Applied Mechanics and Data Analysis (AMDA)

Universal Robot Policy

Using a surrogate model in combination with TRPO

Joren Erens
Msc Thesis
May 2024

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Controllers for robots are generally designed for only one particular robot morphology and one objective. However, a controller can also be trained for multiple robots or objectives. Such a policy is known as a *universal policy.*

In this thesis, the foundation is laid for the design of a universal policy for a locomotion task. That is done by considering one robot, consisting of a number of bodies with unknown density and length. The robot is part of the RoboGrammar design space. The policy is trained with reinforcement learning, using the Trust Region Policy Optimisation (TRPO) algorithm. The dynamics of the robot are modelled using generalised Polynomial Chaos Expansion (PCE) and a model ensemble to investigate whether there are advantages to using a surrogate model instead of the real environment, when training the controller.

Results show that the dynamics cannot be accurately modelled yet with PCE, but that the method is promising. A more practical problem with the PCE algorithm is the computational time required. It was not used in combination with TRPO for both reasons. The model ensemble surrogate was implemented in combination with TRPO, but failed to train a successful policy. However, using the original environment from the RoboGrammar library, instead of a surrogate model, showed promising results for a universal policy. In future research, this work can be extended to a larger variety of robots, that not only have unknown density and length, but also an unknown shape and number of bodies.

Contents

Nomenclature

List of symbols:

- A: advantage function [-]
- a: acceleration, subscript indicates the direction $[\frac{m}{s^2}]$
- α: polynomial index for PCE [-]
- c_t: polynomial coefficient [-]
- $\mathbb{D}$: set with collected samples [-]
- D_{KL}: kullback-leibler divergence [-]
- Δ: diagonal Gram matrix [-]
- δ: maximum allowed KL divergence [-]
- δ_{estimate}: upper limit for the accuracy of the PCE model [-]
- $e(t)$: tracking error of the PD controller [-]
- f: force $[N]$
- g: implementation function surrogate advantage TRPO [-]
- γ: discount factor [-]
- H: implementation function KL-divergence [-]
- $\boldsymbol{I}$: inertia matrix $[kgm^2]$
- i: link index [-]
- J: objective function [-]
- $\mathscr{J}$: multi-index for PCE [-]
- K_D: differential gain of the PD controller [-]
- K_P: proportional gain of the PD controller [-]
- L: surrogate advantage [-]
- l: length of a link $[m]$
- λ: parameters of the value function [-]
- M: model from model ensemble [-]
- μ: parameters of the policy [-]
- N: number of sampled trajectories [-]
- N_{ME}: number of sets of coefficients for multi-element [-]
- $N_{\text{model updates}}$: number of model updates in a training cycle [-]
- N_s: number of collected samples [-]
- n: number of links in a robot [-]
- ν: learning rate [-]
- P, transition function [-]
- $\boldsymbol{p}$: bias forces $[N]$
- π_μ: reinforcement learning policy [-]
- ϕ: pitch angle of the link $[rad]$
- Ψ: polynomial [-]
- ψ: yaw angle of the link $[rad]$
- Q: quality function [-]
- r: reward function [-]
- ρ: density of a link $[\frac{kg}{m^3}]$
- s: state (vector of position, velocity and acceleration) [-]
- $\mathbb{S}$: set with collected states [-]
- T: action window size for PCE [-]
- T_{joint}: time interval for joint actions [-]
- t: time-index [-]
- $\mathbb{T}$: trajectory [-]
- τ: torque $[Nm]$
- $\Theta(\omega)$: random variable [-]
- θ: roll angle of the link $[rad]$
- u: action [-]
- V: value function [-]
- v: velocity, subscript indicates the direction $[\frac{m}{s}]$
- w: weight vector for reward function [-]
- x: position in x direction $[m]$
- y: position in y direction $[m]$
- z: position in z direction $[m]$

List of acronyms:

- ABA: articulated body algorithm
- DE: deterministic environment
- JSD: Jenson-Shannon divergence
- KL: Kullback-Leibler
- ME: multi-element
- MOE: model ensemble
- NN: neural network
- PCE: polynomial chaos expansion
- RL: reinforcement learning
- SE: stochastic environment
- TRPO: trust region policy optimisation

1 Introduction

Controlling the dynamics of a real system is a difficult task: there is almost no real system, the behaviour of which is not affected by uncertainties and disturbances. Because of that, a controller must be robust to the stochastic (uncertain) behaviour in the system. Such a controller is known as a stochastic controller or a stochastic policy [1]. The field of stochastic control is concerned with finding the optimal controller for a system with stochastic parameters. When the controlled robot also has a varying number of actuators or a varying configuration, the stochastic controller is known as a *universal controller* [2]. A universal controller can control robots of various shapes without the need to adjust policy parameters when the robot changes in morphology.

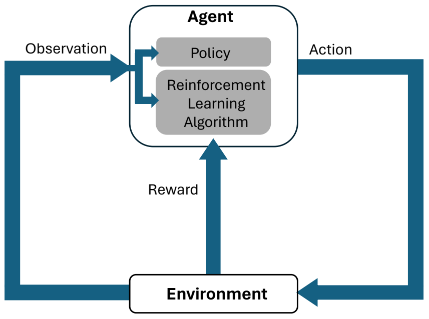


Figure 1.1: Reinforcement learning cycle, from [3].

The universal controller is trained with reinforcement learning (RL), which is already proven to successfully train agents for a large variety of environments [4]. In this thesis, the environment is a robot moving on a flat plane. With RL, an agent interacts with the environment and obtains data through those interactions. This cycle is visualised in Figure 1.1. The agent consists of a policy and a reinforcement learning algorithm. The algorithm is used to update the policy. The policy in this work is parameterized by a neural network. The agent always has an objective, which is quantified by a reward function. The environment returns a reward when the agent executes an action. Besides a reward, the agent also measures the position, velocity and acceleration of the robot (in the figure labelled 'observation'). With all that data, the reinforcement learning algorithm decides whether an action is good or bad, with the aim of improving the policy so that good actions are taken more often. The goal is to maximise the cumulative reward of one episode.

Conventionally, RL is used to train a policy for one robot and one task (a number of examples are given in [5] and [6]), but there is no reason why this could not be expanded for a controller for various robot morphologies [2, 7]. RL is a very intuitive choice for training a policy, since it allows the agent to 'learn from experience' and does not restrict what environments are used or what objectives: it is a very versatile algorithm. A second reason is that RL does not require a model of the environment, which can be difficult to derive for complex environments. Furthermore, the policy learns the relevant characteristics of the environment without the user needing to specify those in advance. This is especially helpful in a complex environment with a lot of data available to the policy. In such a case, it might not be directly clear what information is relevant for the controller and what is not. The user cannot simply decide a priori what information should be used by the RL algorithm: obtaining a universal policy is not a simple task. When training a controller for a single robot and a single task, the hyperparameters are of significant importance for the results [8]. The effect of hyperparameter tuning increases with the complexity of the environment: when a stochastic environment is considered, that effect is even larger compared to its deterministic counterpart.

RL requires a lot of interactions with the environment, especially when the dynamics are complex or a lot of data is available for the policy. That is one of the disadvantages of RL [9]. Besides the many interactions needed, the policy can also explore configurations that are unsafe or damage the robot. In some cases, there are no limitations to how the environment is used or how many interactions with the environment are possible, but generally, this cannot be assumed. Therefore, a model of the dynamics of the environment is useful. Such a surrogate model can be used in combination with RL to reduce interactions with the environment. There are more reasons to use a surrogate model: it might be possible to obtain data faster and more efficiently sample from the stochastic environment, with the result that the controller is learned faster. Stochastic robot dynamics are not often modelled [10], due to their complexity, so it is also a scientific achievement if an accurate model of the stochastic robot can be obtained.

Universal controllers for robots have been designed before using RL. In [7] a universal policy is trained for various robot morphologies. The robots have the objective of walking forward. The control is centred around modular neural networks, labelled as Shared Modular Policies. The Shared Modular Policies algorithm means that every limb has its own policy, which can send and receive messages from the other limbs and, with that, obtain information about the state of the entire robot. The policies are represented by neural networks, which have the same parameters for every limb. By having a decentralised controller, it is possible to deal with the varying number of states and actuators in a robot. The use of Shared Modular Policies results in an effective universal controller for a wide variety of robots. The method shows consistent behaviour and is a promising foundation for universal control.

A different approach of a universal controller is given by [2]. Robots similar to the previous work are used, but they use a transformer network to calculate an action based on the link states. The transformer is a Morphology Aware Transformer and is able to successfully control various morphologies in various environments. A graph neural network is used to represent the robot. The use of a transformer resulted in successful training of a universal controller, which performed well on a large number of robot morphologies. Although the approach is very different compared to the work described in the previous section, they both can obtain successful universal policies.

Transformers and graph neural networks were also used by [11], which implemented a controller that outperformed existing controllers in many environments and showed the potential of using transformers. The transformers obtained higher rewards and needed shorter training times compared to existing controllers. However, this implementation was found to have a high computational complexity, especially for robots with a higher number of bodies. Another disadvantage of this work is that the design space used was limited compared to [7] and [2].

Universal control is also derived for real world systems, as given by [12]. TRPO and DDPG were used for training the policy using a multi-legged robot. The robot had a varying number of legs. The difficulty of training a universal policy was highlighted in this work: it was found that the stochastic environment resulted in high variance in rewards. That means the policy was not able to always control the robot equally well, but it always managed to obtain a walking motion. Compared to the other mentioned examples of universal control, this work did only use one type of robot, but the algorithm was applied to a real-world situation.

All previously mentioned implementations of universal control did not use a surrogate model, but were sampled directly from the environment. However, it is possible to model a stochastic environment, which has been done a lot before. There are multiple methods for modelling a stochastic environment. One of the simplest is Monte Carlo, which is used in, for example, [13] and [14]. But this approach is known to be converging slowly [15]. A more efficient method is generalised Polynomial Chaos Expansion (PCE) [16] (it is explained in Section 4.1). With PCE, a distribution is represented by a polynomial function of random variables (RVs). PCE is used for a large variety of problems, for example: modelling nonlinear dynamic systems [17, 18], sensitivity analysis [19] or parameter estimation [20]. A dynamic system can be modelled using PCE in two ways: the first one is, for example, implemented by [21]. The PCE models a mass damper system and is based on dynamical equations. The random variables (RV) are described by PCE and those expansions are substituted into the dynamical equation. This way, a stochastic differential equation is obtained, where the stochastic parameters are described by polynomial chaos. A similar approach is done by [22] and [23], which used PCE for various multi-body systems. A different approach is to describe the estimated state with a PCE, not the RV in the environment. This is done by [24], which estimated the states directly with PCE. It did so, successfully, for a relatively simple double pendulum, but was also verified with real-life data from an artillery canon. In [25], the constraint equations for a multi-body system are used to link the generalised coordinates (expressed in terms of PCE) with the coordinates of the system. It is applied to a four-bar link system and the results give an accurate representation and show an accurate prediction of the states.

A different approach of a surrogate model is given by [26]. The environment is modelled by a collection of neural networks. The neural networks predict the dynamics of the (stochastic) environment. The model ensemble is used in combination with RL and is successful in multiple environments from the Gym library.

The research done in this thesis is centred around developing a starting point for training a universal policy for locomotion for a stochastic robot, using a surrogate model. For this purpose, one robot with parameterized morphology is used.

The robot is controlled by a policy, which is obtained through reinforcement learning. The robot, considered in this work, is generated by the RoboGrammar library [27]. The library can generate robots for locomotion tasks. The library is used, because it can generate various robot morphologies, all consisting of the same parts and having similar structures. Therefore, the library contains an ideal environment for a universal controller. More details are given in Section 2. Although the library can generate robots of various morphologies, only one is used in this work. That is done to simplify the problem. The robot is parameterized by variations in design.

The objective of the policy is to learn walking in a forward direction. Although other objectives are also possible, these are not considered in this work. The universal policy is trained with the surrogate model of the stochastic environment. That stochastic environment is modelled using PCE (see Section 4.1), which has been chosen because of the efficiency of the algorithm, the option to describe RVs of various distributions, and the simplicity of sampling data from such a model. The model ensemble is a second type of surrogate model considered in this work. Both models are implemented to explore the advantages and differences between both approaches for a surrogate model.

The algorithm used for RL is a Trust Region Policy Optimisation (TRPO) [28]. This algorithm is explained in Section 3. It is one of the state-of-the-art RL algorithms [29], which can perform well in various environments, including environments similar to the RoboGrammar library [30].

2 The Model Environment

The simulation environment used in this thesis is presented. In particular, the parameterization of the environment, its modelling by the use of probability theory and the surrogate model with RL.

2.1 General Layout of the Robot

The robot considered in this work is part of the vertebrates, meaning that the robot has one body to which legs are connected. The robot is symmetrical along the axis of the body, always having an even number of legs (see Figure 2.1). The robot consists of parts, which can be divided into two categories: links and joints. The links are the connections between the joints and are modelled as a rigid body. The joints are the actuators of the robot. They can be oriented in various directions, enabling relative motion between links. The links can be further divided into body links and leg links. The body links are slightly larger in size compared to the leg links. The first link of the body is also referred to as the *base link* and can be described as the head of the robot. The base link is important to define the direction of forward movement: that direction is always the axis aligned with the body in its initial position. That axis is denoted as the x axis, with a positive direction pointing out of the base link, away from the second link in the body (represented by the blue arrow in Figure 2.1). Note that in the initial configuration, the global reference frame is located in the base link, but that it is fixed to the plane. When the robot moves forward, the global reference frame does not move with the robot, but is fixed at that position. The robot always has links and joints that alternate, except for the link that connects a leg to the main body. In Figure 2.1, the robot used in this thesis can be seen. The grey cylinders are the links and the connections between those (orange and green) are the joints. The orange cube is a fixed joint (no degrees of freedom), which has the function of connecting links without the possibility of relative motion. The orange cylinders are roll joints, which always have the rotating axis aligned with at least one of the neighbouring links. The green cylinder is a knee joint that has its rotating axis perpendicular to the main axis of both neighbouring links. The base link is the most right link in the body.

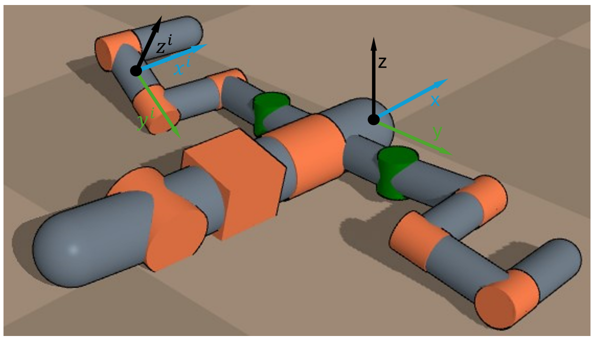


Figure 2.1: Example of a robot from the RoboGrammar design space. The global coordinate frame is given as well. The blue arrow gives the x direction, the green arrow the y direction and the black arrow the z direction. A local coordinate frame has been added is well, indicated by the superscript i.

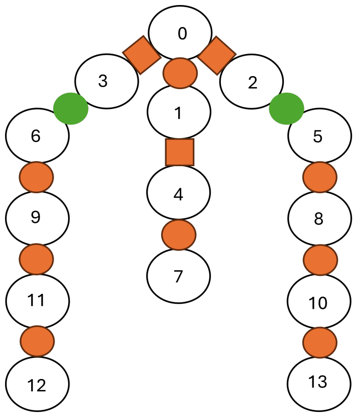


Figure 2.2: Representation of the robot in Figure 2.1 as a kinematic tree. The black circles are the links and the numbers represent the index of the link.

Each robot can be represented as a kinematic tree, in which the base link is the root of the tree. The base link is always labelled as link 0. Every leg is a branch in the kinematic tree. The index of the links increases when they are further away from the base link. The kinematic tree of the robot from Figure 2.1 is drawn in Figure 2.2. The links are represented by the black circles and the joints by the orange or green circles or blocks. The shape of the joints corresponds in both figures.

Every link is modelled as a rigid body in three dimensions, which means that the following states can be calculated per link (position, velocity and acceleration, respectively):

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} \psi \\ \theta \\ \phi \\ x \\ y \\ z \end{bmatrix}, \dot{\boldsymbol{x}}^{(i)} = \boldsymbol{v}^{(i)} = \begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \ddot{\boldsymbol{x}}^{(i)} = \dot{\boldsymbol{v}}^{(i)} = \boldsymbol{a}^{(i)} = \begin{bmatrix} \ddot{\psi} \\ \ddot{\theta} \\ \ddot{\phi} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \tag{2.1}$$

where the superscript i refers to link i in the robot. Each robot consists of n links. In the position vector, the first three indices (ψ, θ and ϕ) give the yaw, roll and pitch angles, which define the orientation of the link. The first time-derivative of the orientation gives the angular velocity and the second gives the angular acceleration. The last three indices in the position vector (x, y and z) are the position of the link in the longitudinal, lateral and vertical directions. All states are given with respect to a global reference frame (that is the coordinate frame given in Figure 2.1). The states of the robot are collected in one vector: $\boldsymbol{s}^{(i)} := \left[(\boldsymbol{x}^{(i)})^T, (\boldsymbol{v}^{(i)})^T, (\boldsymbol{a}^{(i)})^T\right]$.

The states of joints are not included here, but they can be calculated from the difference in movement of neighbouring links. That is possible because the joints are assumed to be rigid, so there is no flexibility. The velocity of a joint is, for example, given by:

$$\boldsymbol{v}_{\text{joint}}^{(i)} = \boldsymbol{v}^{(i)} - \boldsymbol{v}^{(i-1)} \tag{2.2}$$

where the joint is between body i and $i-1$. A similar expression holds for the position and acceleration of a joint. Every joint has only one degree of freedom and thus also has only one actuator.

The robot can move by actuating the joints. If that is done in the correct sequence, the robot is able to walk. The joints are actuated by a motor, which can deliver torque to the joint. There are two methods of controlling the robot. The first is to have one controller that outputs the torques of all joints, based on the current state of the robot. The second method is to separate the control of all the joints combined and the control of the torque delivered by a single joint. In cascaded control [31], the control tasks of a system are separated into multiple controllers. The output of one controller drives another controller, resulting in a hierarchy of controllers. The final controller, or 'lowest' controller calculates the torque of a joint. Such a method of control gives more stability to complex or large systems [32]. This second method is implemented by RoboGrammar [27] and is also used in this work.

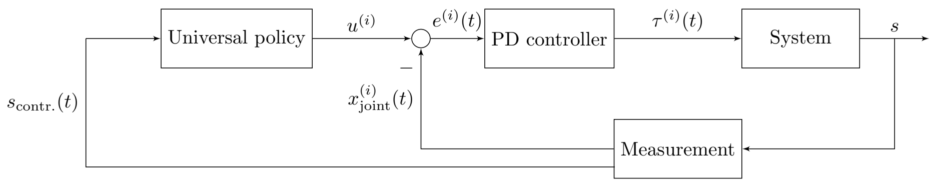


Figure 2.3: Block diagram of the cascaded control loop. The joints are driven by a target position.

In this work, there is a top-level controller, which determines the position or velocity of all joints, and a low-level controller. The feedback loop of the cascaded control is visualised in Figure 2.3. The figure shows the block diagram for one joint, but the robot has multiple joints, so there are multiple parallel low-level control loops. The control loop starts with the universal policy block, which determines the targets for the joints (in this case, joint target positions), labelled as $\boldsymbol{u}$. Those targets are based on the state of the robot. Not all states might be needed to determine the target positions, so only a subset is used, labelled as $\boldsymbol{s}_{\text{contr.}}$. The difference $e^{(i)}(t)$ between the target position for one joint and the measured position is the input for the PD controller:

$$e^{(i)}(t) = u^{(i)} - x_{\text{joint}}^{(i)}(t)$$

where $\boldsymbol{u}^{(i)}$ is calculated by the top-level controller. In this thesis, the reference signal $\boldsymbol{u}^{(i)}$ is a joint position. The resulting torque for the motor actuates the system, which is given by:

$$\tau^{(i)}(t) = K_P e^{(i)}(t) + K_D \frac{d}{dt} e^{(i)}(t)$$

The PD controller gains (K_P for the proportional part and K_D for the derivative part) can be set by the user (default 0.05 and 0.1 respectively). The output of the PD controller is the torque of one joint, so every joint has its own PD controller. The system block (in Figure 2.3) calculates the dynamics of the robot (which are elaborated more in Section 2.2). The states are updated, and they are output of the system block. Finally, the measurement block is added to take into account any transformations needed to the measured signals before they are the state of the robot. The low-level controller calculates the torque for one joint, based on the target (velocity or position) from the top level controller.

The cascaded control requires that the top-level controller is slower compared to the low-level controller; otherwise, the system can become unstable [33]. The top-level control is changed every 0.0625 s (16 Hz), whereas the low-level control (the fast loop) every 0.00417 s (240 Hz). The top-level control frequency has been chosen, such that, generally, in one control interval, the low-level controller is able to reach the target position or velocity. It is not always true, due to the stochastic dynamics of the robot: a heavier leg moves slower and does not reach the control target in the specified time. The interval time for the top-level control is referred to as T_{joints} (since it controls the joints) and the low-level control is referred to as T_{PD}. The simulation time-step at which data is obtained is also 240 Hz, but the algorithm that calculates the kinematics (discussed in the next section), uses sub-steps for sufficient accuracy.

2.2 Physics-based Simulation of the Environment

The movement of the rigid bodies is calculated using the Articulated Body Algorithm (ABA) [34] implemented using the Pybullet library [35]. ABA is an algorithm that calculates the forward dynamics of a multi-body robot. The algorithm can be used for robots of all sizes and is especially efficient for robots with more than nine bodies. This algorithm can be used for vertebrate robots, which includes the RoboGrammar design space. The algorithm is based on the kinematic tree of the robot. The tree is used to calculate the effect of the accelerations and forces of the links in the subtree of link i on the acceleration of link i. The foundation of the ABA algorithm is given by the Newton-Euler equations for rigid bodies [34]:

$$\boldsymbol{f}^{(i)} = (\boldsymbol{I}^{(i)})^A \boldsymbol{a}^{(i)} + (\boldsymbol{p}^{(i)})^A \tag{2.3}$$

The force vector, for three-dimensional bodies, has a dimension of six, and gives the force on body i transmitted to the connected joint. The force results in an acceleration of link i: $\boldsymbol{a}^{(i)}$. The vector $(\boldsymbol{p}^{(i)})^A$ is the constant force on the link (e.g. gravity) and is also known as the bias force. Finally, the rate of acceleration is determined by the inertia matrix $(\boldsymbol{I}^{(i)})^A$. The superscript A (abbreviation of 'articulated') of the bias term and inertia matrix means that it contains the effect of the other joints and links in the robot on the kinematic subtree of link i. The acceleration of the link is the combination of the acceleration of the root link of that link (so the link to which the link i is connected) and the acceleration due to the force on the joint between the links. So, the acceleration is further separated:

$$\boldsymbol{a}^{(i)} = \boldsymbol{a}^{(i-1)} + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)} + \boldsymbol{S}^{(i)}\ddot{q}^{(i)} \tag{2.4}$$

Here, $\boldsymbol{a}^{(i-1)}$ is the known acceleration of the root link. $\dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)}$ and $\boldsymbol{S}^{(i)}\ddot{q}^{(i)}$ are the acceleration due to the joint connected to link i. Here, $\boldsymbol{S}^{(i)}$ is the movement constraint of the joint, also known as the joint motion subspace matrix, and q is the position of the joint. The movement constraint matrix ensures that a joint can only rotate, for example, around the x direction. The first three indices of the joint motion subspace matrix determine the rotation (around the x, y and z axes respectively) and the last three indices determine the translation in x, y and z directions. The constraint matrix is dependent on the orientation of the degree of freedom of the joint. The matrix is expressed in a local coordinate frame and is constant. The local coordinate frame is given in Figure 2.1 by the frame with the superscript i. Every link has such a frame. An example of the motion subspace matrix is a joint that can only rotate around the x axis. It has the following constraint (in three dimensions): $\boldsymbol{S}^{(i)} = [1, 0, 0, 0, 0, 0]$. The torque generated by the actuator in the joint, is transformed into a force vector with:

$$(\boldsymbol{S}^{(i)})^T \boldsymbol{f}^{(i)} = \boldsymbol{\tau}^{(i)} \tag{2.5}$$

These three equations are the basis for ABA. Substituting Equations 2.4 and 2.5 into equation 2.3 gives:

$$(\boldsymbol{S}^{(i)})^T((\boldsymbol{I}^{(i)})^A(\boldsymbol{a}^{(i-1)} + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)} + \boldsymbol{S}^{(i)}\ddot{q}^{(i)}) + (\boldsymbol{p}^{(i)})^A) = \boldsymbol{\tau}^{(i)} \tag{2.6}$$

Every time-step, the acceleration of the joint $\ddot{q}^{(i)}$ is calculated. Once the acceleration of the joint is known, it is possible to iteratively calculate the accelerations for every link in the robot. The expression for the acceleration is:

$$\ddot{q}^{(i)} = ((\boldsymbol{S}^{(i)})^T \boldsymbol{I}^{(i)} \boldsymbol{S}^{(i)})^{-1}(\boldsymbol{\tau}^{(i)} - (\boldsymbol{S}^{(i)})^T \boldsymbol{I}^{(i)}(\boldsymbol{a}^{(i-1)} + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)}) - (\boldsymbol{S}^{(i)})^T(\boldsymbol{p}^{(i)})^A) \tag{2.7}$$

The inertia and bias forces are dependent on the link parameters density and length. Those can be added in Equation 3.13. The final equation for the link acceleration is:

$$\ddot{q}^{(i)} = \left[(\boldsymbol{S}^{(i)})^T(\boldsymbol{I}^{(i)}(l,\rho))^A \boldsymbol{S}^{(i)}\right]^{-1}\left[\boldsymbol{\tau}^{(i)} - (\boldsymbol{S}^{(i)})^T(\boldsymbol{I}^{(i)}(l,\rho))^A(\boldsymbol{a}^{(i-1)} + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)}) - (\boldsymbol{S}^{(i)})^T(\boldsymbol{p}^{(i)}(l,\rho))^A\right] \tag{2.8}$$

The joint acceleration is calculated in three iterations over the body, also named passes. The first pass is from the base link to the roots (end of the legs or tail of the robot) and is used to calculate the bias terms (needed for the bias forces $(\boldsymbol{p}^{(i)})^A$). In the second pass, from the roots to the base link, the articulated inertia $\boldsymbol{I}^{(i)}$ and bias forces $(\boldsymbol{p}^{(i)})^A$ are calculated. Finally, in a third pass from the base link to the roots, the joint acceleration is calculated. After these three passes, the link position and velocity can be calculated using a numerical integration scheme. The time-integration is calculated with Runge-Kutta [36].

