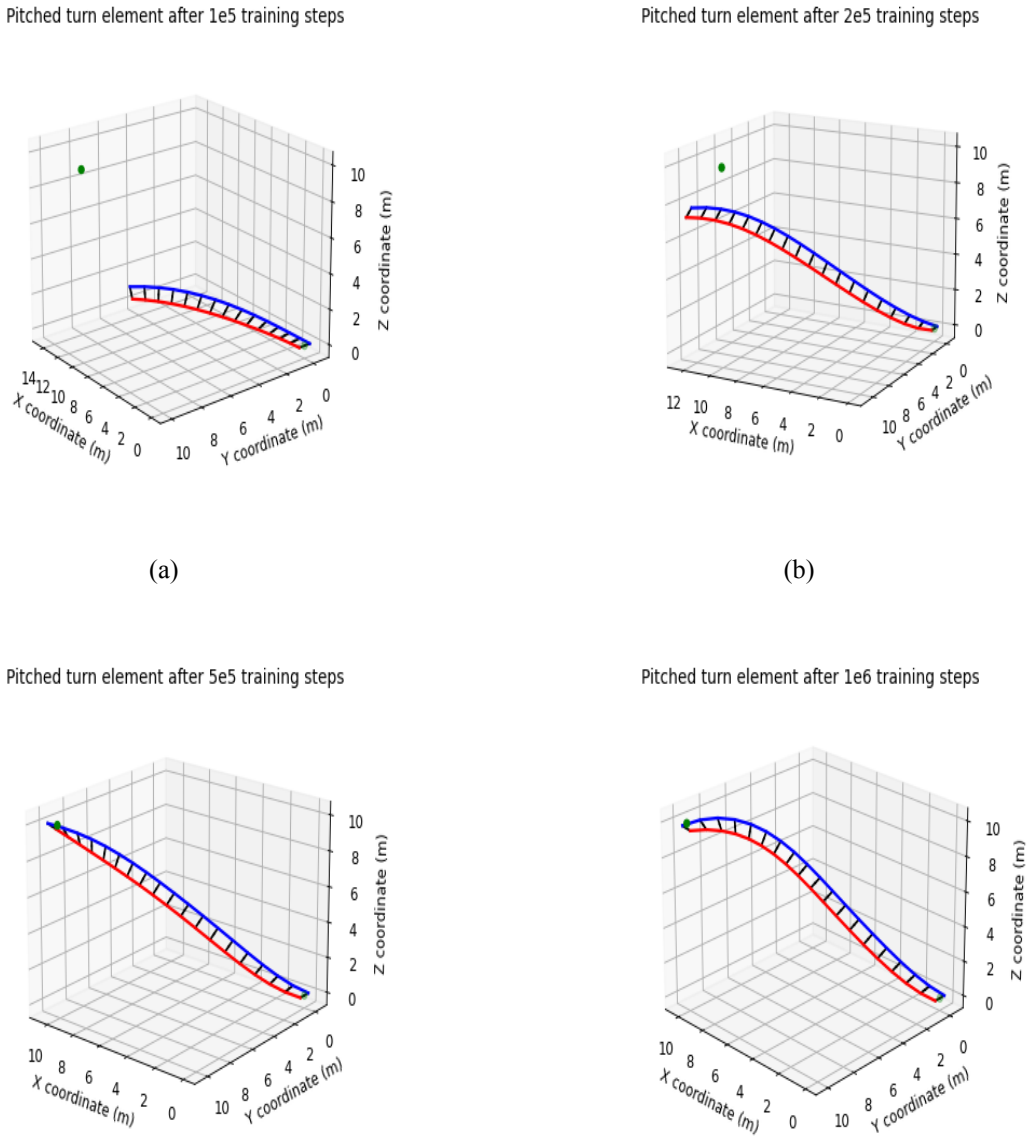
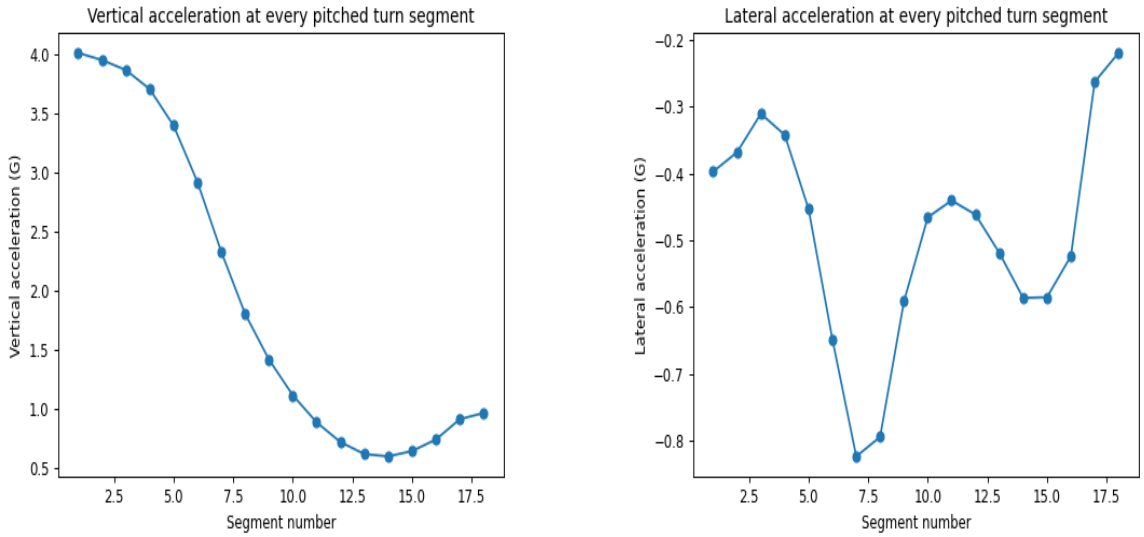


Figure 5.13: Pitched turn results.