The algorithm uses a body-fixed frame, which results in a more efficient computation. When using a body fixed frame, the inertia matrices and bias forces simplify significantly compared to using a global reference frame.

2.3 Design Variation in the Robot

The introduced uncertainty in the robot is the density of the links (ρ) and the length of the links (l). These two variations represent, among others, manufacturing inconsistencies, load variations or design choices. For simplicity, it is chosen that all links in a robot have the same density and length, so within one robot there is no variation in the density or length, but within various robots, there is. The length and density are random variables (RV) of the joint space $(\Omega^l \times \Omega^\rho, \mathcal{F}, \mathbb{P}^l\mathbb{P}^\rho)$ [37], where Ω is the sample space, $\mathcal{F}$ the sigma-algebra of the joint space and $\mathbb{P}$ is the probability measure on $\mathcal{F}$. The length is given by $l(\Theta(\omega))$ and the mass by $\rho(\Theta(\omega))$. The RV represents a mapping $\Omega \to \mathbb{R}$. The assigned distribution of the RVs can be of various types and it is chosen to have both RVs as a uniform distribution:

$$\begin{aligned} l(\Theta(\omega)) &\sim U(l_{\min}, l_{\max}) \\ \rho(\Theta(\omega)) &\sim U(\rho_{\min}, \rho_{\max}) \end{aligned} \tag{2.9}$$

Introducing the equation 2.9 into the equation for the joint acceleration (Equation 2.8), gives:

$$\begin{aligned} \ddot{q}^{(i)}(\Theta(\omega)) = &\left[(\boldsymbol{S}^{(i)})^T(\boldsymbol{I}^{(i)}(l(\Theta(\omega)), \rho(\Theta(\omega)))^A \boldsymbol{S}^{(i)}\right]^{-1} \dots \\ &\left[\boldsymbol{\tau}^{(i)} - (\boldsymbol{S}^{(i)})^T(\boldsymbol{I}^{(i)}(l(\Theta(\omega)), \rho(\Theta(\omega))))^A(\boldsymbol{a}^{(i-1)} + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)}) - (\boldsymbol{S}^{(i)})^T(\boldsymbol{p}^{(i)}(l(\Theta(\omega)), \rho(\Theta(\omega))))^A\right] \end{aligned} \tag{2.10}$$

When torque is applied to a joint, it no longer results in the same joint acceleration every time. That acceleration is now a distribution due to the stochastic density and length. The acceleration of the link is stochastic now:

$$\boldsymbol{a}^{(i)}(\Theta(\omega)) = \boldsymbol{a}^{(i-1)}(\Theta(\omega)) + \dot{\boldsymbol{S}}^{(i)}\dot{q}^{(i)}(\Theta(\omega)) + \boldsymbol{S}^{(i)}\ddot{q}^{(i)}(\Theta(\omega)) \tag{2.11}$$

It is possible to write out the terms in the inertia matrix and bias forces, to directly see how the density and length affect the inertia and bias forces. The previous Equation 2.11 has to be discretized in both time and stochastic domain. There are many ways of achieving this, as is further discussed in Section 4.

The joint acceleration is only dependent on the current action and state of the robot and the subsequent state is described by a probability transition; therefore, the environment is Markovian [38], which is represented by a Markov Decision Process [39]. A Markov Decision Process is defined as the tuple $\langle S, U, P, r\rangle$. S is the finite set of states, U the finite set of actions, P the transition function and r the reward function. The reward function gives the mapping $r : \boldsymbol{s}_t \to \mathbb{R}$ and is defined for this work in Equation 2.12. The transition function gives the probability of a state transition being in state $\boldsymbol{s}_t$:

$$P(\boldsymbol{s}_t, \boldsymbol{u}_t, \boldsymbol{s}_{t+1}) = \mathbb{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{u}_t)$$

A Markov Decision Process is a mathematical framework where the state transitions are due to RVs and control. This is the case for this thesis, where the state transitions are now dependent on the RVs density and length, as well as the control actions from the universal policy.

2.4 Learning the Universal Controller by RL

The Markov Decision Process is the basis for RL. The transition model P and the reward r are used by the RL algorithm to improve the policy π_μ. The policy π_μ in this thesis is parameterized by a neural network (NN) with parameters μ. The set of actions U is the set of possible joint target positions $\boldsymbol{u}$, as defined earlier. The environment and reward are defined in the remainder of this section.

In this thesis, the environment is the robot (as presented in Section 2.1), that moves forward on a flat plane. The latter is solid and infinitely long. So, the robot can never fall through the plane or off the plane. The forward kinematics is modelled using the ABA algorithm (Section 2.2). The states that are collected from the environment are the position, velocity and acceleration of all links. This environment is the transition function T from the Markov Decision Process.

The environment can have various conditions to terminate the episode. In this work, the episode is stopped when a specified number of actions has been executed by the policy. Another option could be, for example, when the robot has reached a certain distance with respect to the reference frame.

The objective of the robot is to move forward, so the reward obtained from the environment is given by:

$$r(\boldsymbol{s}_t) = w_x \cdot x(t) + w_y \cdot |y(t)| + w_{\dot{x}} \cdot \dot{x}(t) \tag{2.12}$$

$x(t)$ is the forward position of the robot, which is multiplied with a weight w_x, which is positive and therefore rewards forward movement. $x(t)$ is measured compared to a global reference frame, so the reward increases if the robot moves in the positive x direction in that reference frame. The robot is placed in the positive x direction, at the beginning of an episode, to simplify the actions that need to be taken (no need to turn at the beginning of the episode). The second term in Equation 2.12 is the sideways movement of the robot. The weight w_y is negative to penalise sideways movement. The term $w_{\dot{x}} \cdot \dot{x}(t)$ is the forward velocity, which is multiplied with a positive weight and it results in the encouragement of faster movement forward. The velocity term also encourages forward movement and is added to emphasise the reward function for fast, forward movement.

The reward (Equation 2.12) is calculated every time-step. The cumulative reward of one episode is the sum of all the rewards from the individual time-steps. With the cumulative reward, it is possible to quantify the performance of the policy. That cumulative reward can be combined with the states collected from that episode to indicate how well the policy performed in that episode.

Finally, the policy π_μ chooses the action $\boldsymbol{u}$ based on the state of the robot. The policy is a NN, which is initialised with random parameters. Every epoch, the parameters of the policy are updated. The training is complete when a certain reward has been reached or improvement has converged.

One epoch consists of two stages. In the first stage, the policy is used to collect states and rewards from the environment. That is done for a number of episodes, such that sufficient information from the environment is collected. In the second stage, the collected data is used to update the parameters of the policy. This cycle is repeated until training is terminated.

3 TRPO

In the previous section, the reinforcement learning environment is introduced. The algorithm that is behind reinforcement learning is elaborated in this section.

Trust Region Policy Optimisation (TRPO) is developed by [28] and is a variation on the policy gradient algorithms [40]. These have the aim of maximising a cumulative reward function during an episode by executing actions in the robot using a policy: $\pi_\mu(\boldsymbol{u}_t|\boldsymbol{s}_t)$. The policy π_μ gives the probability of executing actions $\boldsymbol{u}_t$ being in state $\boldsymbol{s}_t$, with $\boldsymbol{u}_t$ and $\boldsymbol{s}_t$ as defined in Section 2. The policy π_μ is represented by a NN with parameters μ, which are initially random but are optimised with RL. The optimised parameters result in a higher probability of executing an action u that results in the highest cumulative reward. The cumulative reward function has the general shape:

$$J(\pi_\mu) = \mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_\mu}\left[\sum_t \gamma^t r(\boldsymbol{s}_t)\right] \tag{3.1}$$

with γ being the discount factor ($0 < \gamma \leq 1$), which is introduced for continuous environments. Such a discount factor is generally included if future rewards are less relevant. The discount factor places emphasis on the near future when calculating the cumulative return of a trajectory. The reward function r is given in Equation 2.12.

The cumulative reward function is also known as the objective function, since, during training of the policy π_μ, it is the objective to maximise that function by changing the policy parameters:

$$\max_\mu J(\pi_\mu) = \max_\mu \mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_\mu}\left[\sum_t \gamma^t r(\boldsymbol{s}_t)\right]$$

and finally obtain the parameters:

$$\mu^* = \arg\ \max_\mu \mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_\mu}\left[\sum_t \gamma^t r(\boldsymbol{s}_t)\right]$$

It is important to observe that the policy parameters are chosen based on the reward of one trajectory (or multiple trajectories) and not per time-step. That is especially relevant for the objective of this thesis: a walking robot. During walking, not all time-steps result in an increase in the reward function. If a leg is located in a forward position, it has to swing back first before a new step can be made. During such a swing, the reward function is not increasing (maybe it can even decrease if the robot is pulled back). When the parameters μ are optimised based on the reward of a time-step, the robot is not able to walk. But when the cumulative reward of one trajectory is used, it is possible to learn to walk, since such behaviour results in the highest final position and cumulative reward.

The change in the objective function $J(\pi_\mu)$ with respect to the parameters μ is needed to maximise the objective function. The gradient of $J(\pi_\mu)$ with respect to μ is: $\nabla_\mu J(\pi_\mu)$. To obtain the derivative of $J(\pi_\mu)$, the function is rewritten. The function $J(\pi_\mu)$ calculates the reward for one trajectory of length $N_{\text{traj.}}$, which is now abbreviated to $\mathbb{T}$:

$$\mathbb{T} = (\boldsymbol{s}_1, \boldsymbol{u}_1, r_1, \ldots, \boldsymbol{s}_N, \boldsymbol{u}_{N_{\text{traj.}}}, r_{N_{\text{traj.}}})$$

If it is assumed that Equation 3.1 is continuous, it can be rewritten to:

$$J(\pi_\mu) = \int \pi_\mu(\mathbb{T})r(\mathbb{T})d\mathbb{T}$$

The integral means that the total reward of a trajectory is given by the sum of the probability of executing an action (which is the policy π_μ) multiplied by the reward. The gradient with respect to the policy parameters is now:

$$\nabla_\mu J(\pi_\mu) = \int \nabla_\mu \pi_\mu(\mathbb{T})r(\mathbb{T})d\mathbb{T} \tag{3.2}$$

This can be rewritten in logarithmic form using:

$$\nabla_\mu \pi_\mu = \pi_\mu \frac{\nabla_\mu \pi_\mu}{\pi_\mu} = \pi_\mu \nabla_\mu \log(\pi_\mu) \tag{3.3}$$

Substituting Equation 3.3 in Equation 3.2 gives:

$$\begin{aligned}\nabla_\mu J(\pi_\mu) &= \int \pi_\mu(\mathbb{T})\nabla_\mu \log(\pi_\mu(\mathbb{T}))r(\mathbb{T})d\mathbb{T} \\ &= \mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_\mu}\left[\nabla_\mu \log(\pi_\mu(\mathbb{T}))r(\mathbb{T})\right] \qquad (3.4)\end{aligned}$$

The final expression for $\nabla_\mu J(\pi_\mu)$ (Equation 3.4) is approximated. The approximation can be done by sampling using policy π_μ on the environment to gather rewards r. The rewards do not depend on the parameters μ. Now that the direction of the update for the parameters μ is known, it is possible to update those, by using the gradient descent algorithm [40]:

$$\mu \leftarrow \mu + \nu \nabla_\mu J(\pi_\mu) \tag{3.5}$$

The size of the step is determined by the learning rate ν, which is a hyperparameter.

The approximation of the expectation can be made in multiple ways. One of them is the use of Monte Carlo rollouts, as described in [41]. Here, the policy π_μ is used to play out an entire episode for a number of rollouts. Those rewards and states obtained from the trajectory are used to calculate $\nabla_\mu J(\pi_\mu)$ and with that, the new parameters μ. Now the basis for gradient algorithms has been given, which is also the basis for TRPO.

TRPO builds further on the gradient algorithm by solving two problems of the above-mentioned procedure. The first problem is related to the stochastic environment: it is possible that being in a given state $\boldsymbol{s}_t$ and executing action $\boldsymbol{u}_t$, the robot will be in different subsequent states. Therefore, the cumulative reward will also vary. That is because there is uncertainty in actions, but also uncertainty in the design of the robot. Both types of uncertainties result in variations in the reward. That means that there is a high variance in the obtained rewards, which is not beneficial for RL [42]. An example is that a robot with short legs and a high density cannot perform as well as a robot with long legs and a low density, when executing the same actions. The improvement during training of the policy π_μ can therefore fluctuate. A solution for this is correcting the reward function with a baseline:

$$A^{\pi_\mu}(\boldsymbol{s}_t, \boldsymbol{u}_t) = r(\boldsymbol{s}_t) - V_\lambda^{\pi_\mu}(\boldsymbol{s}_t) = Q^{\pi_\mu}(\boldsymbol{s}_t, \boldsymbol{u}_t) - V_\lambda^{\pi_\mu}(\boldsymbol{s}_t) \tag{3.6}$$

in which A^{π_μ} is the advantage function, which gives how much better the sampled trajectory is compared to the baseline, which is given by $V_\lambda^{\pi_\mu}$ (also known as the value function). The value function gives the expected reward when following policy π_μ in a state $\boldsymbol{s}_t$. The value function is represented by a NN with parameters λ. Q^{π_μ} is known as the quality function, which gives the expected return when being in state $\boldsymbol{s}_t$ and executing action $\boldsymbol{u}_t$ and then following the policy π_μ. The difference between the value function is that the quality function focuses on one action. The advantage function tells directly if action $\boldsymbol{u}$ is better compared to the policy in state $\boldsymbol{s}_t$. A^{π_μ} is used when updating the parameters, to see if executing that action more often is improving the performance of the policy or not.

The value function is initially unknown but is learned with

$$\min_\lambda ||V_\lambda^{\pi_\mu}(\boldsymbol{s}_t) - Q^{\pi_\mu}(\boldsymbol{s}_t, \boldsymbol{u}_t)||^2 \tag{3.7}$$

which is a least squares regression. This equation shows that the value function is the expected return being in state $\boldsymbol{s}_t$ and confirms that the advantage function only improves if the new action results in above-average performance.

The second problem with the gradient algorithm is that it is not sure that an update of π_μ will improve the reward obtained from the environment. That is because of the learning rate ν: which might be too large, overshooting the desired performance improvement. On the other hand, choosing a small learning rate will result in very slow convergence. It is possible to sample with the updated policy in the environment and, if there is no improvement, adjust the learning rate; however, sampling from the environment can take a lot of time. A solution for this is importance sampling [43], which eliminates the need to sample again from the environment to check if an update results in improved performance of the policy π_μ. The old samples are used to estimate the performance of the updated policy π_μ. Those samples were generated with the policy from the previous epoch, now called $\pi_{\mu,\text{old}}$. The idea is to calculate the advantage function (Equation 3.6) again with the new policy. However, only the advantage function $A^{\pi_\mu}(\boldsymbol{s}_t, \boldsymbol{u}_t)$ of the old policy is known. That is where importance sampling is used:

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_\mu} A^{\pi_\mu}(\boldsymbol{s}, \boldsymbol{u}) \approx \mathbb{E}_{\boldsymbol{s},\boldsymbol{u}\sim\pi_{\mu,\text{old}}} \frac{A^{\pi_{\mu,\text{old}}}(\boldsymbol{s}, \boldsymbol{u})\pi_\mu(\boldsymbol{s}, \boldsymbol{u})}{\pi_{\mu,\text{old}}(\boldsymbol{s}, \boldsymbol{u})}$$

The equation shows that it is possible to calculate the expected advantage function of the new policy using the old advantage function, thus eliminating the need to resample. With the old samples, it can be estimated if the new policy will result in improved performance. Importance sampling does not directly solve the problem of an incorrect ν, but it makes sure that it is not needed to sample from the environment to see if an update of the policy π_μ results in an improvement of the cumulative reward. However, there is one drawback of using importance sampling: it is only valid if the two distributions are not too far apart, which in this case means that the difference in behaviour of policy $\pi_{\mu,\text{old}}$ and π_μ should not result in a very different distribution of actions $\boldsymbol{a}$. If the behaviour is very different, it is no longer certain if there is an improvement in the advantage function under the new policy, since the old advantage function is not representative of the new advantage function. The behaviour should stay within a given *trust region*, which is enforced by a constraint:

$$\mathbb{E}_{\boldsymbol{s}\sim\pi_{\mu,\text{old}}} \left[\mathrm{D}_{\mathrm{KL}}\left[\pi_{\mu,\text{old}}(*|\boldsymbol{s}), \pi_\mu(*|\boldsymbol{s})\right]\right] < \delta$$

The difference between the two distributions of actions should be within a limit δ and is measured by the KL-divergence (D_{KL}) [44]. The KL-divergence quantifies the similarity between two distributions.

TRPO, as described in [28], uses these two improvements (the advantage function and importance sampling). The TRPO algorithm can very briefly be described as:

$$\begin{aligned} &\max_{\mu} L_{\pi_{\mu,\text{old}}}(\pi_\mu) \\ &\text{s.t. } \mathbb{E}_{\boldsymbol{s}\sim\pi_{\mu,\text{old}}}[\mathrm{D}_{\mathrm{KL}}[\pi_{\mu,\text{old}}, \pi_\mu]] < \delta \end{aligned} \tag{3.8}$$

with:

$$L_{\pi_{\mu,\text{old}}}(\pi_\mu) = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\pi_{\mu,\text{old}}} \sum_{t=0}^{\mathbb{T}} \frac{A^{\pi_{\mu,\text{old}}}(\boldsymbol{s}_t, \boldsymbol{u}_t)\pi_\mu(\boldsymbol{s}_t, \boldsymbol{u}_t)}{\pi_{\mu,\text{old}}(\boldsymbol{s}_t, \boldsymbol{u}_t)} \tag{3.9}$$

The function $L_{\pi_{\mu,\text{old}}}(\pi_\mu)$ is known as the surrogate advantage or simply the objective function. Every epoch, the parameters of the policy are updated to maximise the surrogate advantage, while staying within the trust region. The surrogate advantage is not the same as the $J(\pi_\mu)$, but they can be related to each other with:

$$J(\pi_\mu) = J(\pi_{\mu,\text{old}}) + L_{\pi_{\mu,\text{old}}}(\pi_\mu)$$

This equality is true in an ideal situation where the expectations are exact. However, those are approximated by samples and therefore it is not true in practice and is usually given as:

$$J(\pi_\mu) \approx J(\pi_{\mu,\text{old}}) + L_{\pi_{\mu,\text{old}}}(\pi_\mu)$$

The approximation can be specified better. It is possible to give a lower bound for the improvements, if the trust region is enforced:

$$J(\pi_\mu) - J(\pi_{\mu,\text{old}}) \geq L_{\pi_{\mu,\text{old}}}(\pi_\mu) - C\mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu]$$

The constant C is relating the KL-divergence to the objective function and is formally derived in [28]. The equation above makes sense intuitively: if the new policy is behaving much differently compared to the old, the KL-divergence is large. That means the surrogate advantage is not that accurate, since it is based on importance sampling. Therefore, the lower bound is reduced (a lot) by the KL-divergence. Note that the KL-divergence term has changed to the maximum KL-divergence, because the lower limit is calculated.

The direction of an update of the parameters μ of π_μ is determined by the surrogate loss and the KL divergence:

$$\mu^* = \arg\max_{\mu} \left[L_{\pi_{\mu,\text{old}}}(\pi_\mu) - C\mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu]\right]$$

The new policy is used to calculate the KL-divergence and objective function: if the KL divergence is within the specified limit and the surrogate advantage one is non-negative, the update is accepted. With the new policy, trajectories and rewards are sampled and the procedure is executed again. It is done until convergence or the maximum number of epochs has been reached.

The equations above for TRPO are not used in that form directly. The max KL-divergence and $L_{\pi_{\mu,\text{old}}}(\pi_\mu)$ are difficult to calculate, so they are approximated by the Taylor expansions:

$$\begin{aligned} L_{\pi_{\mu,\text{old}}}(\pi_\mu) &\approx L_{\pi_{\mu,\text{old}}}(\pi_{\mu,\text{old}}) + \nabla_\mu L_{\pi_{\mu,\text{old}}}(\pi_\mu)|_{\pi_{\mu,\text{old}}}(\pi_\mu - \pi_{\mu,\text{old}}) + ... \\ \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu] &\approx \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_{\mu,\text{old}}] + \nabla_\mu \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_\mu, \pi_\mu]|_{\pi_{\mu,\text{old}}}(\pi_\mu - \pi_{\mu,\text{old}}) \\ &\quad + \frac{1}{2}(\pi_\mu - \pi_{\mu,\text{old}})^T \nabla_\mu^2 \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu]|_{\pi_{\mu,\text{old}}}(\pi_\mu - \pi_{\mu,\text{old}}) + \dots \end{aligned}$$

The second term in the expansion of the objective function has been left out, because it is much smaller in magnitude compared to the first. All of the zero terms of the Taylor expansions can be removed, simplifying the expression to:

$$L_{\pi_{\mu,\text{old}}}(\pi_\mu) \approx \nabla_\mu L_{\pi_{\mu,\text{old}}}(\pi_\mu)|_{\pi_{\mu,\text{old}}}(\pi_\mu - \pi_{\mu,\text{old}}) := g(\pi_\mu - \pi_{\mu,\text{old}}) \tag{3.10}$$

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu] &\approx \frac{1}{2}(\pi_\mu - \pi_{\mu,\text{old}})^T \nabla_\mu^2 \mathrm{D}_{\mathrm{KL}}^{\max}[\pi_{\mu,\text{old}}, \pi_\mu]|_{\pi_{\mu,\text{old}}}(\pi_\mu - \pi_{\mu,\text{old}}) \\ &:= \frac{1}{2}(\pi_\mu - \pi_{\mu,\text{old}})^T H(\pi_\mu - \pi_{\mu,\text{old}}) \end{aligned} \tag{3.11}$$

in which H and g have been introduced for convenient notation. It can be seen that g has the same function as $\nabla_\mu J(\pi_\mu)$. Finally, Equation 3.8 can be rewritten to contain the practical equations for the KL-divergence and objective function:

$$\begin{aligned} \mu^* &= \arg\max_{\mu} g(\pi_\mu - \pi_{\mu,\text{old}}) \\ &\text{s.t. } \frac{1}{2}(\pi_\mu - \pi_{\mu,\text{old}})^T H(\pi_\mu - \pi_{\mu,\text{old}}) \leq \delta \end{aligned} \tag{3.12}$$

and it can be reformulated to obtain the update equation for the parameters μ:

$$\mu = \mu, \text{old} + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{3.13}$$

These updated parameters are used to calculate the objective function and KL-divergence, which are checked against the requirements. It needs to be done, since the updated parameters were calculated with a Taylor expansion, so they are not exact.

The TRPO algorithm, as described, is used in combination with a surrogate model. That surrogate model is required for the collection of samples, states and rewards. It does not matter what type of surrogate model is used, since the TRPO algorithm is not dependent on how the samples are acquired. The types of surrogate models and how they are used in combination with TRPO are presented in the next section.

4 The Surrogate Models

The stochastic environment, as defined in Section 2.3, needs to be simulated. The model estimates the state of the robot, based on its initial position $\boldsymbol{s}_0^{(i)}(\Theta(\omega))$ and all actions taken, from the initial state to the moment in time t $(\boldsymbol{u}_0, \ldots, \boldsymbol{u}_t)$, such that:

$$\boldsymbol{s}_0^{(i)}(\Theta(\omega)) \times \boldsymbol{u}_0 \times \cdots \times \boldsymbol{u}_t \rightarrow \boldsymbol{s}_{t+1}^{(i)}(\Theta(\omega)) \tag{4.1}$$

in which $\boldsymbol{s}_{t+1}^{(i)}(\Theta(\omega))$ is the state to be estimated. The model has as inputs the actions, which are the target positions of the joint, and an initial state. The output of the model is an estimate of the next state. With such a model, it is possible to optimise the actions so that the best next state is reached, using the TRPO algorithm as presented in Section 3. The best state at $t+1$ is the one that gives the highest reward.

A stochastic system can be discretized using various methods. The simplest is to use Monte Carlo simulation [45], which is based on random sampling to estimate the distribution of states. The RVs are sampled randomly and used to simulate a state transition. This is done for a large number of samples and initial states, such that, in the end, the distribution of executing a given action in a given state is obtained. The Monte Carlo method gives accurate results, but the problem with this method is the slow convergence rate [15]. That is especially a problem for systems with a high number of states, which require a lot of samples. In the environment in this thesis, there are two RVs (density and length of the robot links), but the actions $\boldsymbol{u}$ are also uncertain. That means that if a robot has ten actuators, twelve variables need to be sampled. With those twelve variables, all the possible states at $t+1$, being in a given state at time t, can be reached and sampling over all variables gives the distribution at a moment in time. But, that must also be done for every time-step. The result is that a large number of samples are required for the Monte Carlo method.

A more effective estimation method is generalised Polynomial Chaos Expansion (PCE) [16], which generally needs fewer samples for an accurate estimate. It is an effective method to represent a system with stochastic parameters. A second type of surrogate model that is considered is the model ensemble method [26], which uses neural networks to model the state transitions.

4.1 Polynomial Chaos Expansion

With PCE, the system is represented in terms of the polynomial functions of known RVs. A general expression for PCE is:

$$\hat{\boldsymbol{s}}_{t+1}^{(i)}(\Theta(\omega)) = \sum_{\alpha \in \mathcal{J}} c_t^{(\alpha)} \Psi_\alpha(\Theta(\omega)) \tag{4.2}$$

with $\mathcal{J}$ being the multi-index, as defined in [46], and $c_t^{(\alpha)}$ are the polynomial coefficients. Ψ are the polynomials, with arguments being the standard uniform variables $\Theta(\omega)$. The polynomial type used for a standard uniform distribution is the Legendre polynomial [47], which is given by the summation of the monomial:

$$\Psi_\alpha(\Theta(\omega)) = \frac{1}{2^\alpha \alpha!} \frac{d^\alpha}{d\Theta(\omega)^\alpha} (\Theta(\omega)^2 - 1)^\alpha$$

When PCE is applied, there is one condition that should be met [48, 49]: the variance of the estimated RV should be finite, which means that

$$\mathbb{V}\left[\boldsymbol{s}^{(i)}(\Theta(\omega))\right] = \mathbb{E}\left[(\boldsymbol{s}^{(i)}(\Theta(\omega)) - \mathbb{E}(\boldsymbol{s}^{(i)}(\Theta(\omega)))^2\right] < \infty$$

No formal proof is given for this condition, but it can be reasoned that the possible states at time t are limited because of the finite acceleration, according to Equation 2.10, since both uncertainties also have a limited distribution (see Equation 2.9). The position and velocity can increase over time, but they are always finite. Therefore, it can be concluded that the variance of the states is finite, but possibly growing over time.