.



(a) (b)

Figure 5.14: Dynamic values of the best-generated pitched turn element.

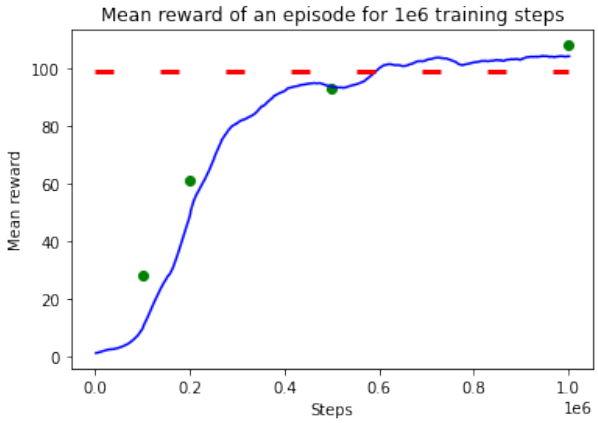


Figure 5.15: Learning curve for the pitched turn element.

.

5.6 Narrow 180 Turn

To push the limits of the points mass model, a narrow 180 degree is attempted. The agent has to control pitch, yaw and roll to guide the point mass into a safe trajectory.

- Start $= [0, 0, 0, 0, 0, 0]$(X Y Z P W R), End goal $= [0, -5, 0, 0, 180, 0]$(X Y Z P W R)
- $L_{avg} = 1$, $N_{seg} = 50$
- $V_{initial} = 15m/s$
- Action Space $= [\Delta^P, \delta^P, \Delta^W, \delta^W, \Delta^R, L_{seg}]$
- State space $= [X, Y, Z, P, W, R, E, V, t, A_{Vert}, A_{Lat}, j, X_{end}, Y_{end}, Z_{end}, P_{end}, W_{end}, R_{end}, S_{numb}]$
- Rewards are r_{place}, r_{safe}, r_{goal} and r_{dir}
- Geometric boundary at Y is minus six, at Y one, at X minus -1, at X 25 and at Z minus 1.
- Thrill criterion is speed, where +1 is given for every m/s of average speed. This reward is only given if 95% of other rewards are achieved.

The reporting style for this case is different because this case highlights a particular shortcoming of the point mass model. First, the learning curve is plotted in Fig. (5.16), here it looks comparable to the other elements. It took the model 20 million training steps to reach the thrill criterion. The total possible general reward is 285, this model reached a total reward of 269. The sharp increase in learning time can have several explanations. First, the initial conditions or boundary conditions of the environment could be unrealistic for a roller coaster element. Secondly, the PPO algorithm parameters should be altered for faster learning. Although this seems less likely because the learning curve does show a consistent upward trend. Thirdly, environmental parameters could hinder the learning process. Fourthly, an increase in action and state complexity together with a high N_{seg} becomes exponentially harder to solve.

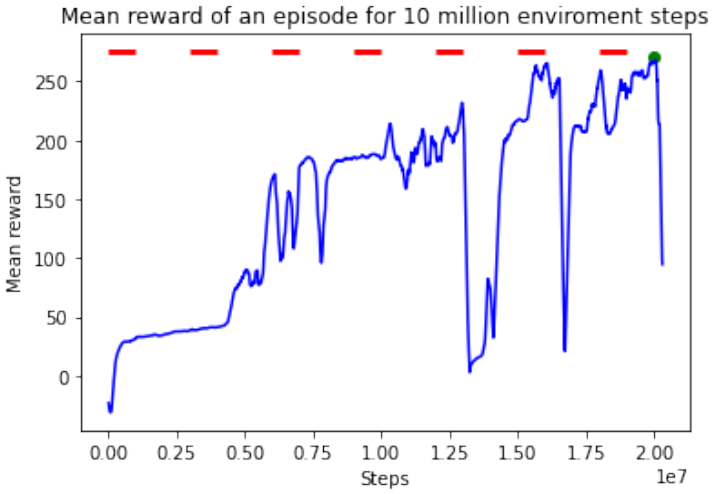
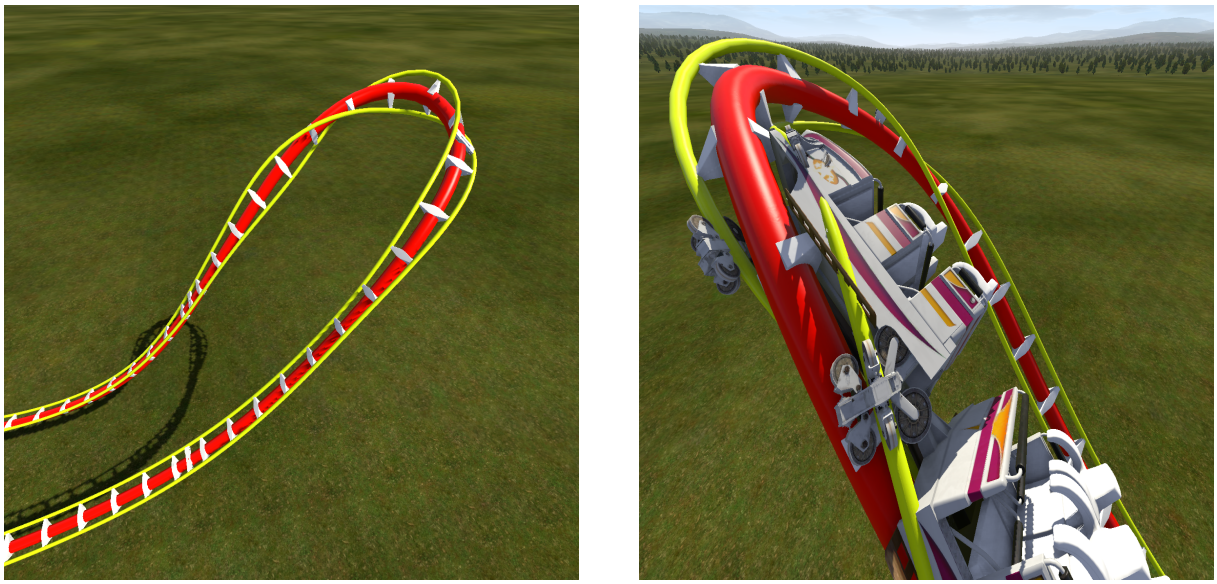