The model given in Equation 4.2 is not yet correct. The RVs contain density and length. However, it is noted, in the introduction of this chapter, that the actions $\boldsymbol{u}$ are also variables. In 4.2, those actions are hidden in the coefficients, which is only true if the actions are deterministic. The actions are modelled as being uncertain, in order to find the optimal action with RL. It is assumed that every action allowed by the joint has an equal probability of occurring. The actions are therefore drawn from a uniform distribution: $\boldsymbol{u}(\Theta(\omega)) \sim U(\boldsymbol{u}_{\min}, \boldsymbol{u}_{\max})$. The actions can be modelled as deterministic, but only in the situation where the actions are the same for the collected trajectories, which means that the distribution in states is only because of uncertain density and length. When the actions vary between trajectories, additional uncertainties are added to the PCE model every time a new action is executed. An important note is that the actions change every control-interval, not every simulation time-step. In the robots generated by RoboGrammar, there can be various numbers of joints. As mentioned before, only joints with one degree of freedom are used, therefore, a robot with n joints executes n actions per time-step. The uncertainties of the actions are given in the list: $\boldsymbol{u}_t(\Theta(\omega)) = \left[\boldsymbol{u}_t^1(\Theta(\omega)), \ldots, \boldsymbol{u}_t^n(\Theta(\omega))\right]$. The PCE description is now updated to:

$$\hat{\boldsymbol{s}}_{t+1}^{(i)}(\Theta(\omega)) = \sum_{\alpha \in \mathcal{J}} c_t^{(\alpha)} \Psi_\alpha(l(\Theta(\omega)), \rho(\Theta(\omega)), \boldsymbol{u}_t^1(\Theta(\omega)), \ldots, \boldsymbol{u}_t^n(\Theta(\omega))) = \sum_{\alpha \in \mathcal{J}} c_t^{(\alpha)} \Psi_\alpha(\Theta(\omega)) \tag{4.3}$$

in which ω belongs to a new joint probability space for l, ρ and $\boldsymbol{u}_t^1$ until $\boldsymbol{u}_t^n$, with the assumption that they are all independent RV.

The polynomial coefficients in Equation 4.3 need to be determined. To fit the coefficients, the density and length are sampled and trajectories are simulated to obtain states. The actions executed during those trajectories are calculated by the policy π_μ. The samples of density and length combined with the actions are the input for the model [24]. The collection of samples and states is used to fit the coefficients. The states are collected in the vector $\mathbb{S} = \left[\boldsymbol{s}_0^{(i)}, \ldots \boldsymbol{s}_t^{(i)}\right]$. The input of the model are the samples, which are collected in dataset $\mathbb{D} = [l_0, \rho_0, \boldsymbol{u}_0, \ldots, l_t, \rho_t, \boldsymbol{u}_t]$, when running trajectories. The number of collected samples is N_s. There are multiple methods of calculating the coefficients, but here it is chosen to calculate them with least squares regression:

$$\min_{c_t^{(\alpha)}} \sum_{j}^{N_s} \left[\boldsymbol{s}_{t+1,j}^{(i)} - \sum_{\alpha \in \mathcal{J}} c_t^{(\alpha)} \Psi_\alpha(\Theta(\omega_j))\right]^2 \tag{4.4}$$

Once the coefficients are calculated, it is relatively simple to extract the statistical properties of the estimated state. The tilde notation for the coefficient matrix is introduced for convenience and is an abbreviation of the coefficient matrix with the first column (corresponding to the mean) removed:

$$\tilde{\boldsymbol{c}}_t^{(\alpha)} = \boldsymbol{c}_t^{(\alpha)} \backslash \boldsymbol{c}_t^{(0)}$$

The first and second moments (mean and variance) are now given by:

$$\mathbb{E}\left[\boldsymbol{s}_t^{(i)}(\Theta(\omega))\right] \approx \mathbb{E}\left[\hat{\boldsymbol{s}}_t^{(i)}(\Theta(\omega))\right] = c_t^{(0)} \tag{4.5}$$

$$\mathbb{V}\left[\boldsymbol{s}_t^{(i)}(\Theta(\omega))\right] \approx \mathbb{V}\left[\hat{\boldsymbol{s}}_t^{(i)}(\Theta(\omega))\right] = \sum_{\alpha \in \mathcal{J}} (\tilde{\boldsymbol{c}}_t^{(\alpha)})^2 \mathbb{E}\left[\Psi_\alpha \Psi_\alpha\right] \tag{4.6}$$

These two statistical properties could, for example, be used during the training of the universal policy.

4.1.1 PCE Reduction

A substantial problem can be observed from Equation 4.3: the growing number of RV representing the actions results in more computational power needed to obtain the polynomial coefficients as time increases. Not only does the computation become longer, but also the least squares regression (Equation 4.4) can become unstable and an increasing number of samples is needed to obtain an accurate estimate. The instability of the least squares regression can be due to outliers or singularities [50]. Therefore, this model is not practical.

The problem can be solved by reducing the number of RVs representing actions in Equation 4.3. The idea is that a reduced basis can predict the variance as well as the original polynomial basis. A reduced basis has a predetermined number of past actions. For example, if there are three past actions included in the limited basis, there are $n \cdot T$ uncertainties related to the actions. Here, T is the predetermined number of subsequent actions. The first time-steps do not change, only when the number of actions exceeds T, there is a reduction in samples. The coefficients are used to calculate the variance of the estimation of the state [46]:

$$\mathbb{V}\left[\hat{\boldsymbol{s}}_{t+1}^{(i)}(\Theta(\omega))\right] = \tilde{\boldsymbol{c}}_t^{(\alpha)} \Delta (\tilde{\boldsymbol{c}}_t^{(\alpha)})^T \tag{4.7}$$

which is a different form of Equation 4.6. Δ is the diagonal Gram matrix, which is defined as:

$$\Delta = \mathbb{E}(\Psi_\alpha \Psi_\alpha) = \mathrm{diag}(\alpha!) \tag{4.8}$$

The reduced basis has the predetermined number of coefficients, but should have the same variance:

$$\tilde{c}_{t,\text{reduced}}^{(\alpha)} \Delta_{\text{reduced}} (\tilde{c}_{t,\text{reduced}}^{(\alpha)})^T = \mathbb{V}\left[\hat{\boldsymbol{s}}_{t+1}^{(i)}(\Theta(\omega))\right]$$

where the reduced Gram matrix is known, since the reduced polynomial length is known as well. However, this equation cannot be solved easily. That is because the right side of the equation is a square matrix with dimensions equal to the number of states. The left side of the equation has, of course, the same dimensions, but the number of reduced coefficients is, generally, not equal to the number of states squared. It results in an over- or underdetermined system of equations. The simplest method of solving this is with least squares regression:

$$\min_{\tilde{c}^{(\alpha)}_{t,\text{reduced}}} \left| \tilde{c}^{(\alpha)}_{t,\text{reduced}} \Delta_{\text{reduced}} (\tilde{c}^{(\alpha)}_{t,\text{reduced}})^T - \mathbb{V}\left[\hat{\boldsymbol{s}}^{(i)}_{t+1}(\Theta(\omega))\right] \right|^2 \tag{4.9}$$

The mean was removed for the calculation of the variance, so that needs to be included again. The mean of the reduced coefficients is the same as the original coefficients, so the first coefficient can be simply concatenated with the reduced coefficients of the variance:

$$c^{(\alpha)}_{t,\text{reduced}} = \left[c^{(0)}_t, \tilde{c}^{(\alpha)}_{t,\text{reduced}}\right]$$

The reduced polynomial coefficients are now known. The reduction procedure is done every time a new action is executed. In that case, the oldest action $\boldsymbol{u}_{t-T}$ is removed from the RVs in the PCE and the newest action $\boldsymbol{u}_t$ is added. The result is an iterative algorithm that can calculate the reduced PCE model. The final PCE model is given by:

$$\hat{\boldsymbol{s}}^{(i)}_{t+1}(\Theta(\omega)) = \sum_{\alpha\in\mathcal{J}} c^{(\alpha)}_{t,\text{reduced}} \Psi_\alpha(l(\Theta(\omega)), \rho(\Theta(\omega)), \boldsymbol{u}^1_{t-T}(\Theta(\omega)), \dots, \boldsymbol{u}^n_t(\Theta(\omega))) \tag{4.10}$$

4.1.2 Multi-Element PCE

Although the PCE method can be applied to the system, it is possible to further improve the accuracy of the model. PCE is known to have slow convergence in environments with discontinuities, strong non-linearity or long-time integration [51, 52]. The RoboGrammar environment is non-linear, so it might be beneficial to apply the multi-element approach [34].

The idea behind multi-element PCE (ME-PCE) is simple: each RV is divided into a number of smaller domains and for each domain, an estimate is made. The total distribution is now estimated by a number of local PCEs. Every PCE describes a smaller section of the total distribution, which results in higher accuracy and faster convergence of the model. The RVs are given in the vector $[l(\Theta(\omega)), \rho(\Theta(\omega)), \boldsymbol{u}^{t-T}_t(\Theta(\omega)), \dots, \boldsymbol{u}^n_t(\Theta(\omega))]$ of dimension d, where every possible combination of RVs can be described by a hypercube. That is possible since all RVs have a uniform, bounded distribution. That hypercube, representing all the possible combinations of the RVs present in the environment, can be divided into smaller hypercubes, each representing a part of the total hypercube. The hypercube (representing the random space Ω) is now divided into N_{ME} non intersecting parts:

$$\Omega^{n_{\text{ME}}} = [\Theta(\omega)^{n_{\text{ME}}}_{\text{lower},i}, \Theta(\omega)^{n_{\text{ME}}}_{\text{upper},i}) \times \cdots \times [\Theta(\omega)^{n_{\text{ME}}}_{\text{lower},d}, \Theta(\omega)^{n_{\text{ME}}}_{\text{upper},d}) \text{ for } i = 1, \dots, d \tag{4.11}$$

One element of the hypercube is bounded by a lower bound $\Theta(\omega)_{\text{lower},i}$ and an upper bound $\Theta(\omega)_{\text{upper},i}$, which can be chosen freely. The total estimate of the surrogate model is the sum of all the elements:

$$\hat{\boldsymbol{s}}^{(i)}_{t+1}(\Theta(\omega)) = \sum_{n_{\text{ME}}=0}^{N_{\text{ME}}} \mathbb{I}_{n_{\text{ME}}} \sum_{\alpha\in\mathcal{J}} (c^{(\alpha)}_{t,\text{reduced}})^{n_{\text{ME}}} \Psi^{n_{\text{ME}}}_\alpha(l(\Theta(\omega)), \rho(\Theta(\omega)), \boldsymbol{u}^1_{t-T}(\Theta(\omega)), \dots, \boldsymbol{u}^n_t(\Theta(\omega))) \tag{4.12}$$

where $\mathbb{I}_{n_{\text{ME}}}$ is the indicator function. The estimate of the state distribution is the sum of the estimated distributions by the local PCEs.

4.2 The Model Ensemble

The model ensemble (MOE) [26] is implemented as a second type of surrogate model and comparison for the PCE model. The idea behind the model ensemble is to represent a (stochastic) environment with an ensemble of NNs. Together, the NNs predict the distribution of the estimated state.

The model ensemble consists of a predefined number of neural networks that give the next state based on the current state and action: $\hat{\boldsymbol{s}}^{(i)}_t \times \boldsymbol{u}_t \to \hat{\boldsymbol{s}}^{(i)}_{t+1}$. The difference with the PCE model is that it does not depend on older actions and that the current state can be a direct input for the neural network. The neural network has, thus, as inputs the current state and action, and the output is the next state. It is important to note that the MOE is initialised with an observed initial condition, but all the future estimates depend on estimated states. An accumulation of errors occurs with this type of model, which is not the case for the ME-PCE surrogate. With the accumulation of errors, there is a risk of unstable and diverging estimations.

The state distribution should be represented by the ensemble of models. That is done by dividing the collected states in the same way as for ME-PCE. The hypercube of RVs is divided into smaller hypercubes. Each hypercube is estimated by one NN. That means that the samples are sorted into their respective hypercubes, as well as the states, and that combination is used to fit the coefficients of one neural network from the ensemble. The network is fit using a least squares estimate. This is done for every NN with their respective samples. The network needs to have some non-linear activation functions, so that it can replicate the non-linear dynamics of the system. There are two widely used options for activation functions: ReLU or hyperbolic tangents. One NN is labelled $M_k^{p_k}$, where the subscript indicates the index of the network and the superscript are the parameters of that network. The total number of NN in the ensemble is a hyperparameter and is given by N_{MOE}.

The estimate of the model at one point in time is the collection of all the different NNs:

$$\hat{\boldsymbol{s}}_{t+1}^{(i)} = \sum_{k=0}^{N_{\text{MOE}}} \mathbb{I}_k M_k^{p_k}(\boldsymbol{s}_t^{(i)}, \boldsymbol{u}_t) \tag{4.13}$$

which can be used, for example, to calculate the mean of the estimated distribution.

During training, a network is picked from the model ensemble and used to generate a trajectory. If multiple, different networks are used, the collected states contain a distribution, which can be learned by the policy π_μ. In that way, the policy should be robust to the uncertainty in the environment.

The difference between the PCE and the MOE is that the PCE model can estimate the distribution of the estimated state much better compared to the MOE. That is because the MOE uses N_{MOE} NNs to estimate the distribution. So, there are N_{MOE} data-points that estimate the distribution. If the distribution is complicated, that number needs to be very high before the estimated distribution by the MOE is the same as that from PCE. A large number of NNs is not practical: it takes a long time to fit all those NNs. The advantage of MOE is its simplicity for implementation and fitting of the coefficients of the NNs.

4.3 Surrogate Models with TRPO

The two given types of surrogate models are used in combination with TRPO. Together with the simulation environment from the RoboGrammar library, there are three different possibilities for collecting data from the environment. The RoboGrammar simulation environment (referred to as the 'original' environment), PCE surrogate or MOE surrogate are the three options for collecting data.

When the original environment is used, it is possible to directly sample from it. That environment can either be deterministic (without variation in parameters) or stochastic (with variation in the parameters). These two options are abbreviated as DE and SE (deterministic environment and stochastic environment, respectively). When the PCE model is used for collecting data, the training loop for TRPO changes a bit: the coefficients of the PCE need to be fit. The method to do so is inspired by [26]. Every training loop starts with collecting data from the environment. This data is much larger compared to when the original environment is used. With that data, the PCE model is fit. After the model is trained, it is used to sample data for TRPO, which improves the policy. This variation of TRPO is abbreviated to PCE-TRPO (which can be used in SE or DE).

The training cycle of PCE-TRPO consists of two parts: in the first part, the PCE model is fit and, in the second part, the PCE model is used to obtain states and rewards to train the policy with TRPO. One training cycle starts with collecting data from the original environment. The policy is initialised with random parameters, so in the beginning, the visited states are limited. That means that PCE model is only accurate for those limited states. When the policy has been trained on the PCE model and no longer improves, the original environment is used to collect new data. The policy performs better compared to the previous time, so a different or larger part of the state space is explored. However, the policy, most likely, performs worse than what is expected from the cumulative reward obtained with the PCE surrogate, due to overfitting on the surrogate, which is inaccurate. The PCE model is then fitted to that new data and is more accurate for a larger range of states. This cycle continues until the policy performs well enough in the original environment.

Another advantage of PCE-TRPO is that, after one full training cycle, an accurate model of the robot and environment is obtained. That means, when retraining (for example; with a different controller type), there is no longer a need to sample from the real environment. The complete training can be done with the PCE model. Another advantage is that when the robot changes (for example, with an additional pair of legs), the original PCE model can be used as a starting point for the parameters. It accelerates the training process since the behaviour of the new robot is somewhat similar to that of the previous robot.

The use of a PCE model allows for a more efficient generation of the fictitious samples. For example, the highest and lowest possible density and length could be used to simulate the limits of stochastic dynamics. Another possibility is to use different sampling options. A disadvantage of using a PCE model is that it is only valid for the time in which samples have been obtained.

The model ensemble has the same training cycle as the PCE surrogate. First, states need to be collected from the original environment to fit the parameters of the NNs in the MOE. That is done with the untrained policy, so the MOE might not be accurate for states far from the range visited when the data is collected from the original environment. The policy is trained on the MOE with TRPO (abbreviated as MOE-TRPO) and after a stopping condition for training, the cycle repeats. New data is collected from the original environment and the NN parameters are fit again. This cycle is repeated until the policy reaches high enough rewards in the original environment or the predefined number of epochs is completed.

After one full training cycle, an accurate surrogate MOE is obtained, which can be used for training a new policy, while taking much less time per epoch. As with the PCE surrogate, the parameters of the MOE surrogate model can be used as starting parameters when the environment is changed.

5 Implementation and Evaluation

In this section, the implementation of the theory given in Sections 3 and 4 is explained. The implementation was made in Python, which is also used by the RoboGrammar library.

The method of evaluation is also given for every algorithm, along with the reasoning behind why that data is presented and how it can be used to determine the performance of the algorithm.

5.1 Implementation of (ME-)PCE

The theory and equations in Section 4 can be implemented relatively straightforwardly. The first step is data collection, which is done by running trajectories using the RoboGrammar environment. Algorithm 1 shows how that is done. The actions $\boldsymbol{u}$ are chosen by the policy π_μ, so, in the beginning of a training loop, the policy executes random behaviour. When the policy improves, different states are visited. The collected states in $\mathbb{S}$ contain a large part of the (necessary) sample space. $N_{\text{traj.}}$ gives the number of trajectories that are run, which is especially important for the PCE and MOE implementation, since those models require a lot of sampled data before they are accurate. The actions are changed every top-level control interval, which is given by T_{joint}.

Algorithm 1 Collect samples

```
Require: $N_{\text{traj.}}$, $\tau$, $T_{\text{joints}}$
  while $n \leq N_{\text{traj.}}$ do                         ▷ Sample the trajectories one at a time
      Sample $\rho \sim U(\rho_{\min}, \rho_{\max})$           ▷ $\rho$ and $l$ do not change during a trajectory
      Sample $l \sim U(l_{\min}, l_{\max})$
      while $t \leq \tau$ do
          if $t \mod T_{\text{joint}} = 0$ then              ▷ The actions are changed every top-level control interval
              Sample $\boldsymbol{u} \sim \pi_\mu(\boldsymbol{s}_t)$
          end if
          Simulate time-step with ABA
          Store $\boldsymbol{s}_{t+1}^{(i)}$ in $\mathbb{S}$
          Store $\rho, l, \boldsymbol{u}$ in $\mathbb{D}$
      end while
  end while
```

The samples are used to determine the surrogate models. The PCE model is calculated according to Algorithm 2. For the initialization of the algorithm, the Gram matrix and reduced Gram matrix need to be calculated once. They are used for the reduction step. After that, the samples and states need to be sorted into their respective hypercubes, which is needed for ME-PCE. First, it is checked to see to which part the samples belong and then the corresponding states from that sample are added to that respective hypercube. The result is that each hypercube contains samples and states, which are used to fit the coefficients that represent that part of the RVs. This step is skipped when multi-element is not used.

Algorithm 2 Calculate the reduced ME-PCE coefficients

```
Require: $D$, $S$, $\alpha$, $\alpha_{\text{reduced}}$, $T$, $\tau$
  $\Delta \leftarrow \text{diag}(\alpha!)$
  $\Delta_{\text{reduced}} \leftarrow \text{diag}(\alpha_{\text{reduced}}!)$
  while $t \leq \tau$ do                                     ▷ The reduction is done per time-step
      Take $\boldsymbol{s}_{t+1}^{(i)}$ from $\mathbb{S}$
      Take $\rho, l, \boldsymbol{u}_t$ from $\mathbb{D}$
      Divide the samples and states in their respective hypercubes according to Equation 4.11
      for $N_{\text{ME}}$ do
          Obtain $c_t^{(\alpha)}$ with regression according to Equation 4.4
          $\tilde{c}_t^{(\alpha)} \leftarrow c_t^{(\alpha)} \backslash c_t^{(0)}$
          $\mathbb{V}\left[\hat{\boldsymbol{s}}_t^{(i)}\right] \leftarrow \tilde{c}_t^{(\alpha)} \Delta (\tilde{c}_t^{(\alpha)})^T$
          Obtain $\tilde{c}_{t,\text{reduced}}^{(\alpha)}$ with regression according to Equation 4.9
          $c_{t,\text{reduced}}^{(\alpha)} \leftarrow \left[c_t^{(0)}, \tilde{c}_{t,\text{reduced}}^{(\alpha)}\right]$      ▷ Concatenate with the original mean
      end for
  end while
```

With these two algorithms, the PCE model can be determined. The verification must show how accurate the model is, both for the estimation of the distribution as well as a single estimate. The verification of the model is calculated with a validation set. The collected data in $\mathbb{D}$ and $\mathbb{S}$ is split into a training set and a validation set. For both training and validation, two quantities are calculated. Those two quantities give an indication of the performance of the model on the training and validation data. The first one is the Jensen-Shannon divergence:

$$\text{JSD}(P||Q) = \frac{1}{2}D(P||\frac{P+Q}{2}) + \frac{1}{2}D(Q||\frac{P+Q}{2}) \tag{5.1}$$

were $D(P||\frac{P+Q}{2})$ is the Kullback-Leibler divergence. P and Q are two probability distributions, which, in this case, are the state distribution estimate by the model and the sampled distribution from the training or validation set. The JSD is used, since it is a symmetric and smooth version of the KL divergence, so it does not matter which distribution is taken first. The value of the JSD is between 0 (identical distributions) and $ln(2) \approx 0.7$ (no overlap). The value of JSD gives, therefore, an indication of how well the distributions match, without needing to plot both distributions. There is no threshold for the JSD. That is because it is not an intuitive quantity. It is used to see how the difference in distribution changes over time. It is desired that it is as low as possible (in that case, they are similar), but more important is that it is constant, such that all estimates during a trajectory have a similar accuracy.

The second quantity is used to estimate the accuracy of the model based on single estimates. A single estimate is the situation where the PCE model is used to calculate one state value at a point in time, by substituting the respective samples. The difference is subtracted and divided by the sampled state magnitude. That normalisation is done to be able to compare different states with each other. Per time-step a lot of this estimation error is calculated (equal to the number of samples used). The model does not have to perform perfectly on every sample: as long as the general estimate is below a given threshold, the model can be used. That is because TRPO does not train on a single state estimate, but on a batch of estimates. The equation for the relative error is given by:

$$\mathbb{E}\left[\frac{|\boldsymbol{s}_t^{(i)} - \hat{\boldsymbol{s}}_t^{(i)}|}{|\boldsymbol{s}_t^{(i)}|}\right] < \delta_{\text{estimate}} \tag{5.2}$$

The estimate of the state $\hat{\boldsymbol{s}}_t^{(i)}$ is calculated with Equation 4.12. The state $\boldsymbol{s}_t^{(i)}$ is from the training or validation set. Small values are ignored, which is because they can result in a high relative error, but are not relevant looking at the objective and size of the robot. If, for example, a position of 1 mm is estimated as -1 mm, that gives a relative error of 2, but the robot is about a metre in length, so that error in estimation is insignificant compared to the size of the robot.

A third plot that can be made for evaluation of the model is the confidence interval of the estimated trajectory as a function of time and the sampled confidence interval. In that case, the dynamic behaviour and magnitude of the distribution can be observed, which gives an intuitive idea of how the model performs. It is not a good performance indication, since a confidence interval gives too limited information about the similarity in distributions.

5.2 Implementation of the MOE

The model ensemble is implemented as a comparison against the PCE model, with the aim of verifying if a better model (the PCE model) is resulting in improved training in combination with TRPO. The model ensemble can be implemented very similarly to ME-PCE. The data is obtained in the same way as for ME-PCE. The fitting of the coefficients of the model ensemble (Algorithm 3) is not done on all the data-points. Batches of data are used. This reduces the computational time, but also gives a better model, that is less prone to overfitting. For the neural networks, it is also not important that the samples are presented in chronological order, since the model is learning state transitions and not entire trajectories. The ADAM [53] optimisation algorithm is used for the neural networks.

The estimate of the states is separated between two networks: one is used to estimate the velocity and the other the position. That means that for every division in RV, two NN are used. The reason to separate the estimates is to improve the accuracy, because the velocity terms are larger in magnitude and during regression, those terms dominate. The result is that the network is trained more on velocity. It is solved by separating the estimates. To reduce the size of the NNs, not all states of the robot are estimated, only the states that are used by the TRPO algorithm. Those states are specified in Section 5.3.

The evaluation of the model ensemble is more complicated compared to the PCE model. That is because the model ensemble is not expected or designed to give perfect estimates: it just needs to be sufficiently accurate to be used in combination with TRPO. Only when the number of NNs is very large, the distribution is similar compared to the real data and PCE. Therefore, two type of plots are used to visualise the performance of the MOE. The first one is the state with a confidence interval as a function of time. In this case, the model can be visually inspected and assessed. The second metric is the normalised difference between the sampled mean and the estimated mean. The normalised mean difference is calculated with:

$$\bar{\boldsymbol{s}}_{\text{dif.}} = \frac{|\hat{\bar{\boldsymbol{s}}}_{\text{MOE}} - \bar{\boldsymbol{s}}|}{|\bar{\boldsymbol{s}}|} \tag{5.3}$$

Algorithm 3 Obtain the model ensemble

```
Require: $\mathbb{D}$, $\mathbb{S}$, $T$, $\mathbb{T}$
  Initialize $N_{\text{ME}}$ neural networks
  while $t \leq \mathbb{T}$ do
    Take batch of $\boldsymbol{s}_{t+1}^{(i)}$ from $\mathbb{S}$
    Take batch of $\boldsymbol{s}_{t}^{(i)}$ from $\mathbb{S}$
    Take batch of $\rho, l, \boldsymbol{u}_t$ from $\mathbb{D}$
    Divide the samples and states in their respective hypercubes according to Equation 4.11
    for $N_{\text{MOE}}$ do
      for $N_{\text{epochs nn}}$ do
        Fit the neural network coefficients using least squares regression
      end for
    end for
  end while
```

with the mean of the state represented by a bar, from the data and the estimated mean from the MOE. $\bar{\boldsymbol{s}}_{\text{dif.}}$ is the relative difference between the estimated and sampled state. This can be used to compare the different states with each other.