Figure 5.16: Learning curve for the narrow turn element.

For the visualisation of this element, the spline is exported to the program Nolimits2, since a plot makes it difficult to follow the trajectory. The exporting to NoLimits is reported upon in appendix A.4. The final result is plotted in Fig. (5.17a). The track starts on the upper track. The heartline starts with a little roll anti-clockward, then while going steep uphill it turns quickly clockward. The turn is made on the outside of the bend, this is made possible due to the minimal speed of the roller coaster at that point. Finally, the heartline converts back to a straight track ending at the desired endpoint.

This result highlights the flaws of the currently defined point mass. The tight turns and twists of the heartline are impossible to perform if the cart has actual dimensions. Nolimits 2 allows the insertion of a train, then the mechanical infeasibility becomes visible. In Fig. (5.17b) the cart is put on the upper part of the turn. To be able to make the turn the program lets the train go through the track which is obviously impossible in real-life.



(a) (b)

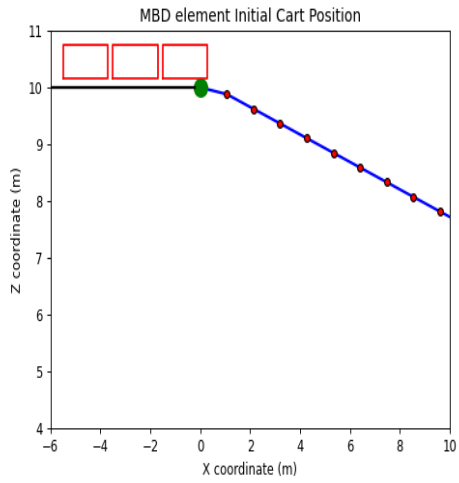
Figure 5.17: 3D render of designed element and roller coaster train.

5.7 Multi-body Drop

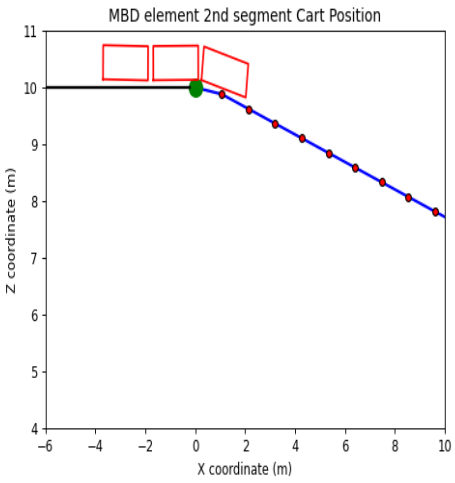
The surrogate model is used in the design of a drop element. The initial conditions for the environment are:

- Start $= [0, 10, 0]$(X Z P), End goal $= [20, 0, 0]$(X Z P)
- $L_{avg} = 1$, $N_{seg} = 25$
- $V_{initial} = 3m/s$
- Action Space $= [\Delta^P, \delta^P]$
- State space $= [X\ Z\ P\ E\ v\ t\ X_{goal}\ Z_{goal}\ P_{goal}\ S_{numb}\ ]$
- State space $= [CoM_x^1\ CoM_y^1\ CoM_\theta^1\ CoM_x^2\ CoM_y^2\ CoM_\theta^2\ CoM_x^3\ CoM_y^3\ CoM_\theta^3]$
- Rewards are r_{place}, r_{goal} and r_{dir}
- Geometric boundary at Z is minus one and at X is minus one.
- No thrill criteria is given