5.3 Implementation of TRPO

There are three versions of TRPO that are used. The first one is TRPO with the RoboGrammar environment. The second one is with the model ensemble (MOE-TRPO) and the final implementation is with the PCE model (PCE-TRPO). All three versions can be implemented in a stochastic or deterministic environment. The three versions only differ in the way data is obtained, the TRPO algorithm is the same every time. The TRPO implementation of [54] is used. This implementation is based on the original TRPO implementation by [28]. The TRPO implementation has been adapted to work with the RoboGrammar library.

Algorithm 4 TRPO

```
Require: $N_{\text{epochs}}$
  Initialize $\pi_\mu$
  Initialize $V_\phi$
  for $N_{\text{epochs}}$ do
    Collect $\mathbb{D}$ and $\mathbb{S}$ according to Algorithm 1
    Calculate the advantages with Equation 3.6                 ▷ Train $\pi_\mu$ with TRPO
    while $\text{D}_{\text{KL}}^{\max} > \delta$ do             ▷ Update parameters until requirement is met
      Calculate the surrogate advantages with Equation 3.9
      Update the parameters with Equation 3.13
    end while
  end for
```

The implementation of TRPO is given in Algorithm 4. The data collection can be done with a DE or SE, but that does not change the implementation of TRPO. The training loop is repeated until a specified number of epochs, but that could also be changed to stop when a certain reward has been reached or there is no improvement anymore (the training has converged).

When the MOE is used instead of the RoboGrammar environment, another change has to be made. In Algorithm 5, a while loop is added, which makes sure that the MOE is used to sample data and update the policy. The while loop is stopped when a certain condition is met. There are multiple options for this condition. It could stop when the policy is no longer improving on the MOE, but this will result in the policy, but if the MOE is behaving differently compared to the original environment, the policy will not perform very well. That means that a lot of the training time was useless. A better stopping condition is to break the loop if a certain improvement has been reached compared to the performance at the beginning of that epoch. This has the advantage that the policy is not trained too long on the MOE. A third option is to set a specified number of policy updates every $N_{\text{model updates}}$. This last condition is implemented, because it is the simplest to determine and control the training time. The number of epochs (or policy updates) is now equal to the number of model updates $N_{\text{model updates}}$ multiplied by the set number of policy updates.

The last implementation is with PCE (Algorithm 6) and it does not require much change compared to MOE-TRPO. The largest difference is that the PCE model does not require the current state, only the next state, the current action and previous actions (that are within the action window). The rest of the algorithm is unchanged.

Algorithm 5 MOE-TRPO

```
Require: $N_{\text{model updates}}$
  Initialize $\pi_\mu$
  Initialize $V_\phi$
  for $N_{\text{model updates}}$ do
      Collect $\mathbb{D}$ and $\mathbb{S}$ according to Algorithm 1
      Obtain reward $r$ from RoboGrammar environment
      Calculate the model ensemble according to Algorithm 3
      while stopping condition is false do                    ▷ Loop until stopping condition is met
          Collect states, actions and rewards using MOE surrogate
          Train $\pi_\mu$ with TRPO, using the collected data
          Check the stopping condition
      end while
  end for
```

Algorithm 6 PCE-TRPO

```
Require: $N_{\text{model updates}}$
  Initialize $\pi_\mu$
  Initialize $V_\phi$
  for $N_{\text{model updates}}$ do
      Collect $D$ and $S$ according to Algorithm 1
      Calculate $c^{(\alpha)}_{t,\text{reduced}}$ according to Algorithm 2
      while stopping condition is false do                    ▷ Loop until stopping condition is met
          Use the PCE surrogate to collect states, rewards and actions.
          Train $\pi_\mu$ with TRPO with collected data
          Check stopping condition
      end while
  end for
```

The TRPO performance is evaluated by looking, first of all, at the cumulative reward per epoch. This can be plotted for every epoch, to see if the training converges and what the expected reward from the policy is. Here, the monotonic improvement from TRPO should be observed. The cumulative reward is saved for multiple training cycles, such that it is possible to have a confidence interval for the cumulative reward. It is important to note that a part of the confidence interval is to the variation in training cycles, but also due to the robot used. For example: a long, light robot can move faster and further compared to a short, heavy robot, thus the reward is different due to the robot and not the policy.

The trained policies are evaluated for a number of combinations of length and density, such that the entire range is represented. The cumulative reward per episode is used for this. This is also done for the policies obtained with DE-TRPO, which give a sanity check: those policies should not be able to obtain high rewards over the entire range of RVs.

The policy is trained on a limited version of the robot. 'Limited' refers to the fact that only four of the ten joints are actuated, the other six joints are locked. The used joints can be seen in Figure 5.1. With these four joints, the robot is able to walk. Enabling the additional joints could improve walking, but also increase the complexity of controlling the robot. The joints are limited in their movement: they have to be within 1.5 *rad* of their initial position. This limitation has been added to make sure that the robot does not self-collide or get into physically impossible positions. Also, a limited number of states are presented to the controller. Only the velocity and position of the base link and the two outer links of the legs are used. With the limited state information, the policy can still obtain the position of the robot and its legs, but with the advantage of strongly limiting the number of states it needs to process.

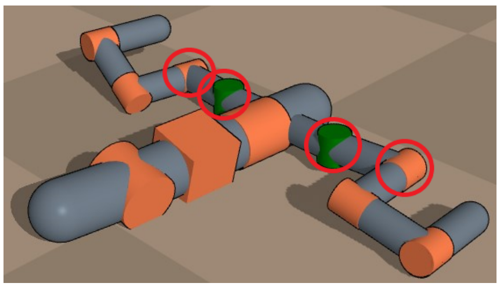


Figure 5.1: Robot used for the simulations with the four actuated joints encircled.

The simplified implementation of the robot reduces the number of outputs for the policy to only four joints and the number of states to only eighteen. This significantly reduces training time.

A final important detail is that the tuning of hyperparameters and implementation algorithm is of significant importance for the performance of TRPO, as highlighted by [55, 56]. Those parameters are set by trial and error, using experience and taking inspiration from papers using RL and TRPO.

6 Results

The results are separated into three parts: the first two sections are dedicated to the surrogate models. The focus is on the accuracy of the prediction of the states. The third part is about the performance of the universal policy in combination with the surrogate models or the original environment.

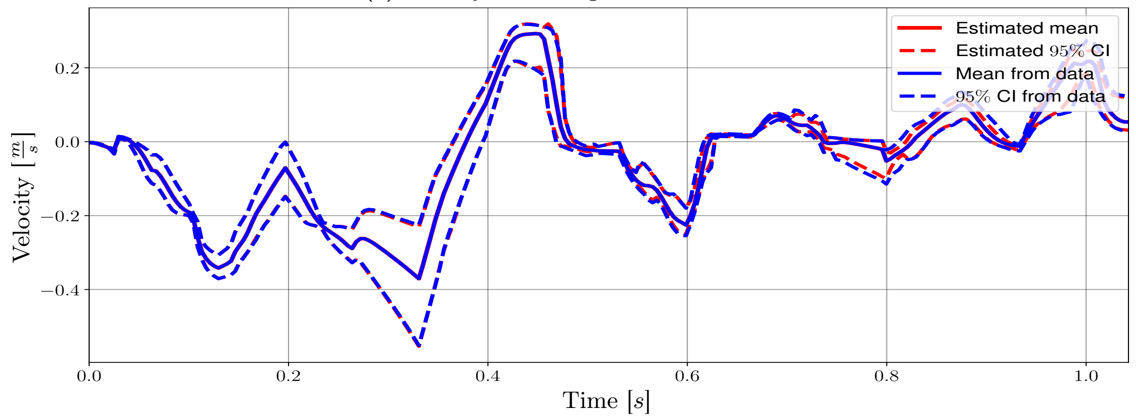
6.1 Results PCE Model

The focus of these results is to show that the PCE model is able to predict the necessary states with sufficient accuracy. The model is accurate when the predictions meet the conditions given in Section 5.1. Not all the available states from the robot are used by TRPO, only the selection specified in Section 5.3. Those states are estimated by the surrogate model to reduce the computational time and complexity of the model compared to estimating all states of the robot. All samples used in this section were collected in the original environment.

6.1.1 Uncertain Density and Length

The first implementation of the PCE model is done in a simplified environment: only variation in density and length is considered and the actions are deterministic. That means that when trajectories are simulated, the density and length are sampled from their uniform distribution, but the actions are the same for every trajectory. The action sequence was randomly generated once. With this simulation, it is possible to determine the bounds of ρ and l, which result in an acceptable distribution. This simulation is also used to get an indication of the polynomial order needed for an accurate estimate. An acceptable distribution is a qualitative choice: the distribution in the states should not be too small (meaning that the variation of density and length does not influence the dynamics), but also not too large, such that the distribution in states is too large to be captured by PCE.

(a) Velocity in the longitudinal direction



(b) Jensen-Shannon divergence of the fit and evaluation

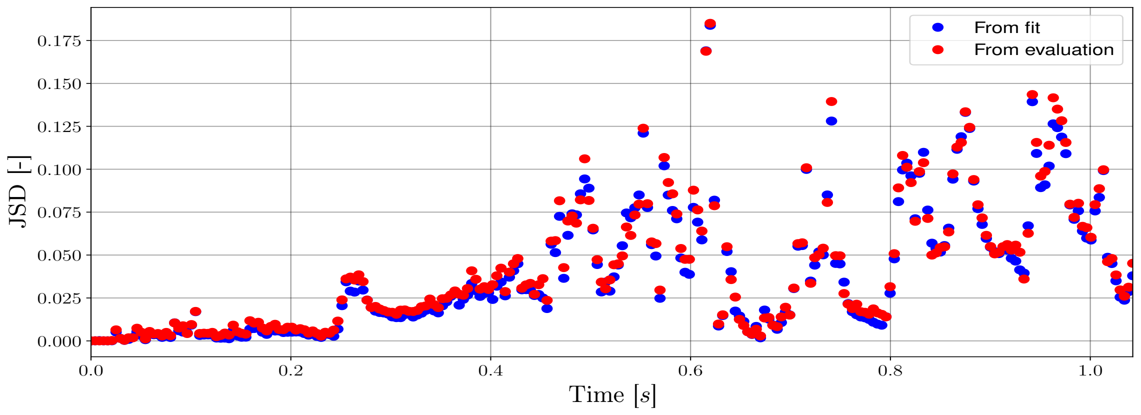


Figure 6.1: Estimation of the velocity by the PCE model with a polynomial order of 15. The estimated state is the velocity of the base link. The top plot shows the trajectory (estimated and sampled). The bottom plot shows the JSD of that trajectory, for both the fit and the evaluation of the model.

The estimate from the PCE model can be seen in Figures 6.1a and 6.1b. The distribution of the density was bounded between 0.7 and 0.8 $\frac{kg}{m^3}$ and the distribution of the length between 0.15 and 0.17 m. These bounds were chosen based on multiple simulated trajectories and ranges of density and length. For the calculation of the polynomial coefficients, $5 \cdot 10^5$ samples were used, with 10% of those samples used for evaluation. A polynomial up to and including order 15 was used for estimation, for both uncertainties. The length of the sampled trajectories was 16 actions, which is shorter than what will be used in combination with TRPO. The length has been reduced to save computational time and stored data.

In Figure 6.1a the velocity of the base link in the longitudinal direction can be seen. The estimated distribution (in red) is similar to the sampled distribution, only around 0.8 seconds, there is a larger difference between the estimation and the data in the 95% confidence interval. The figure suggests that the estimate by the PCE model is quite accurate. But, that figure gives an intuitive measure of accuracy. Although the mean and confidence interval are similar, it is possible that the distribution shapes do not correspond. The similarity in distribution is given in Figure 6.1b, where the JSD is plotted for the data used for fitting (the blue dots) and the data used for evaluation (the red dots). First of all, the values of the fit and evaluation are close to each other, meaning that enough samples were used. If too few samples were used, the coefficients would overfit the training data and the difference between the JSD of the fitting data and the evaluation data would be large. The JSD increases as time increases, meaning that the estimate becomes less accurate as time increases. From the two plots, it is concluded that the distribution is estimated accurately at the beginning of the trajectory but, as time increases, becomes more inaccurate.

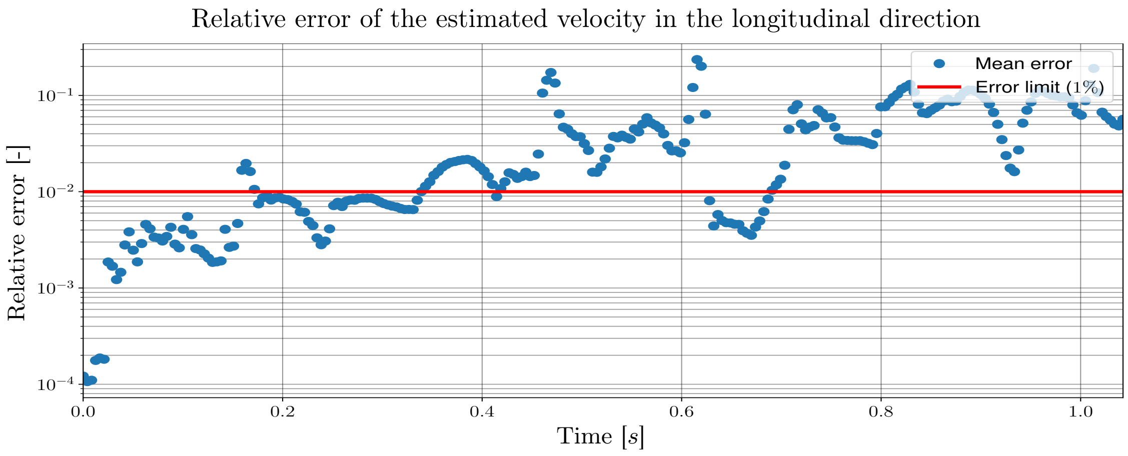


Figure 6.2: Estimation of the velocity by the PCE model of order 15. The data is from the same trajectory as in Figure 6.1a.

Figure 6.2 gives an indication of the performance of estimating individual data points. It shows the mean relative error (calculated with equation 5.2), between the estimated data and the data used for fitting. The error limit is set at 1% (so 0.01). That has been chosen based on the size of the robot: It is about 1 metre in length and width. If the surrogate model estimates a velocity of 1 $\frac{m}{s}$, it has a maximum error of 1 $\frac{cm}{s}$, which is small compared to the size of the robot. Another justification for the error limit is the goal of the robot: the robot does not have to move to an exact position or move with a predetermined velocity, it only has to move forward. Therefore, the objective does not require a very accurate model. The relative error is also calculated using a lower limit of 1 cm or 1 $\frac{cm}{s}$, so data of a smaller magnitude is ignored in the calculation. The error limit was set before using PCE in combination with TRPO, so it is possible that it can be higher, or must be lower to work in the desired application. The relative error from fitting (Figure 6.2) and evaluation (Figure A.6) are almost identical, meaning that the number of samples used for fitting and evaluation is sufficient.

The most important observation from Figure 6.2 is that the relative error is much higher compared to the target of 1% (0.01): it has a maximum around 0.25. The fact that PCE with a polynomial up to order 15 cannot capture the data well, shows that the data is too complex to be fit by a simple PCE model. Usually, a polynomial order smaller than 8 is used in literature [15, 18, 21]. It indicates that multi-element PCE is needed. Figure 6.3 confirms the complexity of the distribution: it shows the distribution of the velocity in longitudinal direction at 0.42 s. The distribution has multiple peaks. Such a multi-peak distribution is difficult to estimate and is an indication of non-linear behaviour in the dynamics of the system. The figure shows that the PCE model poorly estimates the distribution, explaining the increase in JSD and relative error.

The surrogate does not perform as badly in all estimated states. The estimation of the longitudinal position of the base link (can be seen in Figures A.1a, A.1b and A.1c) is much better, even below the desired error limit. That can be confirmed by looking at the distribution of data, see Figure 6.4, which does not show the multi-peaks and a much better estimate from the model. However, the model is becoming inaccurate over time, just as with the estimate of the velocity in the longitudinal direction. This is visible in Figures A.1b and A.1c, where both the relative error and JSD are increasing during the trajectory and do not converge within the trajectory. So, it cannot be concluded that the PCE model can accurately estimate the position for the entire length of the trajectory, when used with TRPO. It can only stay below the error threshold for the first second.

All other estimated states had similar behaviour as the two states given above. The trajectory, relative error and JSD are also given for the position in vertical direction (Figures A.2a, A.2b and A.2c), the position in lateral direction

(Figures A.3a, A.3b and A.3c), velocity in vertical direction (Figures A.4a, A.4b and A.4c) and the velocity in lateral direction (Figures A.5a, A.5b and A.5c). All states are from the base link. The estimated states from the two outer links of the legs showed similar behaviour, and therefore the respective plots are not added to this report.

It was found that the velocity in the longitudinal direction of the base link is the least accurately estimated state. That state is, therefore, used to determine the accuracy of the model, as it is desired that all states are estimated below the desired threshold. The longitudinal position of the base link is the hardest position to estimate, so that state is also used to verify the accuracy of the surrogate model.

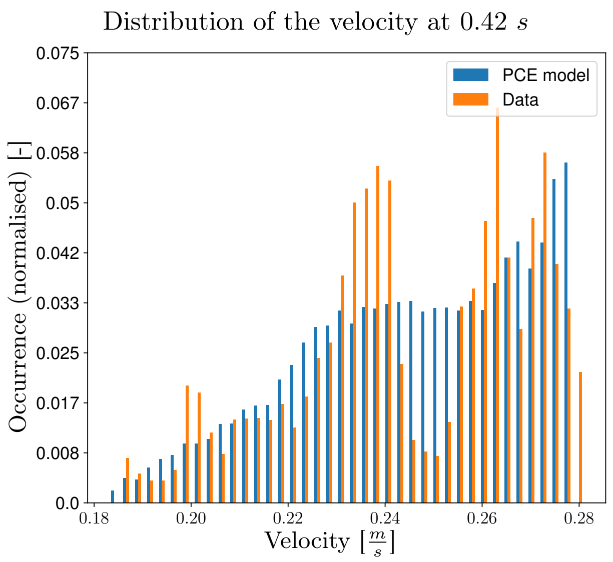


Figure 6.3: Distribution of the velocity in the longitudinal direction at 0.42 s, from the trajectory shown in Figure 6.1a.

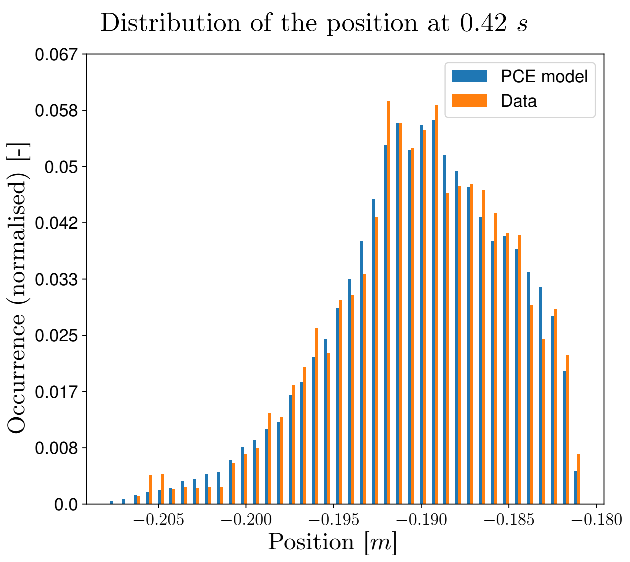


Figure 6.4: Distribution of the position in the longitudinal direction at 0.42 s. The trajectory and relative error can be seen in Figure A.1a and A.1b respectively.

ME-PCE was implemented to reduce the range of the distribution that a PCE has to predict and, with that, improve the accuracy of the model. The distribution of density and length was divided into equal parts. It was chosen to give both RVs an equal number of divisions. The accuracy of the estimate is now dependent on the polynomial order and the number of divisions per uncertainty. There is an optimal choice between the two: a higher number of divisions results in a lower polynomial order needed to get the same accuracy. Figure 6.5 shows a heatmap of the maximum mean relative error per trajectory of the velocity in longitudinal direction (the same trajectory was used for the previous results; it is the trajectory from Figure 6.1a) for various combinations of polynomial order and RV divisions. Note that the divisions are per uncertainty: so if there are two divisions per uncertainty, there are four local PCEs that make up the total ME-PCE model. The bounds for the polynomial order were chosen based on literature and practical considerations: an order higher than six is not used much and requires much more samples and computational time. The range of the number of divisions was also based on practical considerations: more divisions require more samples, which again, takes more computational time.

It can be seen (Figure 6.5) that using a polynomial order of two and two divisions per uncertainty (top left square in the heatmap) gives an estimate that is as good as using a polynomial order of 15 without multi-element. It clearly shows the benefit of using ME-PCE. The effect of more divisions is much stronger compared to using a higher polynomial order: going from two divisions to six divisions results in an improvement of about 0.08, whereas increasing the polynomial order from two to six results in an improvement of about 0.05. So it is more beneficial to use a higher number of divisions compared to a higher polynomial order. Two combinations from Figure 6.5 seem to be the optimal choice: a polynomial order of three and seven divisions or a polynomial order of four and seven divisions. The latter gives almost the highest possible accuracy (within the given range), but it was chosen to use the first option. That was done because, if the actions are added as RVs, the number of coefficients increases a lot, so choosing an order of three results in a lot fewer samples needed and less computational time.

Another important observation is that the required accuracy of 0.01 can still not be reached: the lowest obtained value for the maximum mean relative error per trajectory is 0.126. Although ME-PCE results in an improvement of about 0.12, this is not sufficient. Another change has to be made in the ME-PCE model in order to reach the target accuracy.

The required number of samples, needed for fitting the coefficients, was determined by comparing the maximum mean relative error between the fit and the evaluation. If the number of samples used is sufficient, both should be almost the same. Figure 6.6, shows that. The orange data points are the mean relative error from the evaluation, and the blue data points are the mean relative error from the fitting. For a ME-PCE estimate of order three and

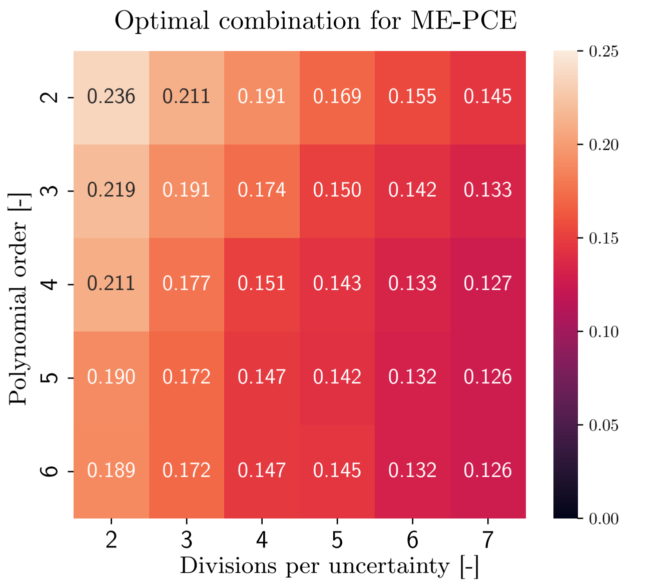


Figure 6.5: Heatmap for selection of optimal combination of the polynomial order and the number of divisions per uncertainty for ME-PCE. The values are the maximum mean relative error per trajectory of the velocity in the longitudinal direction of the base link.

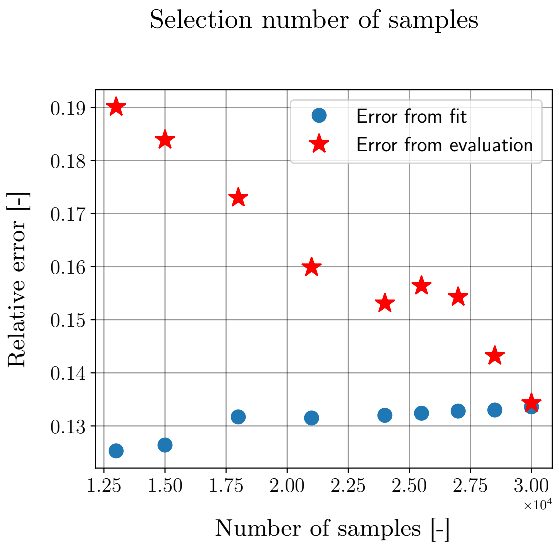


Figure 6.6: Plot of the mean relative error per trajectory of the velocity in the longitudinal direction for a various number of samples used for estimation of the coefficients. The PCE model had seven divisions per uncertainty and a polynomial order of three.

seven divisions per uncertainty, $3 \cdot 10^4$ samples are sufficient. When a small number of samples are used for fitting, the relative error from the fitting is decreasing, but the error from the evaluation is increasing. That is because the coefficients are overfitting the data, resulting in poor evaluation.

The relative error for the velocity in longitudinal direction with the multi-element implementation is much lower (see Figure A.8) compared to the normal implementation of PCE (Figure 6.2). This is also the case for the estimation of the position in the longitudinal direction (see Figure A.7a and A.7b). The results indicate that the use of ME-PCE is increasing the accuracy of the model significantly, but it is not yet sufficient to reach the target accuracy.