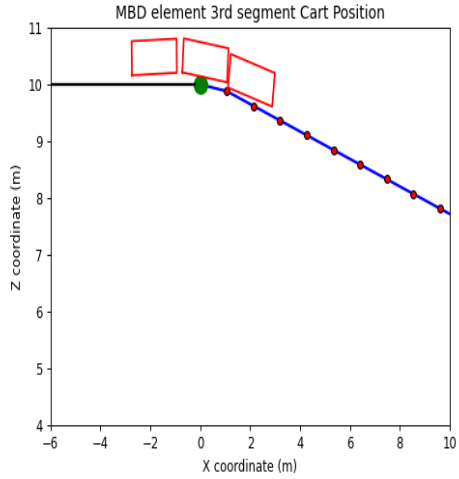
The results for the cart positions are seen in Fig. (4.7b). This is after 50 000 training steps. A simple straight track is created by the agent. The cart is evaluated at the start of six different segments to check the performance of the surrogate model. In segment one and segment two, Fig. (5.18a) and Fig. (5.18b) the cart CoM seems to be in a valid position. However, from segment three onwards Fig. (5.18c) the performance of the surrogate model quickly deteriorates. The cart positions spiral out of control in Fig. (5.18d), Fig. (5.18e) and Fig. (5.18f). This is due to the propagation of error from the predictions made by the surrogate model. From this, it is concluded that the surrogate model performance is inadequate for use in an RL setting. However, the implementation and method behind the surrogate do work.



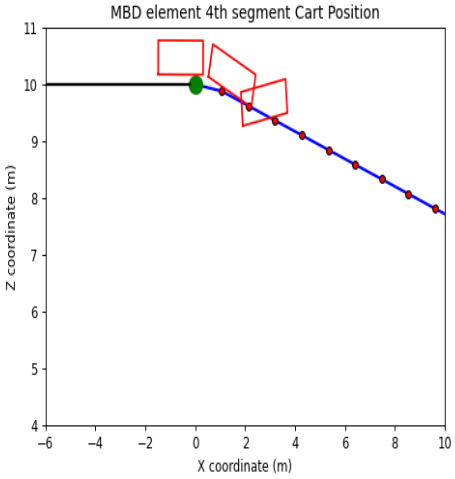
(a)



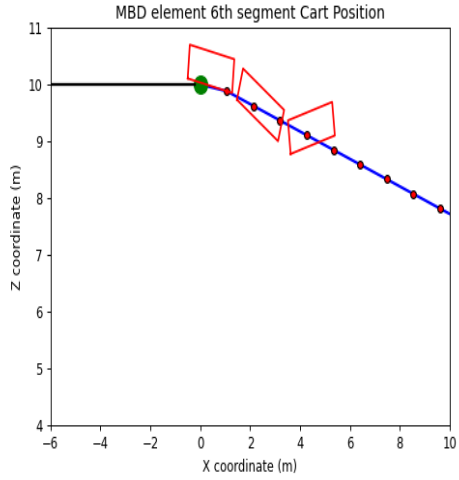
(b)



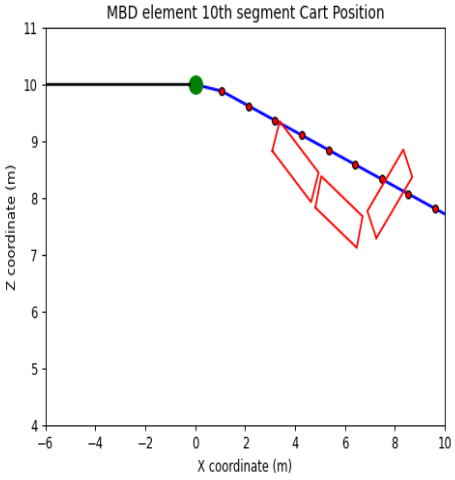
(c)



(d)



(e)



(f)

Figure 5.18: Locations of the roller coaster train in an element.

Chapter 6

Conclusion

The outcomes of this research have provided insight into the application of the reinforcement learning method in roller coaster element design. In this chapter, the research is concluded and discussed. The chapter begins with a review of the research objectives. Then, the considerations and assumptions taken and their implied limitations are discussed. Afterwards, a statement on the more general application is given. The chapter finalises with recommendations for further research in RL-based design optimisation.

The main topic in this thesis is the translation of the roller coaster design problem into a reinforcement learning problem. The proposed custom environment and learning algorithm have succeeded in the case study by designing a drop, a hill, a banked turn, a 2D looping and a pitched turn. The approach optimised elements with inputs, thrill criterion and boundary conditions. The narrow 180-degree turn did adhere to its boundary conditions, but the results were considered mechanically infeasible via NoLimits 2. The secondary topic is the development of a surrogate model for cart geometry and its implementation in an RL environment. Here the implementation of the surrogate model is successful and increased computational efficiency by roughly 250 times, but the performance of the surrogate model is lacking. The propagation of error at each RL time step is too much, quickly resulting in obviously invalid solutions. The applied method does provide promise in integrating an MBD in the RL model using a surrogate model. Insightful RL is obtained by plotting the intermediate stages showing how the actions of the model develop over time.

Although the objectives of this research are met, the case study results are to be viewed with care. In the case study, inputs were defined with little knowledge of actual roller coaster requirements or dimensions. For feasible outputs often several tries were needed. Especially the reduction of the term thrill into a single thrill criterion is disputable and likely needs adaptation. Better inputs and a more realistic thrill criterion are best obtained through cooperation with a roller coaster manufacturer.