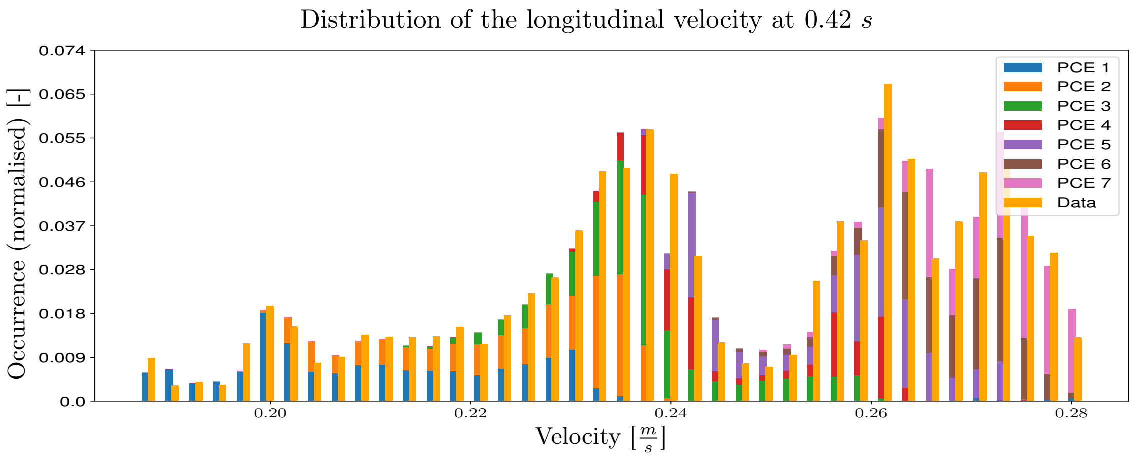


Figure 6.7: Distribution of the velocity in the longitudinal direction, estimated with ME-PCE with a polynomial order of three and seven divisions per uncertainty. The estimate shows the contributions of the seven divisions for the RV length, labelled as PCE $1 - 7$. The shown distribution is the same sampled data as shown in Figure 6.3.

It is interesting to verify the contribution of one of the local PCE from the ME-PCE to the total distribution. It is difficult to visualise this for all 49 (7 per uncertainty) PCEs, therefore, it has been chosen to only focus on the division in length. Figure 6.7 shows the sampled velocity and the distribution due to every division in the RV length. It is clear that each set is dominant for a part of the distribution, which shows that using the multi-element approach makes sense in this case: the distribution range of each local PCE has been decreased and the accuracy increased.

That effect is also observable in the other states. In Figure A.9 the histogram of the distribution of the position in longitudinal direction is given for the seven divisions of the RV length. Comparing this figure to the distribution calculated with the PCE model of order 15 (Figure 6.4) it can be seen that the distribution is estimated more accurately. The relative error of the estimate improves as well: in Figure A.10 the relative error from the evaluation of the PCE model with seven divisions per uncertainty and a polynomial order of three is plotted. When compared to the relative error of the PCE of order 15 (Figure A.1b), it is lower over the entire length of the trajectory.

6.1.2 Uncertain Actions

In the next results, the actions are also stochastic, next to the density and length. Now, every trajectory has different density, length and actions, which are all drawn from a uniform distribution. All RVs had a polynomial order of three, which was based on the findings in the previous section. The action window contained three actions, so that means that there were a maximum of 14 RVs in one estimate (three times four actions together with density and length). With a polynomial order up to and including three, that resulted in a maximum of 512 coefficients per time-step. The estimates are made with ME-PCE, however, it is chosen to only divide the density and length. If all RVs were divided, that would result in too many local PCEs. That is because the previous results indicate that for density and length, seven divisions are needed, so even if the action distribution were divided by two, over $2 \cdot 10^5$ PCEs are needed ($7^2 \cdot 2^{12} \approx 2 \cdot 10^5$). That would not be a practical model and thus it is chosen not to divide the action distribution and to keep the division of density and length at seven. It is not expected that the estimation is more accurate compared to the estimation done without uncertain actions: in the best case, the estimate is as good.

The implementation of ME-PCE used 10^6 samples and 5% was used for evaluation. The length of the estimated trajectory was only 0.4 s, which has been limited compared to the previous results (which were about 1 s in duration) due to the computational time needed and the number of samples. The estimated trajectory, JSD and relative error were again used to assess the performance of the model.

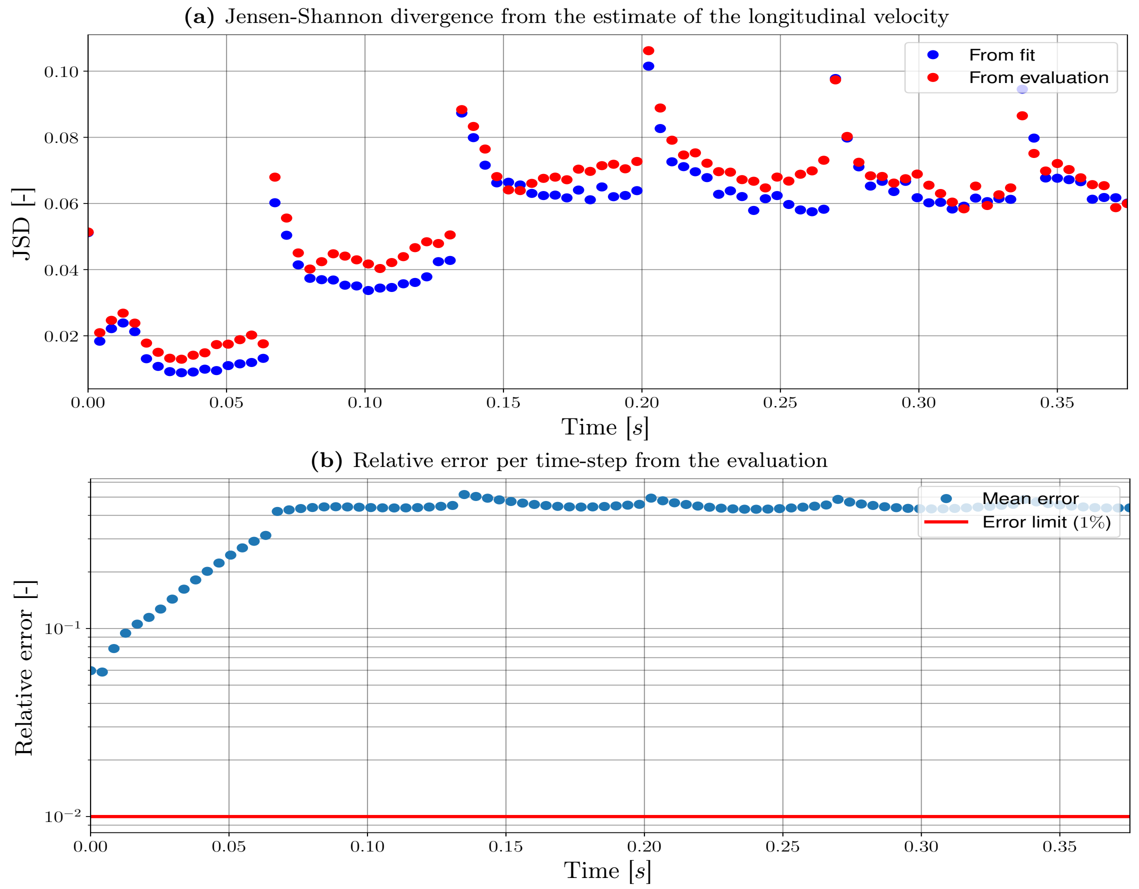


Figure 6.8: Estimation of the velocity by the ME-PCE model, with a polynomial order of three and seven divisions for the uncertainty in density and length. The top plot shows the JSD. The bottom plot is the mean relative error.

The same types of figures are used as with the environment without varying actions, such that they can be compared easily. The estimate of the trajectory of the velocity in the longitudinal direction can be seen in Figure A.11. The JSD of the estimate of the velocity in longitudinal direction (Figure 6.8a) is lower compared to the JSD from the estimate without varying actions (Figure 6.1b). That can be explained by looking at Figure 6.9, in which the distribution of the velocity has been plotted, for the PCE model and the estimate. The multi-peak that was observed earlier in the distribution of the velocity is not present, but the model is still not able to predict the distribution nicely. The fact that the JSD is lower compared to Figure 6.1b, is because the distribution is much simpler: it looks similar to a normal distribution. The estimated distribution by ME-PCE also shows a normal-like distribution, but with shorter tails. But even though the tails are shorter, the JSD is lower because the estimated and sampled distributions are more similar.

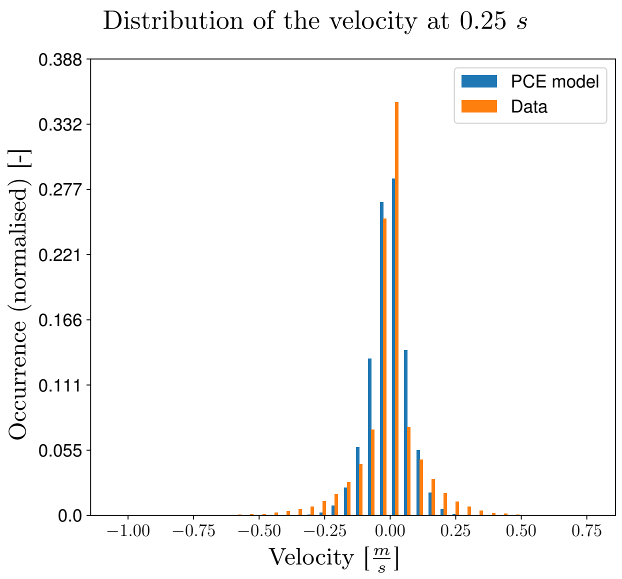


Figure 6.9: Distribution of the velocity in the longitudinal direction at 0.25 s. The trajectory from which this distribution is taken can be seen in Figure A.11. The estimate was made with a ME-PCE of polynomial order 3 and 7 divisions per uncertainty.

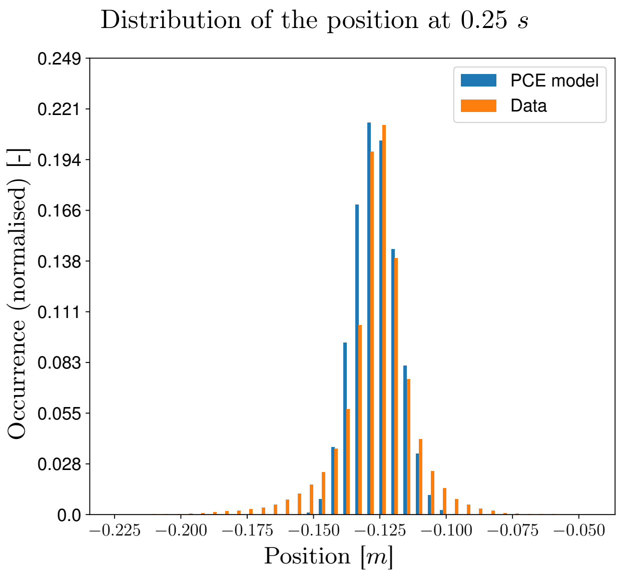


Figure 6.10: Distribution of the position in the longitudinal direction at 0.25 s. The distribution is from the trajectory in Figure 6.11a. The estimate was made with a ME-PCE of polynomial order 3 and 7 divisions per uncertainty.

However, the relative error from evaluation (Figure 6.8b) is worse compared to the estimate without uncertain actions. The average mean relative error is around 0.5. That is much higher compared to the target of 0.01. A reason for this can be that when varying the actions, the distribution of the states is larger compared to the situation without varying the actions. The PCE model performed already below the requirements in the case without varying the actions. By having a larger distribution to fit, the ME-PCE model performs worse, because each local PCE has to capture a larger distribution.

Another observation can be made with regard to the reduction algorithm used. Looking at the estimate of the velocity in longitudinal direction, the relative error and JSD are converged around 0.19 s (Figures 6.8a and 6.9 respectively). At that same point in time, the confidence interval of the estimate trajectory (Figure A.11) is also converged to a wave-like behaviour of constant magnitude. At that moment in time, three actions have been executed and the fourth action is taken. That means that the first action is removed from the PCE model (only three actions are in the action window), so the reduction step is used for every new action executed. As both the JSD and the relative error are constant for the rest of the trajectory and do not behave differently compared to earlier in that trajectory or the sampled data, it seems that the reduction step is working for this state. That is, because if the velocity was dependent on older actions (outside of the used action window), the error would increase over time. This claim cannot be made for sure, since the estimate is inaccurate, so it is not sure if that is also due to the reduction step.

The estimate of the position (Figure 6.11a) reveals that the reduction of RVs does not work for this state: when the fourth action is taken (at 0.2 s), there is a discontinuity in the confidence interval. The interval is reduced, which is contradictory when looking at the confidence interval of the sampled data. A similar observation can be made looking at the JSD (Figure A.13): after 0.2 s, the divergence increases significantly when a new action is executed, meaning that the difference between the estimated and sampled distribution is increasing. This effect can also be seen looking at the distribution of the position (Figure 6.10), where the estimated distribution has significantly shorter tails compared to the sampled distribution.

This effect can be explained by how the reduction of the ME-PCE model works: at 0.2 s, the model no longer takes into account the first action and the results of that action are also not stored in the model. When looking at the figure (Figure 6.11a), it seems that the displacement, obtained after that action, is removed from the confidence interval: the confidence interval is reduced after the action is executed. That does not make sense, looking at the confidence interval of the sampled data, which keeps increasing over time. The reduction of the confidence interval does not happen when all actions are still included in the PCE model (for the first 0.2 s). The estimate of the position does seem to indicate that the reduction step is not working for this state, however, it cannot be concluded for sure, as the model is inaccurate.

The type of action in the robot (the joint target position), can be noted in the behaviour of the states. This is, for example, observable in the velocity in the longitudinal direction. The JSD plot (Figure 6.8a) has a 'spike' when a new action is executed, which can be explained by how the actions are implemented. The actions are the joint target position, so the resulting velocity is dependent on the difference between the joint position and the target position. The larger the difference, the greater the torque applied to the joint. That results in a large acceleration of the joint, and a large and complex distribution of the velocity. The ME-PCE model is not able to predict such complex distributions well, resulting in an increased JSD. When the target position and joint position are closer to each other, the torque on the joint decreases and with that the acceleration. The velocity distribution becomes less complex and is better estimated by the model, resulting in a lower JSD.

At the moment in time when a new action is taken, it can be assumed that the joint position is equal to (or close to) the joint target position of the previous action. The resulting velocity is therefore only dependent on the current action and the previous action. This was verified (see Figure A.14a and A.14b for the relative error and Figure A.15a and A.15b for the JSD with estimates with two and one action in the action window). It can be seen that the relative error is very similar for all three action windows. But, the JSD is significantly higher for the action window with only one action. Especially, at the first time-steps after a new action is taken. For two or three actions in the window, it is similar. It seems that an action window of only one is too small to be used, which makes sense given the fact that the velocity is dependent on the current action, as well as the current position (which is again dependent on the previous action). This statement cannot be fully confirmed, since the estimate is inaccurate, which can be for multiple reasons.

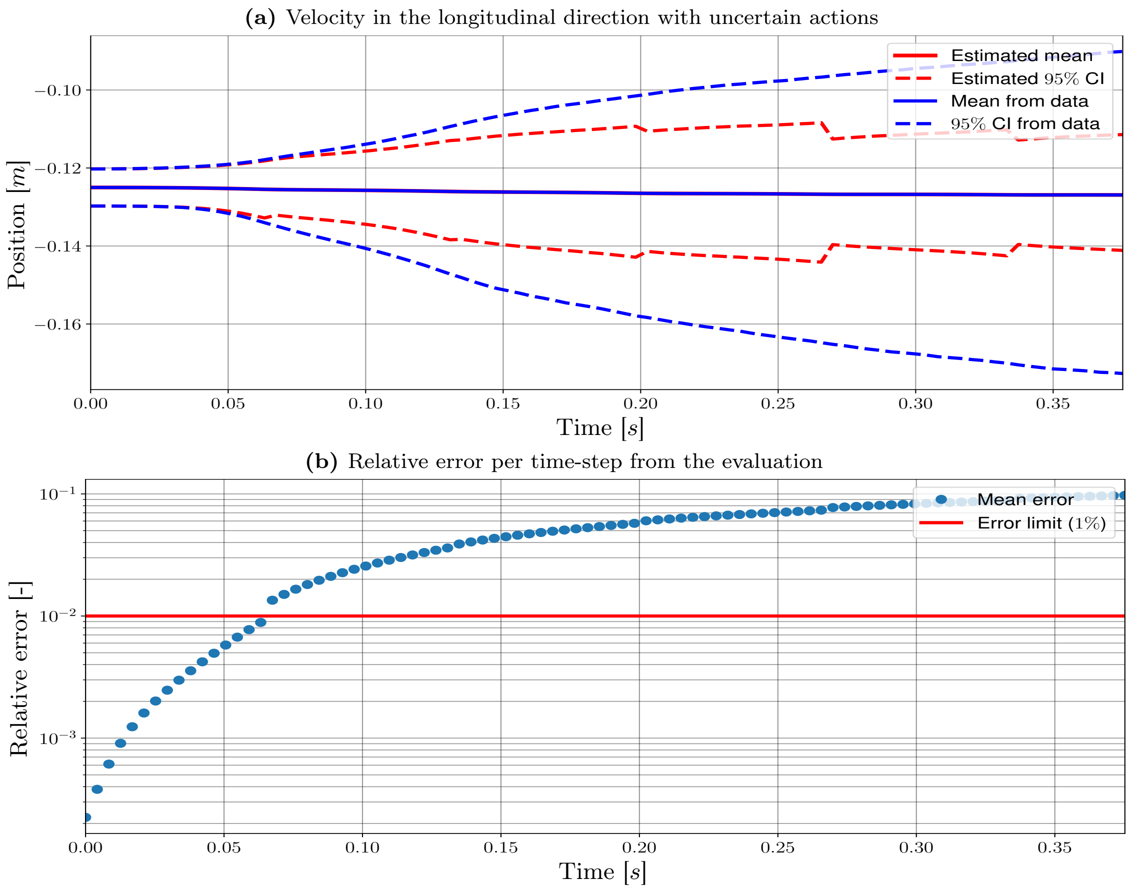


Figure 6.11: Estimation of the position by the ME-PCE model, with a polynomial order of three and seven divisions for the uncertainty in density and length. The top plot shows the trajectory (estimated and sampled). The bottom plot shows the relative error per time-step.

Finally, with the addition of uncertain actions, it is also relevant to look at the estimation performance of a single trajectory. For the model to work with TRPO, it is not only important that the distribution is estimated correctly, but also that a single trajectory is representative of the robot's behaviour. If there are, for example, discontinuities in a single estimated trajectory, the policy might use those for improving on the objective, although that behaviour is not possible in the real environment. One such trajectory can be seen in Figure 6.12. The confidence interval in the estimate represents the uncertainty due to uncertain length and density. It is chosen to keep those uncertainties, since it is not generally known what those are, only that the sampled trajectory is from a robot with the random parameters sampled from the given range. The estimated trajectory does not follow the sampled trajectory well, which is not surprising given that the average relative error is around 0.5. But it seems that the general shape of the estimate corresponds with the trajectory. If the velocity is increasing, the model is also increasing in velocity, same is true when the velocity is decreasing. The confidence interval is similar in magnitude to that in Figure 6.1a, which indicates that the model is able to predict the confidence interval due to density and length, also in this case, well.

The performance of the model on a single trajectory of the position (Figure A.12) is, not surprisingly, very poor: the reduction algorithm does not work for this state and the relative error is high. The discontinuities in the estimate can also be clearly seen.

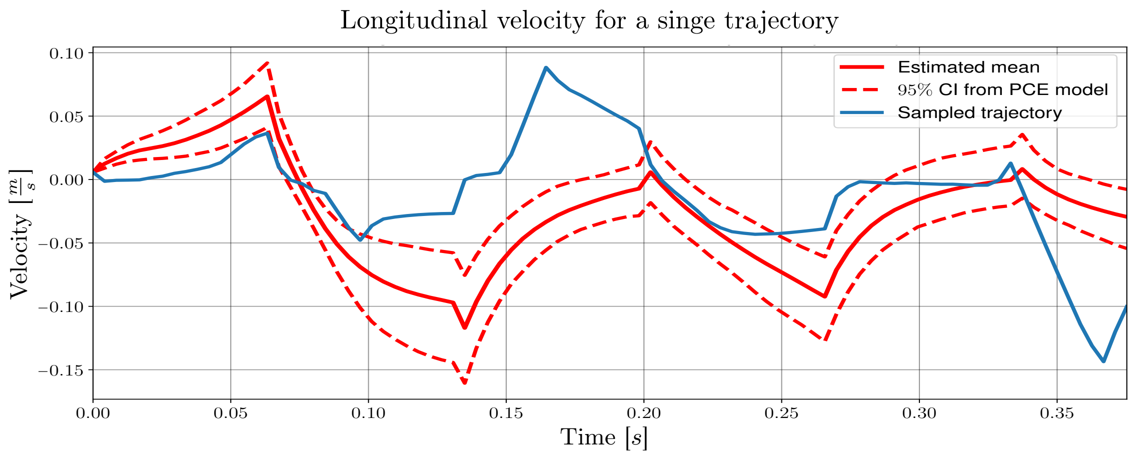


Figure 6.12: Estimation of one trajectory by the ME-PCE model of order three with seven divisions per uncertainty. The sampled trajectory is the trajectory with the same actions as used for estimation.

The presented results show that the ME-PCE model is not able to estimate the states of the robot with sufficient accuracy for the entire trajectory. The ME-PCE model cannot be used in combination with TRPO to train the policy. Another substantial problem with the implementation of the ME-PCE model is the number of samples and computational time needed to obtain an estimate. The current implementation (to estimate 0.4 s) needs around two hours to finish the calculation, so if the trajectory has a length of 6 s, the computation would take approximately 24 hours, which is unpractical. Because of the given two reasons, ME-PCE is not used in combination with TRPO.

6.2 Results MOE

The goal of this section is to verify that the estimate made by the MOE resembles the behaviour of the system. The desired accuracy for the model ensemble is not the same as for the PCE model: there is no accuracy for single estimates (as was given by the relative error in the previous section). That is because the stochastic system is represented by a number of NNs, so only if that number grows very large (around a thousand gives an accurate representation of the distribution), it is fair to compare the model ensemble for individual samples. The goal is to get a sufficiently accurate representation of the states and their respective distributions, in order to use the MOE in combination with TRPO. The focus is, again, on the velocity and position in the longitudinal direction of the base link, because they were found to be the hardest to estimate.

The first simulation is done with the only uncertain density and length, with the same sampled data used for PCE. The neural networks used for this estimate contained two hidden layers of size 200, with ReLU activation functions. There are 6 neural networks used per uncertainty, so a total of 36 networks are used for this estimate. That number has been chosen based on a quick evaluation of some different values (Figure A.16). In this figure, the normalised mean difference is given for 5 different numbers of NNs per uncertainty. Here, 6 NNs per uncertainty gave the best trade-off between accuracy (given as the normalised mean difference) and the computational time needed to fit the model. Although the MOE with 7 or 8 NNs per uncertainty is more accurate, it requires much more computational time, making it less practical. The sampled data was split into test and training data: 80% of the data was used for training and 20% for evaluation. With these settings, the model requires approximately two hours to fit the NNs.

The estimate of the velocity in longitudinal direction (Figure 6.13a) shows the general shape of the trajectory can be estimated quite well. The 95% confidence interval is similar in magnitude, but is, occasionally, significantly larger compared to the sampled data. That is because the model predicts states based on the current state, which is an estimate. The accumulation of errors results in, sometimes, significant fluctuations and differences between the estimated and sampled states.

Figure 6.13b gives an indication of the accuracy of the model. The relative difference between the data mean and the estimated mean is given. The average is approximately 0.5, which is relatively high, but also a pessimistic view of the performance. That is because the normalised mean difference peaks when the sampled state is near zero, in that case, the normalised difference is large. The data points of small magnitude are also less relevant when a least squares regression is used, because large terms dominate the error. That is a second reason the error is large around state values with a small magnitude.

The accumulation of errors is also observable in the estimate of the position in the longitudinal direction (Figure A.17a and A.17b). When looking at the top plot, for example, at 0.8 to 1.0 s, the estimated mean and confidence

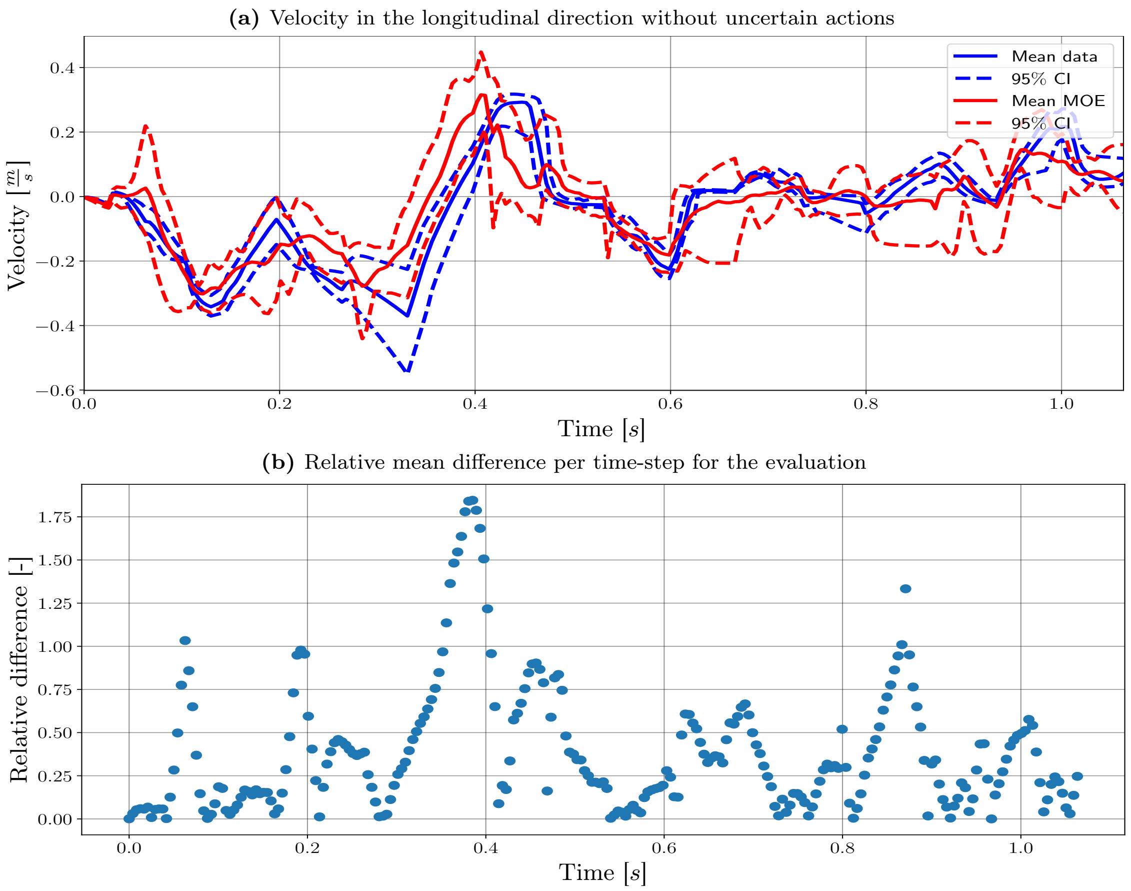


Figure 6.13: Estimated velocity in the longitudinal direction by the model ensemble. The top plot gives the trajectory and the bottom plot is the relative distance between the estimated mean and the data mean. The figures were made with the data used for evaluation.

interval are estimated less accurately compared to other parts of the trajectory. That is because the estimated velocity (Figure 6.13a), at that time, is also estimated incorrectly. That estimated velocity is also used to estimate the position, resulting in an accumulation of errors. However, in general, the position is estimated more accurately than the velocity. The normalised mean difference has an average value of approximately 0.5 and the estimated trajectory has, generally, a comparable mean and confidence interval with the sampled trajectory. It is therefore concluded that this model can estimate the trajectory, in this simplified environment.

The next step is to look at the model with uncertain density, length and actions. To get closer to the implementation with TRPO, a trained policy is used for some of the sampled trajectories. The data, therefore, contains a larger distribution of states, which is representative of the training situation. During training, the policy can, for some combinations of density and length, make the robot walk successfully, but for other combinations, the policy fails. The length of the trajectory increased to match the time window used for TRPO, which is 100 actions or approximately 6.5 s. The range of the density and length were also changed, to $0.4 - 0.6\ \frac{kg}{m^3}$ and $0.15 - 0.20\ m$ respectively. That is done to match the range of RVs used for training the universal policy with TRPO. 5000 samples were used for the estimate, with 80% used for fitting the neural networks and 20% used for evaluation. 6 models per uncertainty were used, so 36 neural networks for the total estimates. A NN contained two hidden layers of size 200, both of which are activated with ReLU activation functions. The results contain now a single sampled trajectory, but a distribution from the estimate. The actions from the sampled and estimated trajectory are the same, but the estimated trajectory has a confidence interval, due to uncertain density and length.

The estimate of the position in longitudinal direction can be seen in Figure 6.14a. The confidence interval is larger compared to earlier results, but that is due to the increased ranges for the density and the length. The normalised relative error (Figure 6.14b) is also much larger compared to the estimate done without uncertain actions. However, the estimated mean is showing similar behaviour compared to the sampled mean. Both figures indicate that the NN is able to predict the robot's dynamics. The velocity estimate (Figure A.18a) shows a similar result: the confidence interval is larger compared to the results with smaller ranges for density and length, but the estimated trajectory (with the confidence interval) is similar to the sampled trajectory

It is important to note that the results are not as consistent as the PCE model. Although the PCE model was inaccurate, changing the polynomial order or number of RVs did not result in unstable or unexpected results. The

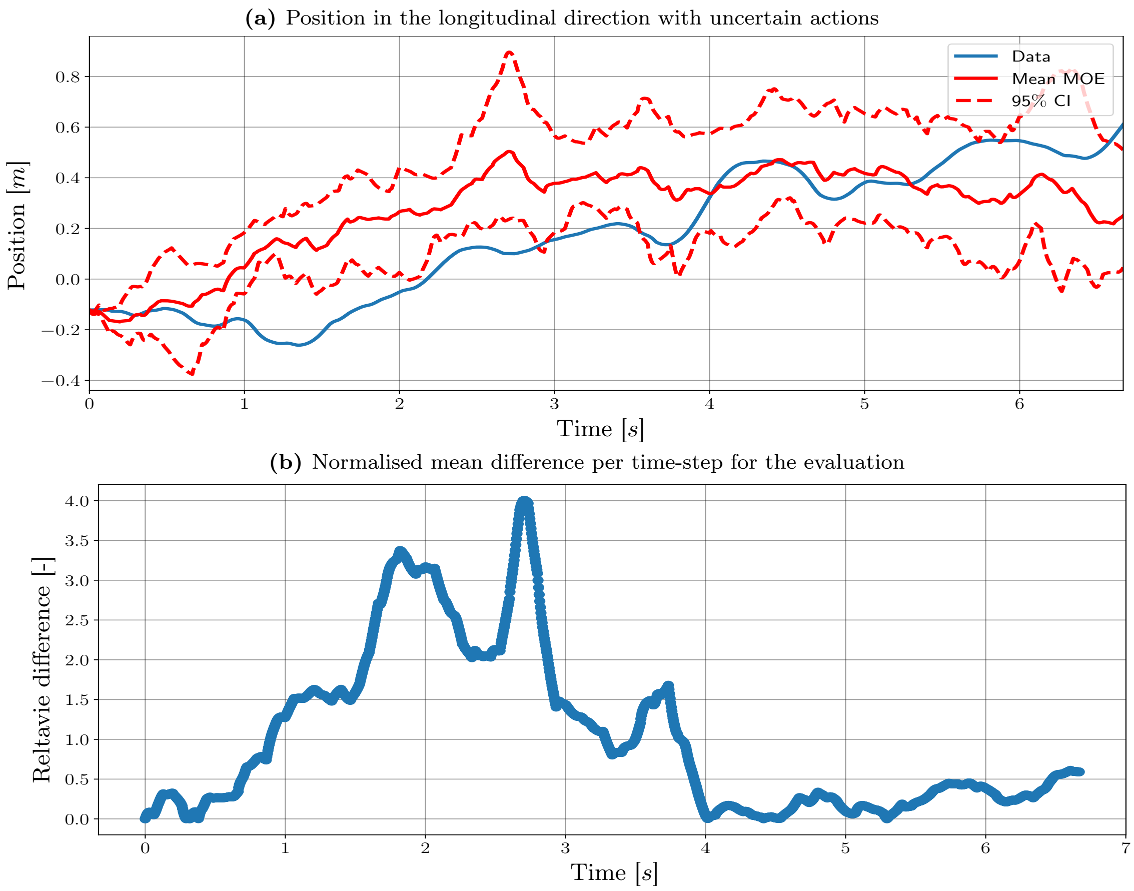


Figure 6.14: Estimation of the position in the longitudinal direction with varying actions by the model ensemble. The top plot is the estimated trajectory and the bottom plot is the relative distance between the estimated mean and trajectory. The figures were made with the data used for evaluation.

MOE does experience that behaviour much more: it is quite often that for some combinations of hidden layer size or number of samples used, the model is unstable or performing significantly worse. The MOE is much more dependent on hyperparameter tuning compared to PCE, which makes it difficult to get consistent results.

6.3 Results TRPO

The results are presented to verify two statements: whether it is possible to use a surrogate model in combination with TRPO and if it is possible to obtain a universal policy for this environment. The samples can be obtained from the original environment (DE/SE-TRPO), the TRPO algorithm with a ME-PCE surrogate (DE/SE-PCE-TRPO) or the model ensemble surrogate (DE/SE-MOE-TRPO). It was concluded that the ME-PCE model could not be used in combination with TRPO, therefore, there are no results with ME-PCE.

DE-TRPO and SE-TRPO are used as baselines to be compared with DE/SE-MOE-TRPO. The policies trained in both the DE and SE are evaluated in the SE. Here, the aim of DE-TRPO is to verify that a policy trained in a deterministic environment cannot perform well in a stochastic environment. SE-TRPO is added to see how the performance is compared with SE-MOE-TRPO and if the surrogate model affects the performance and training of the policy. The range of the RVs (length and density) is set at $l = U(0.15, 0.20)$ m and $\rho = U(0.4, 0.6)$ $\frac{kg}{m^3}$, which is increased from what was used for PCE. That is done to better evaluate how well the universal policy can perform in this environment. If the range is too small, a policy trained in a deterministic environment could also function well. The deterministic environment used the mean of those parameters, so $l = 0.175$ m and $\rho = 0.5$ $\frac{kg}{m^3}$. The batch size used for updating the policy is 16000 samples (the data from 10 trajectories) and the policy was trained for 600 epochs. A trajectory consists of 100 actions and is about 6 s long. This number of actions was chosen to make sure that the policy has enough time to learn the cyclic behaviour needed for walking.

To get an idea of the performance of the policy, it is important to know how the reward and variation in length and density relate to the observed performance. In Figures A.19a, A.19b and A.19c, the robot with three different link lengths is given. It is clear that the highest link length results in a significantly larger robot. It is also important to note that a larger robot is heavier, since the mass of a link is the density multiplied by its size. To give an idea of what the trajectories for various rewards look like, the position and velocity in the longitudinal direction are presented for three trajectories, each from a different range of rewards. A cumulative reward of 200 (Figure

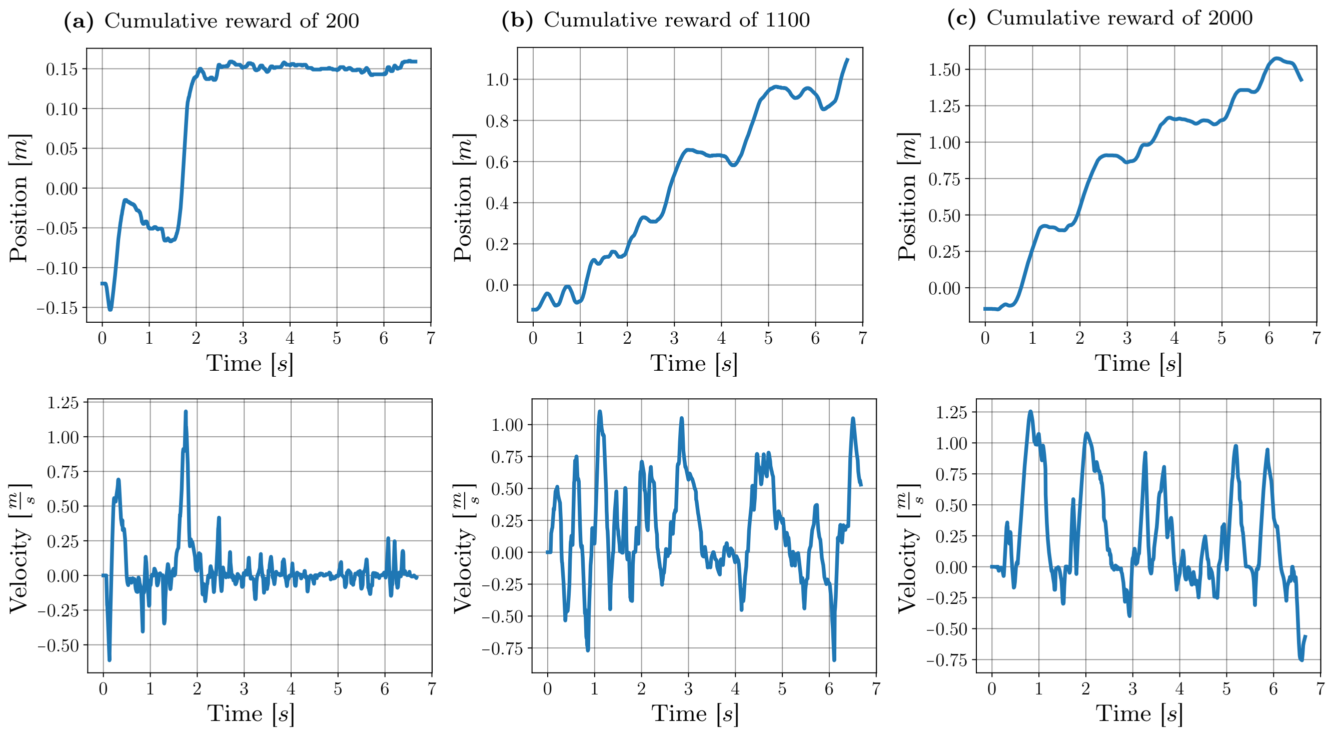


Figure 6.15: The position and velocity in the longitudinal direction for three different trajectories. Each trajectory resembles the performance typical for that range of the cumulative reward.

6.15a shows the position and velocity in the longitudinal direction) means that the robot cannot walk well: in the beginning of the trajectory, two steps are made, resulting in a small advance of the base link. After the two steps, the robot fails to walk any further. This is typical behaviour for trajectories with a cumulative reward of around 200: the robot can make one or two steps, but is not able to make consecutive steps. The states from a trajectory with a cumulative reward of 1100 can be seen in Figure 6.15b. In this trajectory, the robot makes consecutive steps for the entire length of the trajectory. However, the robot makes small steps at the beginning of the episode and it needs some time to 'start up'. Trajectories with a cumulative reward of around 1100 show a robot that can walk, but occasionally the robot makes small steps or does not move forward for a longer amount of time. Finally, a cumulative reward of 2000 (Figures 6.15c) is a trajectory where the robot makes large consecutive steps. The highest obtained reward is around 3000 and if the reward is above 1300, the robot is considered to be able to walk.

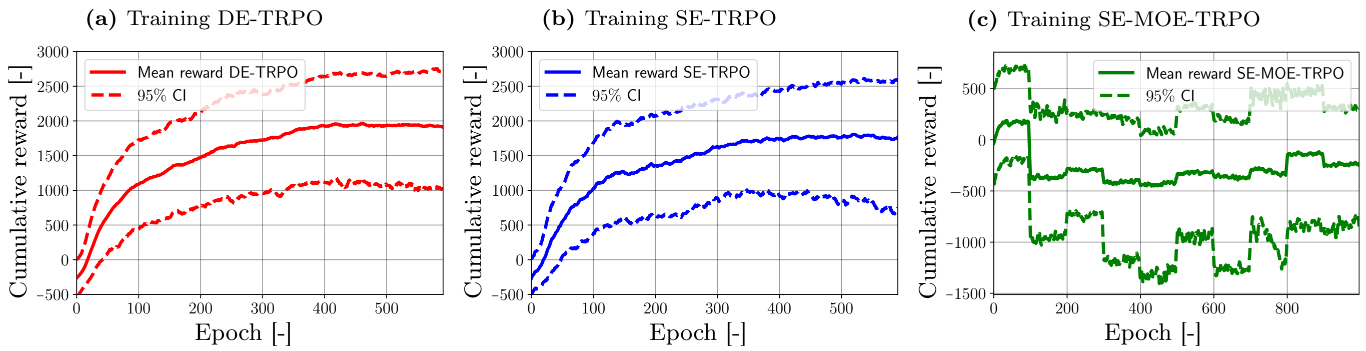


Figure 6.16: Cumulative reward of an episode per epoch, during training of the policy with the three implementations of TRPO.

First, the results from training the policies are discussed. In Figures 6.16a, 6.16b the cumulative reward per episode can be seen for the training of the policy (600 epochs). Both plots were made with 20 training cycles and every epoch was evaluated for 5 trajectories. The data has been smoothed with a moving average filter with a window size of 10. The mean reward of DE-TRPO converges to around 1900, with some maximum values around 2700 and the lowest values around 1000. It shows that the policy can master the environment quite well, since an average cumulative reward of 1900 means that the robot can walk well. Looking at the training for SE-TRPO (Figure 6.16b), the confidence interval is larger, which makes sense, since there are now also robots, that are less ideal for walking: short legs and heavy links. The mean is therefore also lower: around 1800. What can be observed is that the training does take a similar number of epochs, so it seems that the environment has not become much more complex due to the variation.

In Figure 6.16c, the cumulative reward per episode from training a policy with the MOE can be seen. The reward shown is obtained from the MOE. The policy is trained for 1000 epochs and the MOE is updated every 100 epochs. The interval of 100 epochs was chosen based on the results of DE/SE-TRPO: in 100 epochs, the policy is able to improve its performance already significantly. The results are presented from 20 training cycles. The parameters of the policy were updated with a batch size consisting of ten trajectories. The cumulative rewards from the MOE are not improving during the training: both the mean and confidence interval are almost constant during the 100 epochs. An interesting observation is that the policy is able to improve (although only slightly) at the beginning of the training cycle, for about the first 50 epochs. It is strange that the policy is not able to improve anymore after those epochs or when the MOE is fit again. This is most likely caused by hyperparameters, which are not optimal for the training with the MOE, and the MOE not having sufficient accuracy.

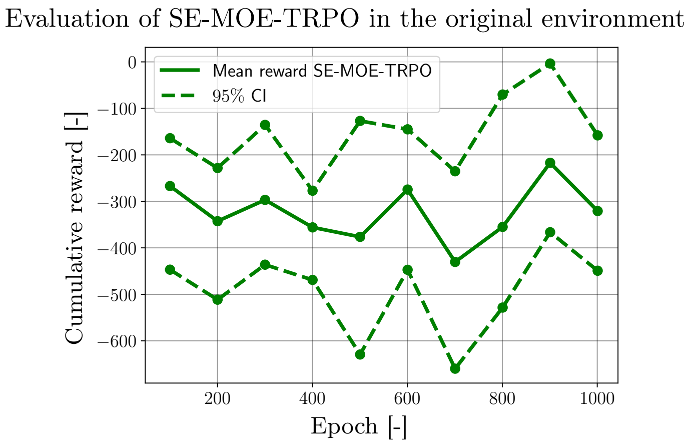


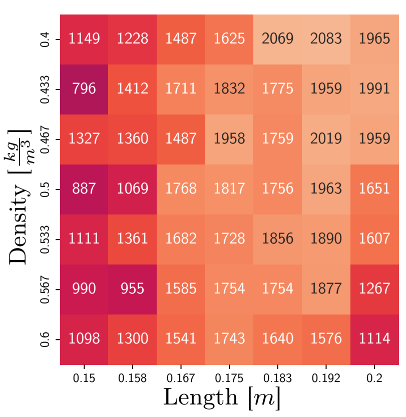
Figure 6.17: Evaluation of twenty policies trained with SE-MOE-TRPO in the original environment. The evaluation was done every 100 epochs and for 10 trajectories per policy.

When using a surrogate model, it is also relevant to verify the performance of the policy in the original environment. The policy trained with SE-MOE-TRPO, does not perform well in the original environment. This can be seen in Figure 6.17, where the policy is evaluated in the original environment every 100 epochs. Every time, one policy is evaluated for 100 trajectories. The data of 20 training iterations is given in this figure. It can be seen that the rewards are all low, meaning that none of the policies have succeeded to learn walking. What is more interesting, is that the confidence interval and mean of the evaluation are different compared to the confidence interval during training, in the MOE. The confidence interval from the cumulative rewards from the MOE is generally, significantly larger compared to the confidence interval from the original environment. This means that the dynamics, predicted by the MOE, are significantly different compared to the dynamics of the original environment. Another observation are the jumps in cumulative reward, which happen occasionally when the MOE is fit again (every 100 epochs). These jumps suggest that the MOE behaves significantly differently after it has been fit again, which should not be the case. The MOE does not seem to be able to consistently predict the dynamics correctly.

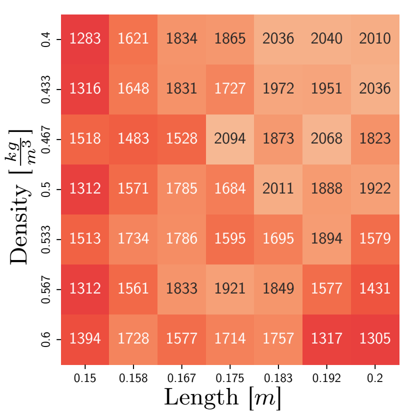
The same observations can be made when looking at the results of DE-MOE-TRPO (Figures A.20a and A.20b). The policy is not able to improve during the 100 epochs and the predictions by the MOE are not consistent. Simplifying the environment (by removing the stochastic behaviour) does not improve the learning of the policy, so it is not the case that the SE-MOE is too complicated for the policy to master, but that there is another problem with the implementation of the algorithm.

The evaluation of the policies is given in Figures 6.18a, 6.18b and 6.18c. It shows the average cumulative reward of an episode, evaluated for the twenty policies, for various combinations of density and length. Starting with the middle figure, for SE-TRPO. The heatmap can be divided into three sections. In the top right corner are the parameters of the robot, which are favourable: low density and long links. For these variations, the highest rewards are obtained: all around 2000, which means that the policy performs really well with those variations. Looking at the first two columns, those rewards are significantly lower: those are the variations of robots with the short links. The maximum reward is therefore lower, since the robot is not able to move as fast. This effect will be corrected later in this section. A third interesting region in the heatmap is in the bottom right corner, where the heaviest variations are (long links and high density). Here, it can be seen that rewards are also lower, which makes sense, since the actuators are not able to move the links as fast as with the other variations.

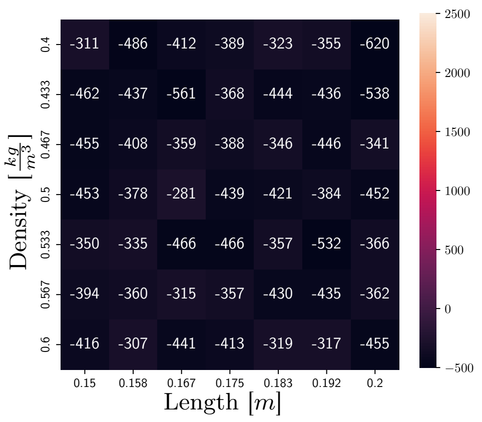
The policy trained with DE-TRPO (Figure 6.18a), performs as expected around its training point (density of 0.5 $\frac{kg}{m^3}$ and a length of 0.175 m). It gives an average reward of 1800, which is close to the average reward from training (which was around 1900). For the preferred variations of the robot (top-right corner of the heatmap), the policy performs as well as the policy trained with SE-TRPO. But, looking at the robots with short links, the policy is



(a) Cumulative reward for DE-TRPO



(b) Cumulative reward of SE-TRPO

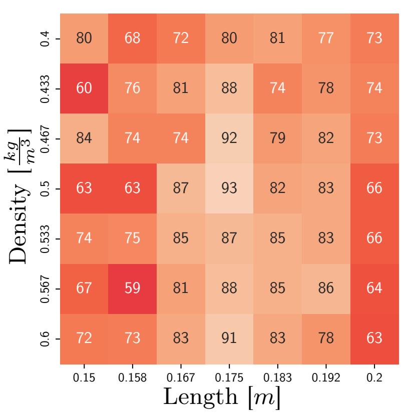


(c) Cumulative reward of SE-MOE-TRPO

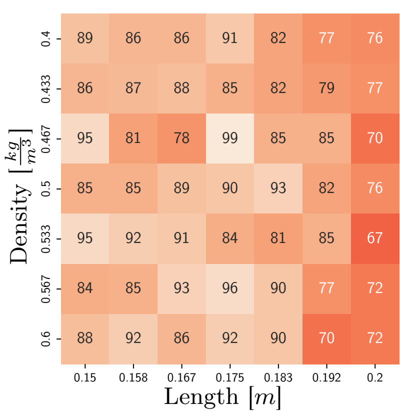
Figure 6.18: Evaluation of the policies. The heatmap shows the cumulative reward for various values of length and density. The average reward of twenty policies is given (for all three figures).

significantly worse compared to the policy trained SE-TRPO. It can be concluded that training with SE-TRPO results in a policy that is much more consistent in performance for the given variations compared to a policy trained with DE-TRPO.

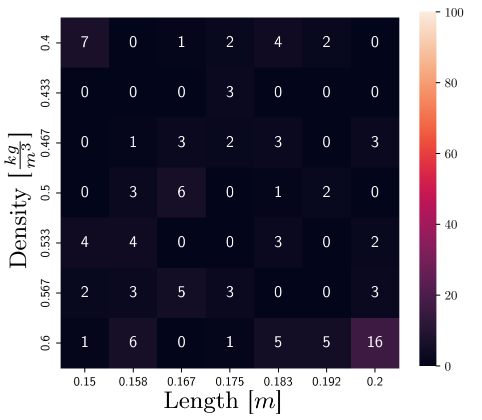
Finally, the policies trained with SE-MOE-TRPO are not able to obtain a high cumulative reward for any of the combinations of density and length (Figure 6.18c), which is not surprising, as the policy was not able to improve during training. The same applies to the policy trained with DE-MOE-TRPO (Figure A.20c)



(a) Cumulative reward for DE-TRPO



(b) Cumulative reward of SE-TRPO



(c) Cumulative reward of SE-MOE-TRPO

Figure 6.19: Evaluation of the policies. The heatmap shows the relative performance compared to the maximal performance possible for that combination of density and length, as a percentage value. The average cumulative reward of twenty policies is given.

The cumulative reward from one episode is not only dependent on the performance of the policy, but also on the robot: some robots can reach higher rewards with, for example, longer legs. Therefore, only comparing the cumulative reward from one episode does not give a complete picture of the performance of a policy. The evaluation is improved by correcting the obtained reward from a policy with the maximum reward that can be reached with that combination of density and length. If that is done, it is possible to give the performance of a policy as a percentage of the maximum achievable cumulative reward. The maximal possible reward is obtained by training a policy with that configuration of density and length (so in a DE) and using the obtained maximum cumulative reward. Dividing the cumulative reward from the policies trained with DE/SE-TRPO and SE-MOE-TRPO by that maximum cumulative reward gives the relative performance of those policies. It is presented as a percentage value. In Figures 6.19a, 6.19b and 6.19c those results are given. Here, it can be observed that the policy trained with DE-TRPO, has the highest percentages in the center. That makes sense, since it was trained with that value of density and length. The percentages drop for the combinations further from the centre, since the dynamics of the robot are also changing more and the policies have not encountered that during training.

The performance of the policies trained with SE-TRPO (Figure 6.19b) is much more consistent over the entire range. Only for the robots with the highest density, the policies perform worse, although better compared to the policies trained with DE-TRPO. A reason for this can be that the dynamics of the robot are significantly different when the legs are long, requiring a different control strategy for effective walking. These results confirm that the policy trained with SE-TRPO performs better compared to the policy trained with DE-TRPO. The performance of SE-MOE-TRPO is, not surprisingly, bad.

The results of TRPO with the original environment (in SE and DE) show that one policy can control the stochastic behaviour of the dynamics. The dynamics predicted by the MOE are not similar enough to the dynamics of the original environment, such that a policy can be trained with them.

7 Discussion and recommendations

The discussion is separated into three parts: first about the PCE model and its accuracy, followed by the model ensemble. In the final part, TRPO is discussed in combination with the models. Also, some recommendations are given for future work.