Another discussion point is the currently chosen values for custom environment parameters. These values are found to be significant for the performance however fairly little experimentation was done on these parameters. Parameters of the custom environment were initialised at a certain value and were only investigated by changing them one by one, such as L_{avg} in Fig. (4.3). A single parameter change always deterred model performance or resulted in invalid elements. This could be caused by a bias towards these settings. The bias is introduced because of the shaping of states, rewards and actions around the initial parameter values. The bias can be reduced by giving more control of the model over to the agent model. This can be achieved by widening parameter ranges or introducing new actions such as is done with the sixth action for L_{seg} in Eq. (4.3).

An important assumption for the implementation is that the coaster can take on any curvature. It allowed faster development of the current approach proved to be insufficient for the 3D point mass model. This is because a roller coaster engineer has spatial awareness about the roller coaster train. The RL agent does

not have any knowledge that has not been told, thus it assumes any type of turn or twist can be made. This is made visible by the narrow 180-degree turn in Fig. (5.17). The surrogate model helps to circumvent this problem, but its quality is limited.

The limited quality of the surrogate model has several reasons. First, there is too little data generated for supervised learning since the validation set did not fully converge. Second, the input tracks are obtained by the point mass RL model, such that the path i,s dependent on the set of actions. Due to this, the variety in the data remained small. This can be overcome by generating tracks through random walks. Further, the input tracks all considered the same hill element with the same initial position making it harder for the surrogate to fully learn the mapping of the equations. Lastly, only a simple FNN is used to represent the kinematics, a more sophisticated supervised learning network can be applied once more data is gathered.

Despite these shortcomings in the current results of the custom environment, they do not take away from the feasibility of the RL method. The shortcomings mainly highlight that the current considerations taken are not sufficient for the roller coaster element design in 3D. The actual method itself, which is solving the problem with RL works well. The results show that the model is able to adhere to a dynamically changing roller coaster environment. However, in the proposed method, the roller coaster design experience is slightly reduced but replaced by know-how in an RL environment. The RL knowledge required mainly relies on how the engineer wants to implement thrill, geometric boundaries and safety. Knowledge about the solving algorithm is less necessary since PPO proved to be a robust algorithm for this environment, as no changes to the default parameters were required for adequate performance. It remains a question whether the application of RL is necessary for the design of rollercoasters since the application is fairly niche.

The more general application of RL in structural design is also evident as long as a problem can be defined as sequential. A few things should be kept in mind for other applications. A good definition of an MDP needs not only RL knowledge but also extensive domain knowledge. Furthermore, implementation and reward shaping are needed but considered time-intensive. It remains debatable whether such a time investment is viable. However, when well-defined, RL could apply to a range of design problems. To do this more efficiently and more effectively, more research is needed. Finally, the robustness of the PPO algorithm can help in creating effective implementations and custom environments.

This research advises three new directions for future research. The first recommended direction is specifically building upon this research, as the model should be tested in the design of actual roller coaster elements, preferably in cooperation with a roller coaster manufacturer. This would largely increase the validity and spotlight the shortcomings of the given method. The initial conditions and current custom environment parameters could be evaluated such that even with a point mass a more realistic 3D design can be made. For instance by reducing the maximum change in pitch, yaw or roll such that the curvature of the track is minimised. The second direction is the improvement of the surrogate modelling approach for a better implementation of the MBD in RL. This would also benefit other novel applications. Regarding the currently presented surrogate modelling approach mainly the collecting of input data which are the tracks in Fig. (4.7a) should be revisited. The third direction would be the search for new applications in engineering design and the creation of a framework for identifying the MDP. This a challenging recommendation but one that is needed to provide a solid place for RL in design optimisation. Overall the automatic RL design is perceived as a promising tool for the roller coaster application where a further focus can be on the achievement of mechanical feasibility.

Bibliography

[1] R. B. Nye, "Eight ways of looking at an amusement park," *The Journal of Popular Culture*, vol. 15, pp. 63–75, 6 1981.

[2] Z. Kruczek, "Amusement parks as flagship tourist attractions. development and globalization," *Economic Review of Tourism. Faculty of Economics, Matej Bela Univeristy, University*, no. 3, 2011.

[3] R. Stephens, "The psychology of roller coasters." Scientific American, 7 2018. [Date accessed 17-11-2022].

[4] D. Eager, "G-force and enjoyment of rides," *Australasian Parks and Leisure Journal*, p. 32, 9 2013.

[5] Z. Zhang, K. Xiaohan, M. N. A. Khalid, and H. Iida, "Bridging ride and play comfort," *Information 2021*, vol. 12, p. 119, 3 2021.

[6] K. Woodcock, "Global incidence of theme park and amusement ride accidents," *Safety Science*, vol. 113, pp. 171–179, 3 2019.

[7] A. R. Pelletier and J. Gilchrist, "Roller coaster related fatalities, united states, 1994–2004," *Injury Prevention*, vol. 11, no. 5, pp. 309–312, 2005.

[8] Coasterman1234, "Corkscrew at cedar point." https://coasterpedia.net/wiki/Corkscrew_(element). [Date accessed 5-8-2022].