7.1 The PCE Model

The main point of discussion is the high relative error for both the model with and without uncertain actions. It was found that using the multi-element approach, the relative error and JSD improved significantly, but the target accuracy was still not reached. There can be two reasons why the accuracy is low: the first reason is that the distribution one PCE has to describe is still too complex and that more divisions of the uncertainties, so more local PCEs in the multi-element, are needed. Extrapolating the results from this thesis, the accuracy would, presumably, increase, but the model would require an unpractical amount of time to be calculated. A second reason why the model is inaccurate is due to the implementation, which might not be the optimal method and can therefore negatively affect the results.

The reduction of the RVs in the PCE model did not work for the position of the robot. That is because there is no information stored in the model about actions that happened before the action window. This could be solved by using the current state as input for the PCE model, but the difficulty is incorporating that state into the PCE model. A simpler solution would be to only predict the velocity of the robot with PCE and calculate the position using a numerical time-integration scheme. The results suggested that the velocity of the robot can be estimated with the reduction step. The disadvantage of predicting only the velocity with PCE is that the position has an additional error due to integration and, together with the prediction error of the velocity, the error in the estimate of the position can become large. For this method to function, it is required that the velocity is estimated accurately. An advantage of this method is that the PCE model only estimates velocities, reducing the computational power and the number of samples needed.

A third point of discussion is the implementation of the actions, which are now implemented as joint target positions. This type of action has been chosen, because they are implemented in RoboGrammar like this and, more importantly, it was not possible to obtain the torque values from the RoboGrammar environment. By modelling the actions as a target position, the dynamics the model has to predict become more complicated (there is an additional feedback loop), so it is better to use the torque of the actuators as a model input, such that only the dynamics of the robot are modelled, simplifying the model. One disadvantage of modelling torque is that every time-step new RVs are added to the PCE model (instead of every control interval, which is the case in the current implementation), so it can be that the number of RVs in the PCE model increases when using torque instead of joint target positions as RVs for the model.

A final, practical recommendation is to use C++ as the programming language, since it is much faster and more efficient compared to Python.

7.2 The MOE

The MOE was able to predict the dynamics of the environment, but not with high accuracy. That is partly due to the fact that estimated states are used to estimate the new states, so there is an accumulation of error. This could be solved by approaching the MOE similarly to the PCE model. With the PCE surrogate, every time-step was estimated by a set of coefficients. A similar thing could be done with the MOE: every time-step is estimated by a NN. That NN would have as inputs the initial state and all executed actions. So, instead of one NN used for estimating the dynamics of the environment, one NN is used for estimating the state at one point in time. There is no accumulation of errors anymore, since all time-steps are estimated independently. The downside of such a method is the number of NNs to fit and, therefore, the computational time needed. Another variation of modelling the dynamics with the MOE is to use a physics informed model. Such a model is based on the dynamical equation describing the system, but some elements in that dynamical equation are described by a NN. In that case, the behaviour one NN has to predict is limited, resulting in more stable and accurate estimations of the states.

The accuracy of the MOE could also be improved by modelling all states of the robot: only a selection of the states is considered in this work. However, that might be too limited for the model to predict the states correctly. That is because the state of one link is dependent on all links and joints in the robot. Adding all the states to the model would significantly increase the complexity of the MOE: the input increases from 18 states to 84 states, but this increase results, mainly in longer training times for the MOE. The time required to sample from the MOE would increase that much, since that is a relatively simple procedure.

Another problem with the MOE is that the prediction is very dependent on hyperparameters and is unstable for some combinations of those. The results are therefore not always consistent: it can be that the estimation of the

MOE is accurate for one trajectory, but significantly worse for another. There is a substantial need for optimisation of the hyperparameters, which was not as relevant for the PCE model. Also, the implementation (especially the least squares estimate) can have a large effect on the performance of the MOE.

7.3 TRPO

The results with TRPO and the original environment showed that it is possible to train one policy that functions well in a stochastic environment. The training results were consistent: all simulated training cycles were able to train the policy that is able to make the robot walk. A difficulty is the hyperparameter tuning, which is of great importance for the training, as is also highlighted in [55, 56]. The implementation is also of importance: some implementations of a RL algorithm perform significantly worse than others, even if the same theoretical algorithm is used. The combination of hyperparameters and implementation can result in the fact that for some environments (for example, the original RoboGrammar environment) the algorithm works well and is able to give consistent results, but for others (for example, the MOE) not. This is a problem that is observed more often with RL.

The main reason that TRPO in combination with the MOE is not able to learn is probably due to the tuning of the hyperparameters. Both the parameters of the TRPO algorithm and the parameters of the fitting of the MOE must be chosen correctly. The difficulty is that there are a lot of parameters involved and the combinations of parameters that result in successful training can be limited. This can only be solved by optimising the parameters or by trial and error. It would be beneficial to change the MOE in combination with TRPO, so that it is more robust to the hyperparameters, making results consistent and more easily reproducible.

8 Conclusion

In this thesis, two types of surrogate models are developed to train a universal policy with TRPO. Although both models do not have the desired accuracy, there are still possibilities to improve the accuracy. The PCE model is extensively evaluated and is, for some of the estimated states, accurate. The difficulty lies in obtaining a practical implementation of this model. The effect of implementing a multi element PCE improves the accuracy significantly and shows that the dynamics are too complicated to be captured by a single PCE. It is found that it is possible to further improve the performance of the model using the implementation of this thesis, but that results in a model that requires too much computational power for an implementation in combination with TRPO. It is for that reason that the PCE model is not used in combination with TRPO.

The model ensemble surrogate has lower accuracy compared to the PCE model, but is simpler to implement and requires significantly less computational time. The difficulty with this model is setting the correct hyperparameters, so that a stable and accurate model can be obtained.

The training of the universal policy with TRPO is successful in the original RoboGrammar environment. Here was a clear result: the policy trained in the stochastic environment is able to perform similarly over the entire range of the two random variables: density and length. The policy trained in the deterministic environment performed significantly worse when evaluated for the entire range of density and length, proving that the policy trained with one value for density and length, cannot control the entire range equally well. So, a universal policy can be trained for this type of environment.

The policy trained with the model ensemble surrogate is not able to improve its performance in the environment, but this can be improved by optimising the hyperparameters of the algorithm. It cannot be concluded that a surrogate model can be used for training instead of the original environment, as the training with the model ensemble did not yield a successful policy.

The main goal of this thesis is to develop a foundation for training a universal policy with a surrogate model. That goal is not yet fully achieved, but when it is accomplished to obtain accurate surrogate models, they can be used in combination with TRPO. After that, it is possible to extend this work to a larger variation of robot morphologies from the RoboGrammar library and develop a surrogate model for those robots, to finally reach a universal policy that can control a variety of robot morphologies.

References

[1] K. J. Åström, *Introduction to stochastic control theory.* Courier Corporation, 2012.

[2] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei, “Metamorph: Learning universal controllers with transformers,” *arXiv preprint arXiv:2203.11931*, 2022.

[3] E. Tzorakoleftherakis, “Reinforcement learning: A brief guide.” [Online]. Available: https://nl.mathworks.com/company/technical-articles/reinforcement-learning-a-brief-guide.html

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] P. Kormushev, S. Calinon, and D. G. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges,” *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[6] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[7] W. Huang, I. Mordatch, and D. Pathak, “One policy to control them all: Shared modular policies for agent-agnostic control,” in *International Conference on Machine Learning.* PMLR, 2020, pp. 4455–4464.

[8] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[9] J. M. D. Delgado and L. Oyedele, “Robotics in construction: A critical review of the reinforcement learning and imitation learning paradigms,” *Advanced Engineering Informatics*, vol. 54, p. 101787, 2022.

[10] D. Nguyen-Tuong and J. Peters, “Model learning for robot control: a survey,” *Cognitive processing*, vol. 12, pp. 319–340, 2011.

[11] V. Kurin, M. Igl, T. Rocktäschel, W. Boehmer, and S. Whiteson, “My body is a cage: the role of morphology in graph-based incompatible control,” *arXiv preprint arXiv:2010.01856*, 2020.

[12] S. Ha, J. Kim, and K. Yamane, “Automated deep reinforcement learning environment for hardware of a modular legged robot,” in *2018 15th international conference on ubiquitous robots (UR).* IEEE, 2018, pp. 348–354.

[13] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[14] J. Hyde and P. Engel, “Investing in a robotic milking system: A monte carlo simulation analysis,” *Journal of dairy science*, vol. 85, no. 9, pp. 2207–2214, 2002.

[15] B. V. Rosić and J. H. Diekmann, “Methods for the uncertainty quantification of aircraft simulation models,” *Journal of Aircraft*, vol. 52, no. 4, pp. 1247–1255, 2015.

[16] R. Ghanem and J. Red-Horse, “Polynomial chaos: modeling, estimation, and approximation,” *Handbook of uncertainty quantification*, vol. 1, p. 3, 2017.

[17] K.-K. K. Kim, D. E. Shen, Z. K. Nagy, and R. D. Braatz, “Wiener’s polynomial chaos for the analysis and control of nonlinear dynamical systems with probabilistic uncertainties [historical perspectives],” *IEEE Control Systems Magazine*, vol. 33, no. 5, pp. 58–67, 2013.

[18] R. Bhusal and K. Subbarao, “Uncertainty quantification using generalized polynomial chaos expansion for nonlinear dynamical systems with mixed state and parameter uncertainties,” *Journal of Computational and Nonlinear Dynamics*, vol. 14, no. 2, p. 021011, 2019.

[19] B. Sudret, “Global sensitivity analysis using polynomial chaos expansions,” *Reliability engineering & system safety*, vol. 93, no. 7, pp. 964–979, 2008.

[20] E. D. Blanchard, A. Sandu, and C. Sandu, “A polynomial chaos-based kalman filter approach for parameter estimation of mechanical systems,” *Journal of Dynamic Systems, Measurement, and Control*, 2010.

[21] G. Kewlani and K. Iagnemma, “A multi-element generalized polynomial chaos approach to analysis of mobile robot dynamics under uncertainty,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2009, pp. 1177–1182.

[22] P. Ryan, “Automating the analysis of uncertainties in multi-body dynamic systems using polynomial chaos theory,” Ph.D. dissertation, Marquette University, 2018.

[23] A. Sandu, C. Sandu, and M. Ahmadian, "Modeling multibody systems with uncertainties. part i: Theoretical and computational aspects," *Multibody System Dynamics*, vol. 15, pp. 369–391, 2006.

[24] L. Wang and G. Yang, "An interval uncertainty propagation method using polynomial chaos expansion and its application in complicated multibody dynamic systems," *Nonlinear Dynamics*, vol. 105, no. 1, pp. 837–858, 2021.

[25] S. Sabet and M. Poursina, "Forward kinematic analysis of non-deterministic articulated multibody systems with kinematically closed-loops in polynomial chaos expansion scheme," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 57168. American Society of Mechanical Engineers, 2015, p. V006T10A017.

[26] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[27] A. Zhao, J. Xu, M. Konaković-Luković, J. Hughes, A. Spielberg, D. Rus, and W. Matusik, "Robogrammar: graph grammar for terrain-optimized robot design," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–16, 2020.

[28] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning.* PMLR, 2015, pp. 1889–1897.

[29] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[31] M. A. Johnson and M. H. Moradi, *PID control.* Springer, 2005.

[32] P. R. Krishnaswamy, G. P. Rangaiah, R. K. Jha, and P. B. Deshpande, "When to use cascade control," *Industrial & engineering chemistry research*, vol. 29, no. 10, pp. 2163–2166, 1990.

[33] M. A. Siddiqui, M. Anwar, S. Laskar, and M. Mahboob, "A unified approach to design controller in cascade control structure for unstable, integrating and stable processes," *ISA transactions*, vol. 114, pp. 331–346, 2021.

[34] R. Featherstone, *Rigid body dynamics algorithms.* Springer, 2014.

[35] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann *et al.*, "Kubric: A scalable dataset generator," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3749–3761.

[36] J. C. Butcher, "On the implementation of implicit runge-kutta methods," *BIT Numerical Mathematics*, vol. 16, no. 3, pp. 237–240, 1976.

[37] M. Grigoriu, *Essentials of Probability Theory.* Springer, 2014.

[38] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[39] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[40] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning.* Pmlr, 2014, pp. 387–395.

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

[42] J. Romoff, P. Henderson, A. Piché, V. Francois-Lavet, and J. Pineau, "Reward estimation for variance reduction in deep reinforcement learning," *arXiv preprint arXiv:1805.03359*, 2018.

[43] S. T. Tokdar and R. E. Kass, "Importance sampling: a review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[44] S. Kullback, "Information theory and statistics—dover publi," *Inc., NY*, 1968.

[45] M. Grigoriu, *Monte Carlo simulation.* Springer, 2014.

[46] O. Pajonk, B. V. Rosić, and H. G. Matthies, "Sampling-free linear bayesian updating of model state and parameters using a square root approach," *Computers & Geosciences*, vol. 55, pp. 70–83, 2013.

[47] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables.* US Government printing office, 1968, vol. 55.

[48] W. Dijkhof, “Solving stochastic differential equations using the polynomial chaos decomposition,” *Technical Report 2001.043*, 2001.

[49] T. Albaraghtheh, “Polynomial chaos expansion — uncertainty quantification.” [Online]. Available: https://dictionary.helmholtz-uq.de/content/PCE.html

[50] S. P. Ellis, “Instability of least squares, least absolute deviation and least median of squares linear regression, with a comment by stephen portnoy and ivan mizera and a rejoinder by the author,” *Statistical Science*, vol. 13, no. 4, pp. 337–350, 1998.

[51] P. Prempraneerach, F. S. Hover, M. S. Triantafyllou, and G. E. Karniadakis, “Uncertainty quantification in simulations of power systems: Multi-element polynomial chaos methods,” *Reliability Engineering & System Safety*, vol. 95, no. 6, pp. 632–646, 2010.

[52] X. Wan and G. E. Karniadakis, “Multi-element generalized polynomial chaos for arbitrary probability measures,” *SIAM Journal on Scientific Computing*, vol. 28, no. 3, pp. 901–928, 2006.

[53] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[54] I. Kostrikov, “Pytorch implementation of trpo,” https://github.com/ikostrikov/pytorch-trpo/tree/master, 2018.

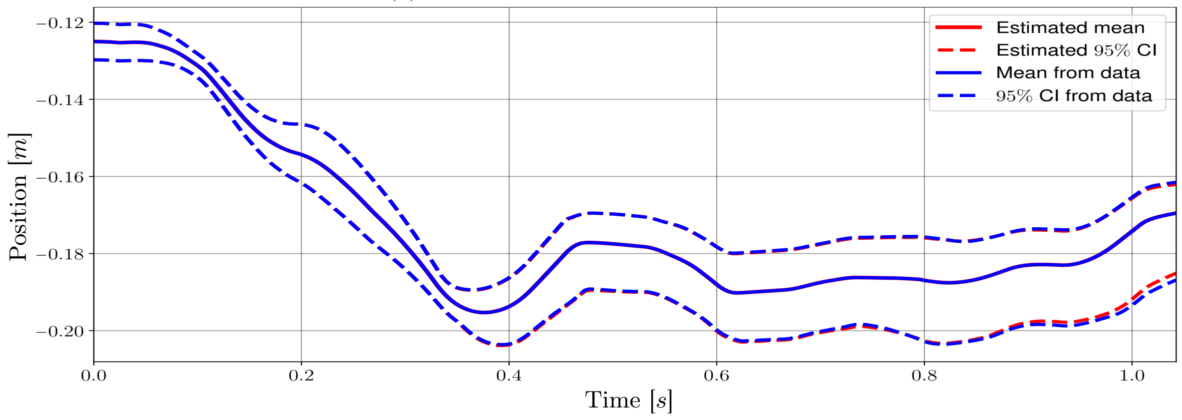
[55] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, “Reproducibility of benchmarked deep reinforcement learning tasks for continuous control,” *arXiv preprint arXiv:1708.04133*, 2017.

[56] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra, “On the importance of hyperparameter optimization for model-based reinforcement learning,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 4015–4023.

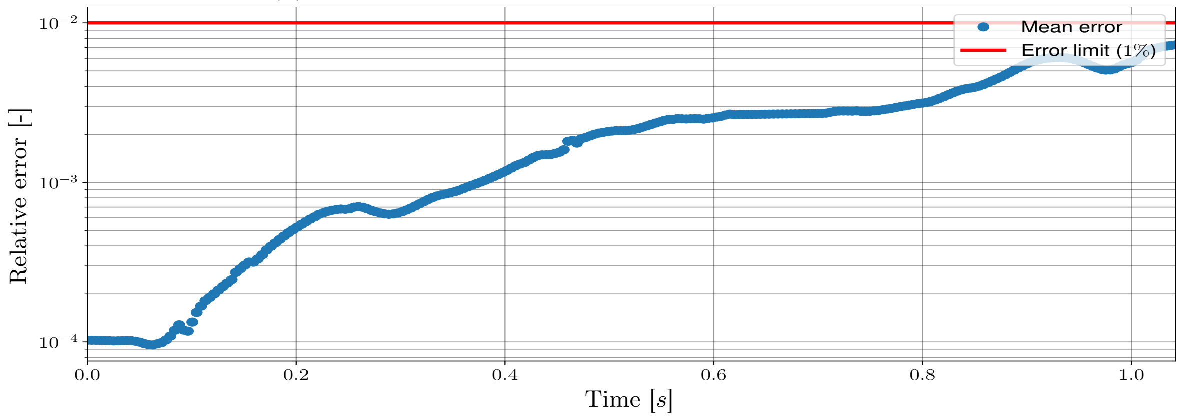
A Appendix

A.1 PCE Results with Uncertain Mass and Density

(a) Position in the longitudinal direction



(b) Relative error per time-step of evaluation of the PCE model



(c) Jensen-Shannon divergence of the fit and evaluation

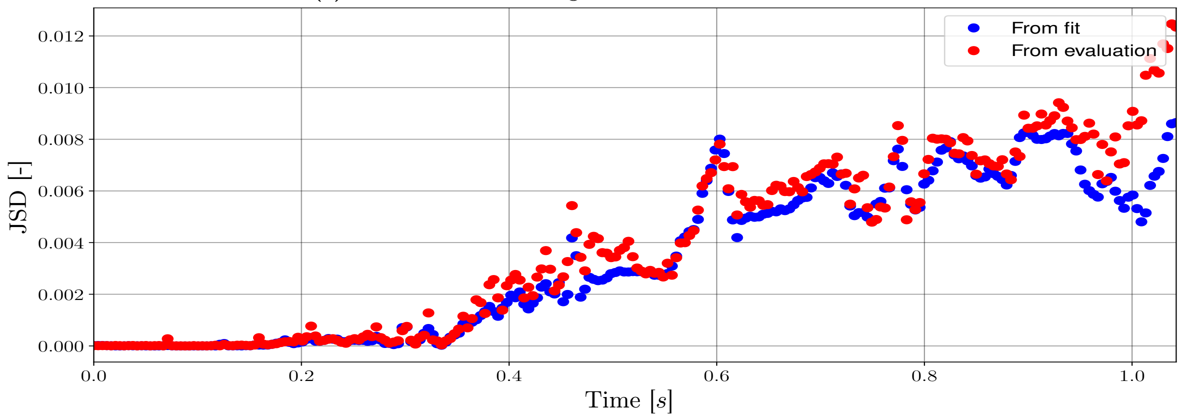


Figure A.1: Estimation of the position in the longitudinal direction by the PCE model with polynomial up to and including order 15. The top plot shows the trajectory, the middle plot the mean relative error of the estimation. The bottom plot gives the JSD for the estimate with the data used for fitting and evaluation.

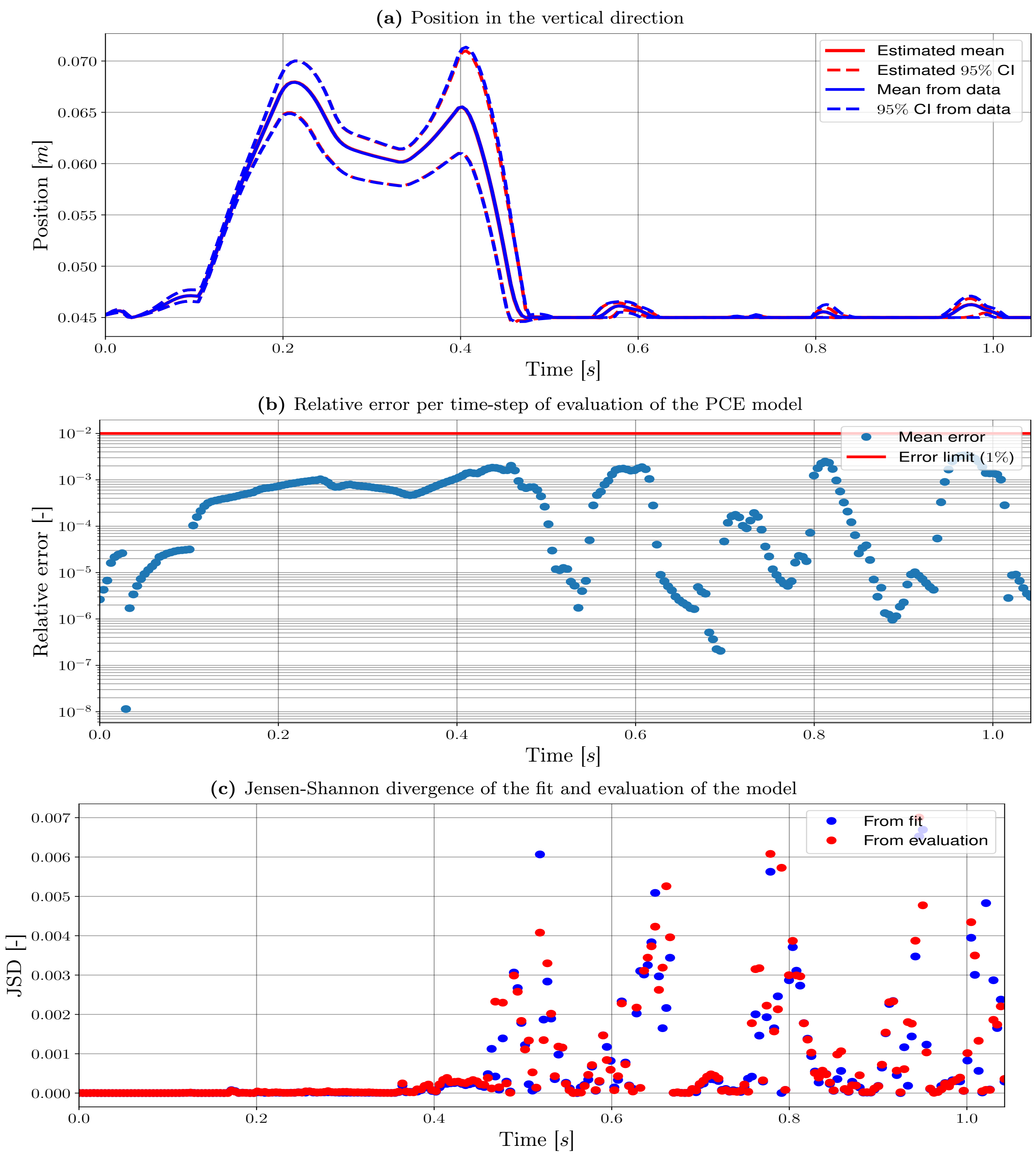


Figure A.2: Estimation of the position in the vertical direction by the PCE model with polynomial up to and including order 15. The top plot shows the trajectory, the middle plot the mean relative error of the estimation. The bottom plot gives the JSD for the estimate with the data used for fitting and evaluation.

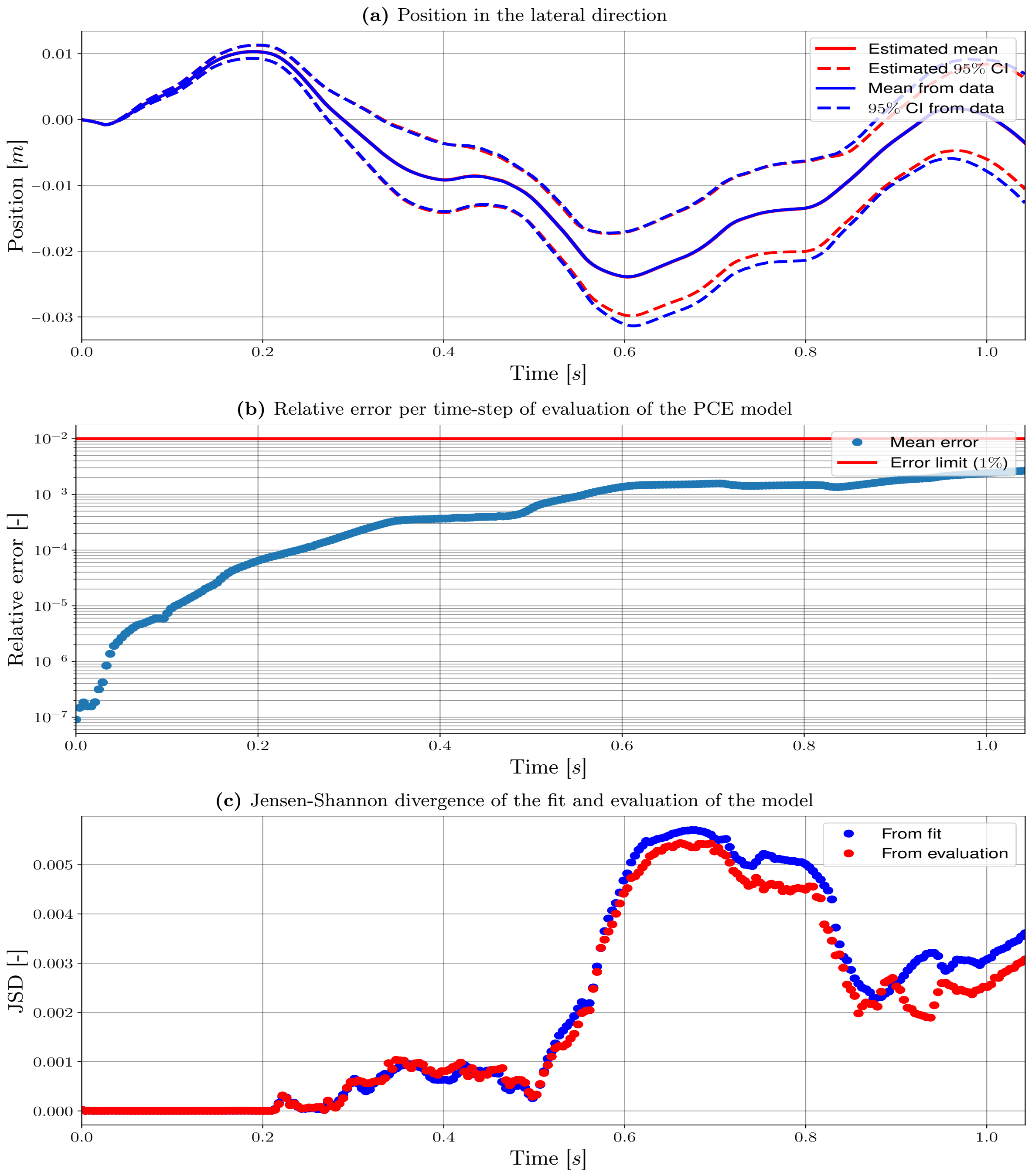


Figure A.3: Estimation of the position in the lateral direction by the PCE model with polynomial up to and including order 15. The top plot shows the trajectory, the middle plot the mean relative error of the estimation. The bottom plot gives the JSD for the estimate with the data used for fitting and evaluation.