[9] Vekoma, "Notes on designing roller coasters." https://www.vekoma.com/public/files-/notes_designing_roller_coasters.pdf. Date accessed 14-11-2022.

[10] K. Hunt, "Design analylsis of roller coasters," Master's thesis, Worcester Polytechnic Institute, Worcester, 2018.

[11] A. Väisänen, "Design of roller coasters," Master's thesis, Aalto University, Espoo, 2018.

[12] J. R. R. A. Martins and A. Ning, *Engineering Design Optimization*. Cambridge: Cambridge University Press, 2021.

[13] T. I. de Paula, G. F. Gomes, J. H. de Freitas Gomes, and A. P. de Paiva, "A mixture design of experiments approach for genetic algorithm tuning applied to multi-objective optimization," *Advances in Intelligent Systems and Computing*, vol. 991, pp. 600–610, 2020.

[14] A. R. Parkinson, R. J. Balling, and J. D. Hendengren, *Optimization Methods for Engineering Design*. Provo: Brigham Young University, 2013.

[15] J. L. J. Pereira, G. A. Oliver, M. B. Francisco, S. S. Cunha, and G. F. Gomes, "A review of multi-objective optimization: Methods and algorithms in mechanical engineering problems," *Archives of Computational Methods in Engineering*, vol. 29, 6 2022.

[16] T. Schlieter and A. Długosz, "Structural optimization of aerofoils for many criteria," *Engineering Mechanics 2020*, pp. 448–451, 01 2020.

[17] Y. Li, K. Wei, W. Yang, and Q. Wang, "Improving wind turbine blade based on multi-objective particle swarm optimization," *Renewable Energy*, vol. 161, pp. 525–542, 12 2020.

[18] S. Ebrahimi-Nejad, M. Kheybari, and S. V. N. Borujerd, "Multi-objective optimization of a sports car suspension system using simplified quarter-car models," *Mechanics and Industry*, vol. 21, pp. 412–424, 6 2020.

[19] L. Abualigah, M. Elsayed Abd Elaziz, A. Khasawneh, *et al.*, "Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: a comprehensive survey, applications, comparative analysis, and results," *Neural Computing and Applications*, vol. 34, pp. 4081–4110, 3 2022.

[20] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multi-objective optimization," *IEEE Transactions on Cybernetics*, vol. 51, pp. 3103–3114, 6 2019.

[21] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature 2016 529:7587*, vol. 529, pp. 484–489, 1 2016.

[22] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature 2019 575:7782*, vol. 575, pp. 350–354, 10 2019.

[23] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature 2021 596:7873*, vol. 596, pp. 583–589, 7 2021.

[24] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers and Operations Research*, vol. 134, p. 105400, 10 2021.

[25] J. C. Vargas, M. Bhoite, and A. B. Farimani, "Creativity in robot manipulation with deep reinforcement learning," 10 2019.

[26] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," *Artificial Intelligence Review*, vol. 55, pp. 945–990, 2 2022.

[27] J. C. Vargas, M. Bhoite, and A. B. Farimani, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics 2021*, vol. 10, pp. 22–38, 1 2021.

[28] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem, "A review on deep reinforcement learning for fluid mechanics," *Computers and Fluids*, vol. 225, p. 104973, 7 2021.

[29] H. T. Thai, "Machine learning for structural engineering: A state-of-the-art review," *Structures*, vol. 38, pp. 448–491, 4 2022.

[30] M. I. Radaideh, I. Wolverton, J. Joseph, J. J. Tusar, U. Otgonbaatar, N. Roy, B. Forget, and K. Shirvan, "Physics-informed reinforcement learning optimization of nuclear assembly design," *Nuclear Engineering and Design*, vol. 372, p. 110966, 2 2021.

[31] N. M. Sukri, S. M. Nor-Al-Din, N. K. Razali, and M. A. A. Mazwin, "Designing roller coaster loop's by using extended uniform cubic b-spline," *IOP Conference Series: Materials Science and Engineering*, vol. 1176, p. 012039, 8 2021.

[32] G. C. Duran and M. S. G. Tsuzuki, "Procedural method for finding roller coaster rails centerlines based on heart-line acceleration criteria," *2018 13th IEEE International Conference on Industry Applications, INDUSCON 2018 - Proceedings*, pp. 1341–1348, 1 2019.

[33] O. Lange, "Nolimits 2 roller coaster simulation." Mad Data GmbH Co. KG *Steam*, 7 2011.

[34] J. Tsai and E. Lee, "Developing a functional roller coaster optimizer." Brigham Young University, 4 2017.

[35] C. Braccesi, F. Cianetti, and L. Landi, "Integrated Roller Coaster Design Environment: Dynamic and Structural Vehicle Analysis," vol. 4B: Dynamics, Vibration, and Control of *ASME International Mechanical Engineering Congress and Exposition*, 11 2015.

[36] J. Ambrosio, M. Viegas, P. Antunes, and H. Magalhães, "Multibody dynamic modelling and analysis of roller coaster vehicles," The 5th Joint International Conference on Multibody System Dynamics, (Lisbon), IDMEC, Instituto Superior Técnico, Universidade de Lisboa, 6 2018.

[37] R. L. J. Mekers, "Structural dynamics of roller coaster structures: transient analysis," Master's thesis, University of Twente, Enschede, 1 2020.

[38] H. S. Choi, J. An, S. Han, J. G. Kim, J. Y. Jung, J. Choi, G. Orzechowski, A. Mikkola, and J. H. Choi, "Data-driven simulation for general-purpose multibody dynamics using deep neural networks," *Multibody System Dynamics*, vol. 51, pp. 419–454, 4 2021.

[39] Y. Pan, X. Nie, Z. Li, and S. Gu, "Data-driven vehicle modeling of longitudinal dynamics based on a multibody model and deep neural networks," *Measurement*, vol. 180, p. 109541, 8 2021.

[40] O. Pantalé, P. T. Mha, and A. Tongne, "Efficient implementation of non-linear flow law using neural network into the abaqus explicit fem code," *Finite Elements in Analysis and Design*, vol. 198, p. 103647, 1 2022.

[41] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, pp. 132–150, 2 1995.

[42] Y. Ye, P. Huang, Y. Sun, and D. Shi, "Mbsnet: A deep learning model for multibody dynamics simulation and its application to a vehicle-track system," *Mechanical Systems and Signal Processing*, vol. 157, 8 2021.

[43] A. M. Pendrill and D. Eager, "Velocity, acceleration, jerk, snap and vibration: forces in our bodies during a roller coaster ride," *Physics Education*, vol. 55, p. 065012, 9 2020.

[44] "The golden ticket awards 2021." Amusement Today - https://goldenticketawards.com/, 2021.

[45] M. Sigler, "Cubic bézier curve with four control points." https://commons.wikimedia.org/wiki/File:Bezier_curve.svg, 2006. [Date accessed 27-06-2022].

[46] Automaticaddison, "Yaw, pitch, and roll diagrams using 2d coordinate systems." https://automaticaddison.com/yaw-pitch-and-roll-diagrams-using-2d-coordinate-systems/, 2020. [Date accessed 02-06-2022].

[47] P. E. Nikravesh, *An Overview of Several Formulations for Multibody Dynamics*, pp. 189–226. Dordrecht: Springer Netherlands, 2005.

[48] D. Svozil, V. Kvasnička, and J. Pospíchal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 39, pp. 43–62, 11 1997.

[49] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural Networks for Perception*, pp. 65–93, 1 1992.

[50] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning: MSE*, pp. 425–428. Boston, MA: Springer US, 2010.

[51] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. MIT Press,Cambridge, second edition ed., 2018.

[52] B. Balaji, S. Mallya, S. Genc, *et al.*, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *CoRR*, vol. abs/1911.01562, 11 2019.

[53] R. J. Willia, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[54] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," ICLR 2016, 6 2015.

[55] J. Schulman, F. Wolski, P. Dhariwal, *et al.*, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 7 2017.

[56] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," *CoRR*, vol. abs/1901.10500, 2019.

[57] OpenAI, "Gym documentation." https://www.gymlibrary.ml/, 2022. [Date accessed: 21-11-2022].

[58] L. Engstrom, A. Ilyas, S. Santurkar, *et al.*, "Implementation matters in deep policy gradients: A case study on ppo and trpo," *CoRR*, vol. abs/2005.12729, 5 2020.

[59] "Stable-baselines3 docs - reliable reinforcement learning implementations — stable baselines3 1.6.1 documentation." Community project - https://stable-baselines3.readthedocs.io/en/master/, 2022. [Date accessed: 21-11-2022].

[60] X. Zhang, *Encyclopedia of Machine Learning: Gaussian Distribution*, pp. 425–428. Boston, MA: Springer US, 2010.

[61] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

Appendix A

A.1 Full set of MBD equations

The full set of constraint equations is presented in this appendix. There are a total of 14 equations each of which is shortly described here. The first two equations describe the relation of wheel one with respect to wheel two and the angle of cart one. Equations three and four describe the relation between wheel two and the first linking bar. The fifth and sixth equations fix the relationship between the first linking bar and wheel three. The seventh and eighth equations fix the relationship between wheel three, wheel four and the angle of cart two. Equations nine and ten describe the relation between wheel four and the second linking bar. The eleventh and twelfth equations fix the relationship between the second linking bar and wheel five. Lastly, equations thirteen and fourteen describe the relationship between the fifth wheel, the sixth wheel and the angle of cart three.

$$
\begin{aligned}
B_x^{q_3} - B_x^{q_1} - L_2\cos(q_2) &= 0 \\
B_z^{q_3} - B_z^{q_1} - L_2\sin(q_2) &= 0 \\
q_4 + \frac{1}{2}L_5\cos(q_6) - B_x^{q_3} - L_2\cos(q_2) - L_3\sin(q_2) &= 0 \\
q_5 + \frac{1}{2}L_5\sin(q_6) - B_z^{q_3} - L_2\sin(q_2) + L_3\cos(q_2) &= 0 \\
q_4 - \frac{1}{2}L_5\cos(q_6) - B_x^{q_7} + L_2\cos(q_8) - L_3\sin(q_8) &= 0 \\
q_5 - \frac{1}{2}L_5\sin(q_6) - B_z^{q_7} + L_2\sin(q_8) + L_3\cos(q_8) &= 0 \\
B_x^{q_9} - B_x^{q_7} - L_1\cos(q_8) &= 0 \\
B_z^{q_9} - B_z^{q_7} - L_1\sin(q_8) &= 0 \\
q_{10} + \frac{1}{2}L_5\cos(q_{12}) - B_x^{q_9} - L_2\cos(q_8) - L_3\sin(q_8) &= 0 \\
q_{11} + \frac{1}{2}L_5\sin(q_{12}) - B_z^{q_9} - L_2\sin(q_8) + L_3\cos(q_8) &= 0 \\
q_{10} - \frac{1}{2}L_5\cos(q_{12}) - B_x^{q_{13}} + L_7\cos(q_{14}) - L_5\sin(q_{14}) &= 0 \\
q_{11} - \frac{1}{2}L_3\sin(q_{12}) - B_z^{q_{13}} + L_7\sin(q_{14}) + L_5\cos(q_{14}) &= 0 \\
B_x^{q_{153}} - B_x^{q_{13}} - L_1\cos(q_{14}) &= 0 \\
B_z^{q_{15}} - B_z^{q_{13}} - L_1\sin(q_{14}) &= 0
\end{aligned}
\tag{A.1}
$$

A.2 Default model settings

The default parameters used in the case study are presented in this appendix. The left three columns are for the PPO algorithm, the right three columns are for the settings of the custom environment.

Learning Parameter	Symbol	Default Value	**Environment Parameter**	Symbol	Default Value
Learning Rate	λ	$3e-4$	Segment interpolation points	N_{int}	20
Rollout size	n_{steps}	2048	Max Vertical acc		5 G
Batch size	n_{batch}	32	Min Vertical acc		-1 G
Epochs	n_{epochs}	10	Max abs Lateral acc		1.5 G
Discount factor	γ	0.99	Max abs Jerk		15 G/s
GAE discount	γ^{GAE}	10	Max pitch change	Δ_{MaxP}	30
PPO clip range	ϵ	0.2	Max yaw change	Δ_{MaxW}	30
Policy network depth		2	Max roll change	Δ_{MaxR}	45
Policy network width		64	Adjustment Parameter	C	1.8
Value network depth		2	Length std	δ_{seg}	0.1
Value network width		64	Force limit	F_c^m	3
Activation Function		tanh	Geomtric limit	G_c^m	3
Optimiser		Adam			

A.3 Simulation specifications

The simulations were run with visual studio code in a Jupyter notebook. Stable baseline 3 allows for computation through the CPU or GPU. In this thesis, the CPU is used since it proved faster in simulation. If parallel environments are applied for the simultaneous training of multiple agents then the GPU is likely faster. The hardware and software on the personal laptop are:

Hard/Software	Symbol
CPU	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s)
GPU	NVIDIA GeForce RTX 3070 Laptop GPU 8GB
RAM	16 GB
Harddrive	1TB SSD
Operating System	Microsoft Windows 10 Home 10.0.19044
Code language	Python 3.9.13
Code editor	Visual studio code 17.1.1
Code organiser	Jupyter Notebook 5.1
MBD solver	Matlab R2021a

A.4 Nolimits Export and cart creation

In order to export a layout of an element to NoLimits 2 a comma-separated value (csv) file has to be created. The columns of the csv file have to be in the order of node number, x coordinate, y coordinate, z coordinate, three values of the tangent vector, three values of the lateral vector and three values of the normal vector. The coordinate axis is the same as in Fig. (2.2) except the y minus direction is the y plus direction. The csv file is directly imported into the editor of NoLimits 2, offering the option for a centre line-based track or a heartline-based track. For the results in Fig. (5.17a) the centre-based track is chosen, since a heartline-based track could not accurately represent the specified geometry. This was due to the tight curves designed by the RL model. To generate a roller coaster train onto the track, a half circle is added by hand to the NoLimits 2 model import visible in Fig. (A.1). This track featured a break section and a booster section such that the roller coaster makes continuous laps.

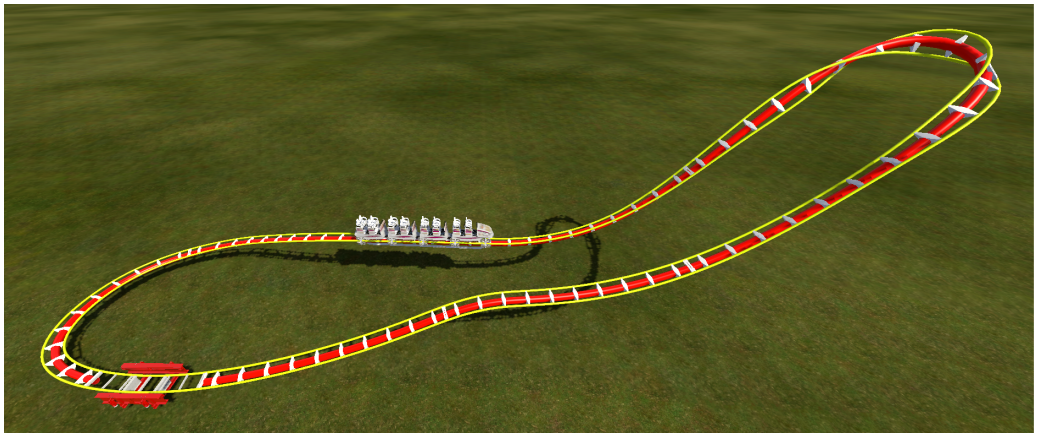


Figure A.1: Full track used to determine train position