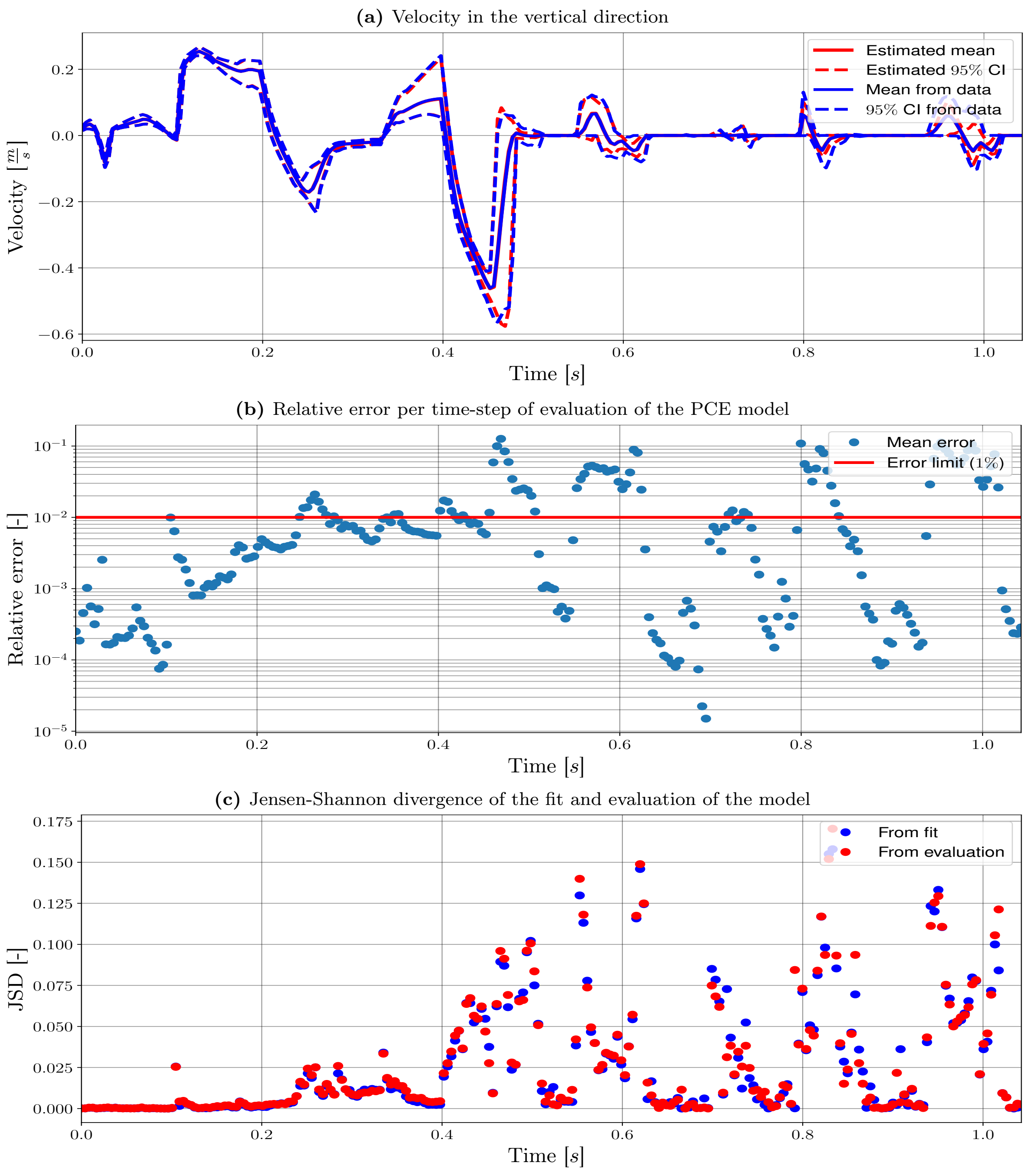


Figure A.4: Estimation of the velocity in the vertical direction by the PCE model with polynomial up to and including order 15. The top plot shows the trajectory, the middle plot the mean relative error of the estimation. The bottom plot gives the JSD for the estimate with the data used for fitting and evaluation.

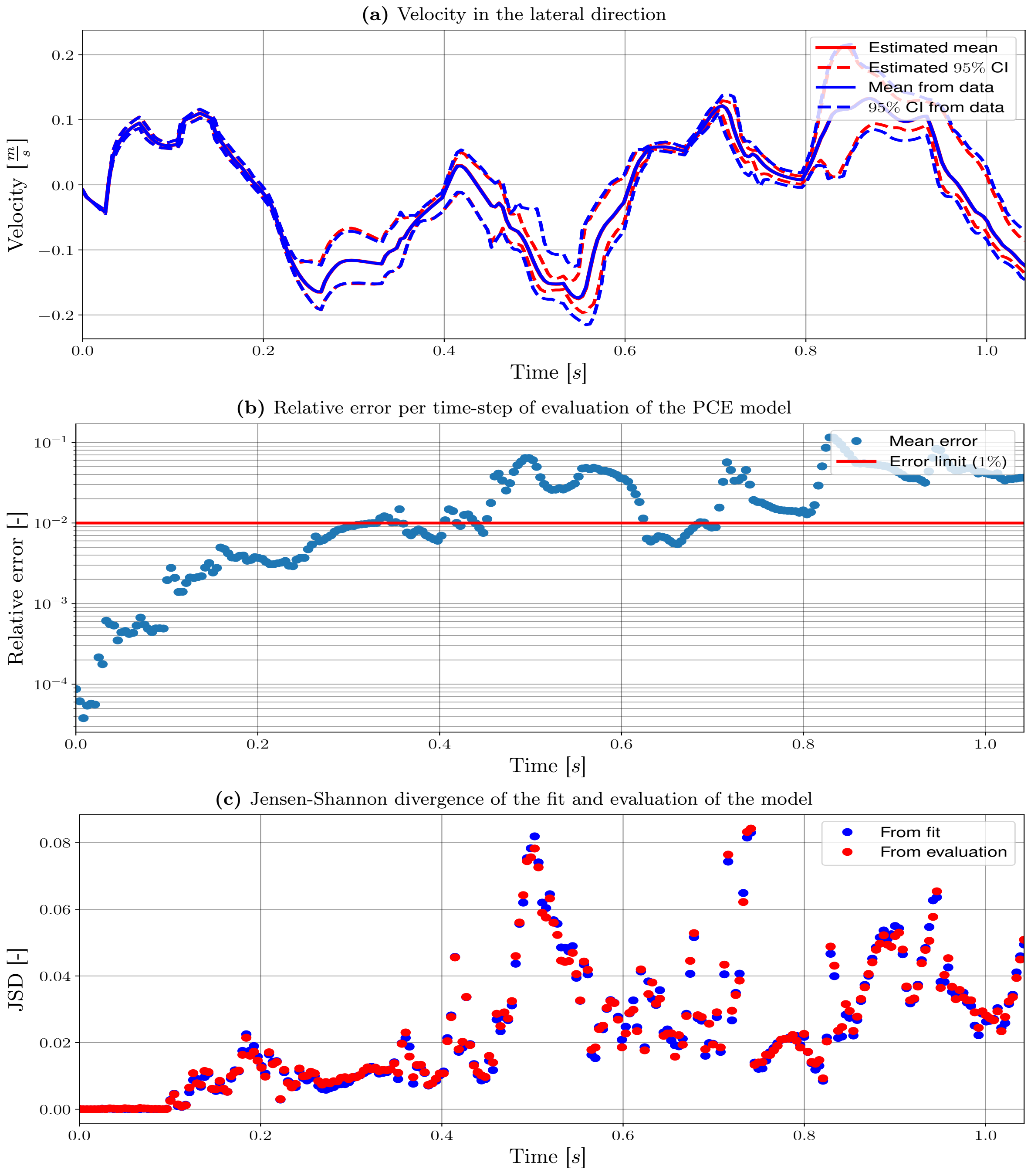


Figure A.5: Estimation of the velocity in the lateral direction by the PCE model with polynomial up to and including order 15. The top plot shows the trajectory, the middle plot the mean relative error of the estimation. The bottom plot gives the JSD for the estimate with the data used for fitting and evaluation.

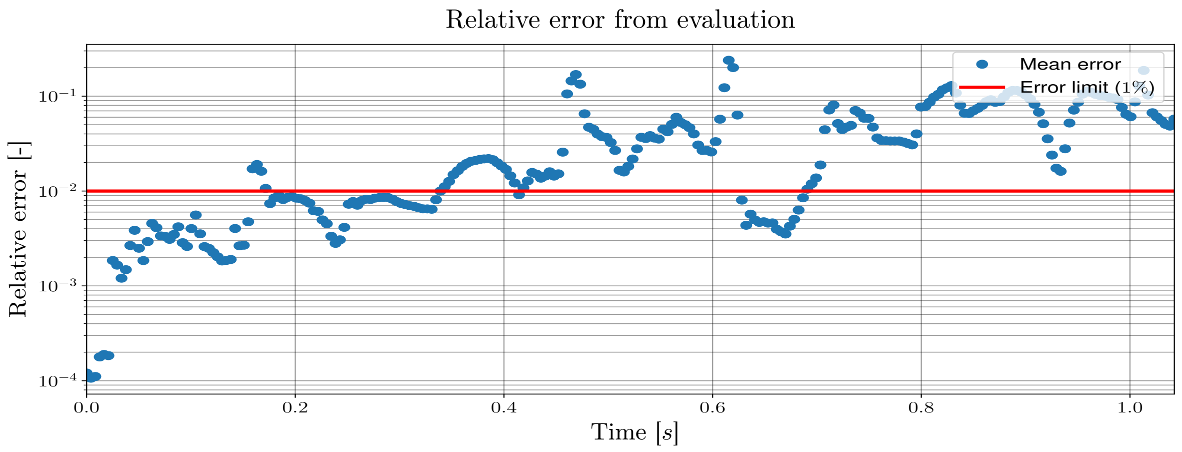


Figure A.6: Relative error of the evaluation of the PCE model. The relative error is from the estimation of the velocity in longitudinal direction, from the trajectory in Figure 6.1a.

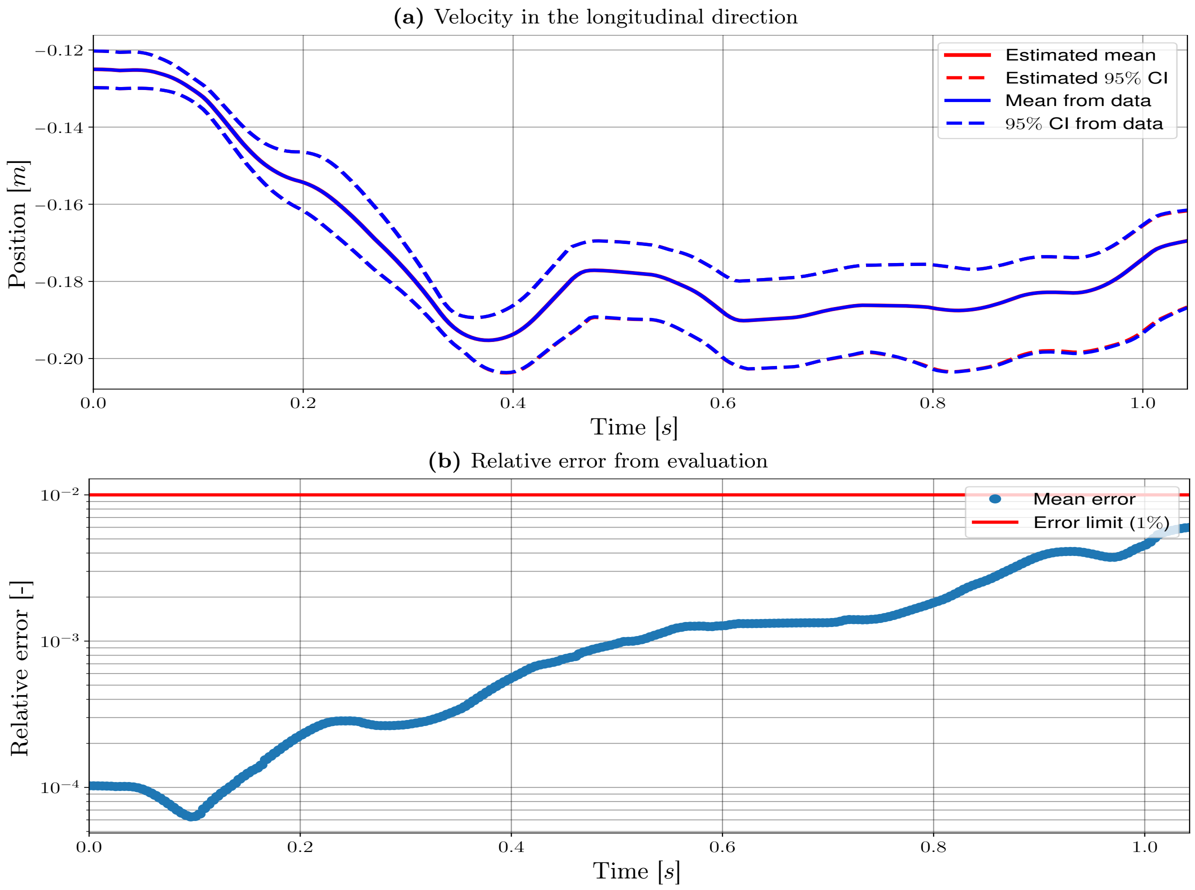


Figure A.7: Estimation of the position by the PCE model, with a polynomial order of three and seven divisions per uncertainty. The top plot shows the trajectory (estimated and sampled). The bottom plot shows the relative error per time-step.

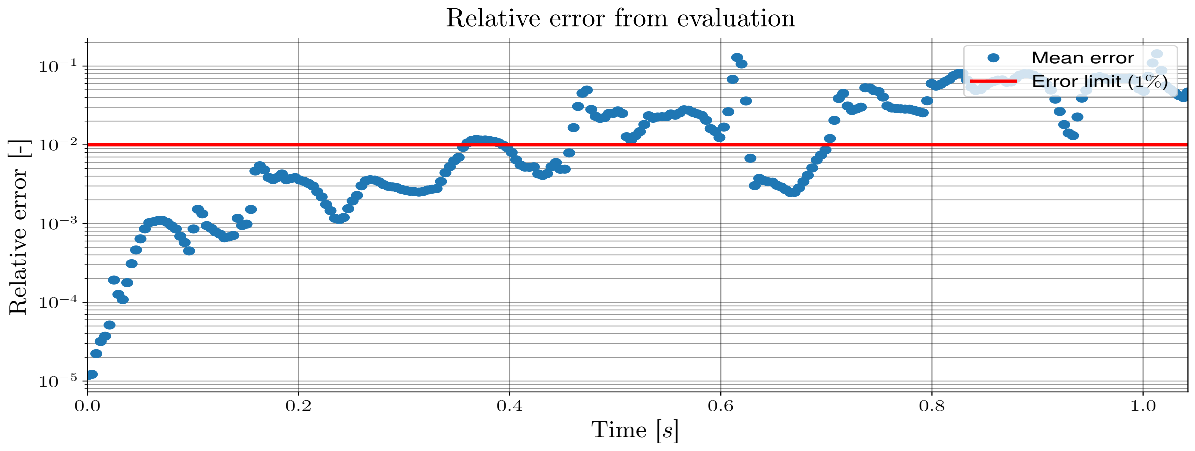


Figure A.8: Relative error of the evaluation of the PCE model, with a polynomial order of three and seven divisions per uncertainty. The estimation is from the velocity in longitudinal direction.

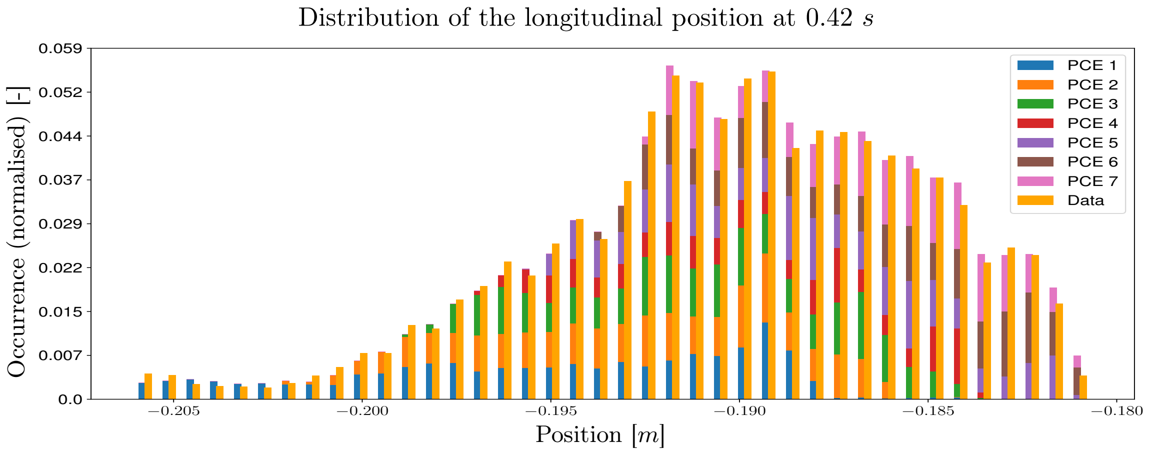


Figure A.9: Distribution of the position in the longitudinal direction, estimated with ME-PCE with a polynomial order of three and seven divisions per uncertainty. The estimate shows the contributions of the seven divisions for the RV length, labelled as PCE $1-7$. The data is the same as in Figure 6.4.

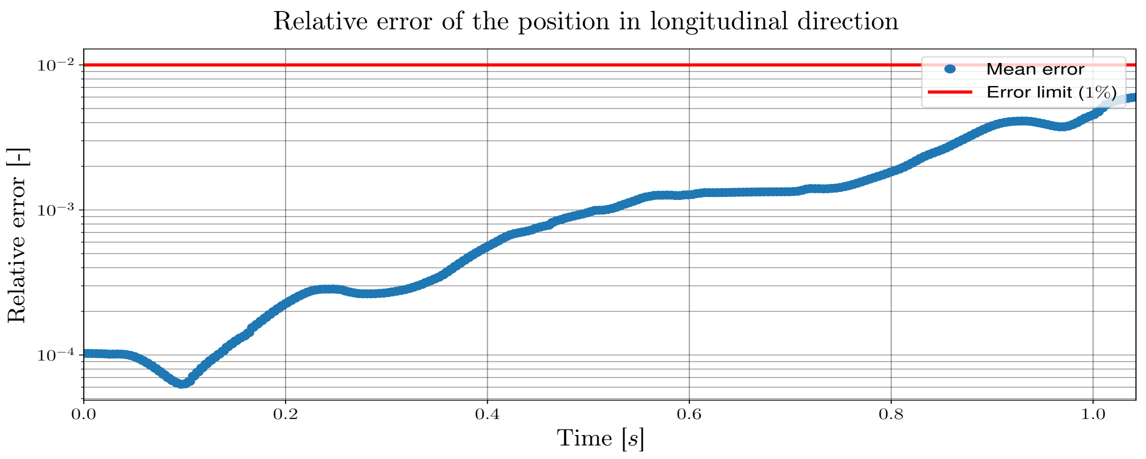


Figure A.10: Relative error of the position in longitudinal direction, estimated with a PCE model of polynomial order three and seven divisions per uncertainty. The error is calculated using the data for evaluation.

A.2 PCE Results Uncertain Actions

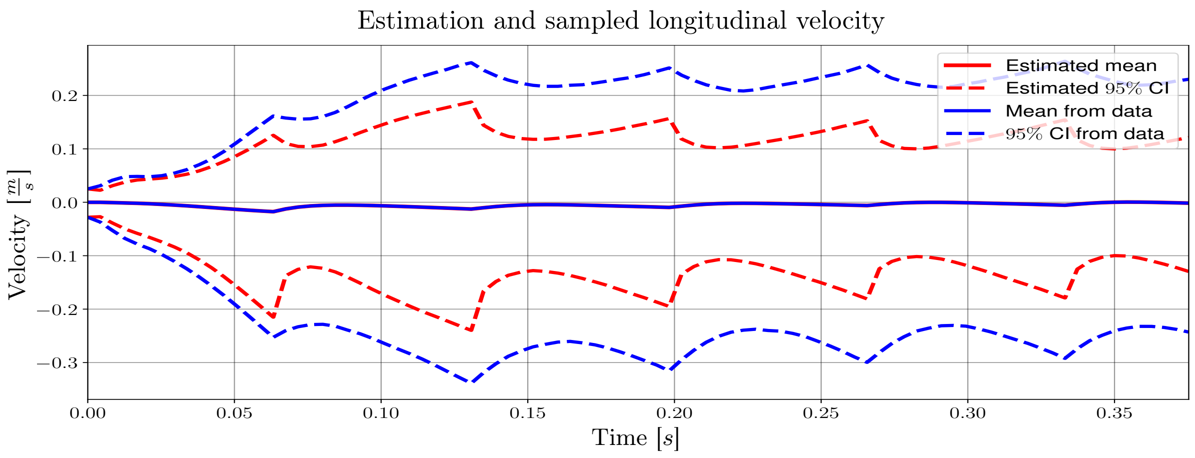


Figure A.11: Sampled and estimated velocity in the longitudinal direction. The ME-PCE model had an polynomial order of three and an action window of three actions.

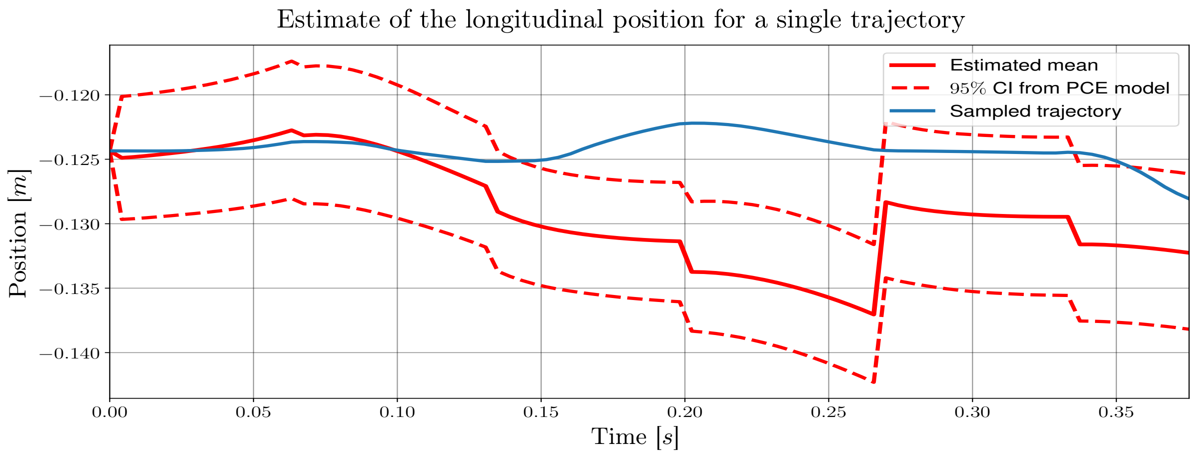


Figure A.12: Estimation of a single trajectory by the ME-PCE model of order three and seven divisions for the uncertainties density and length. The PCE estimate used the same actions as the sampled trajectory.

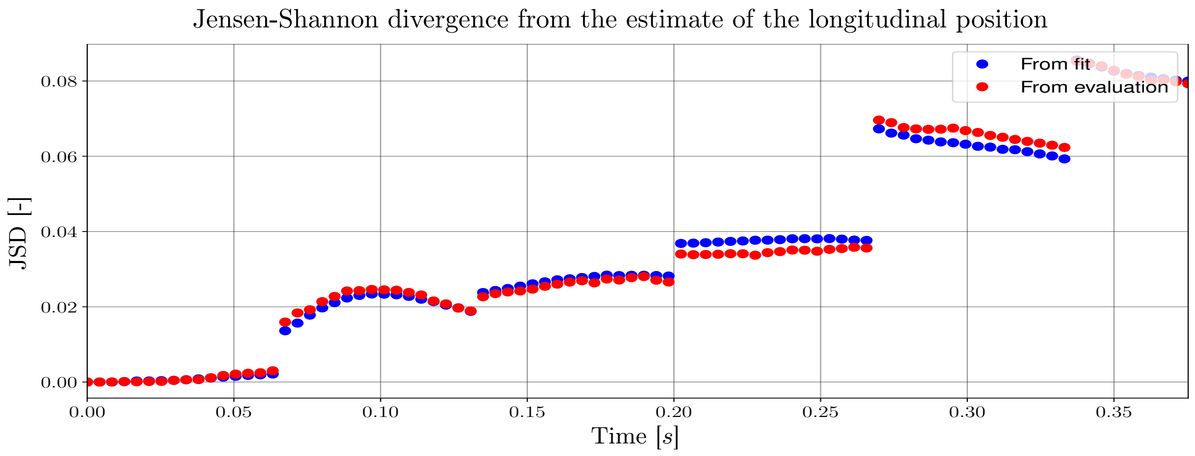
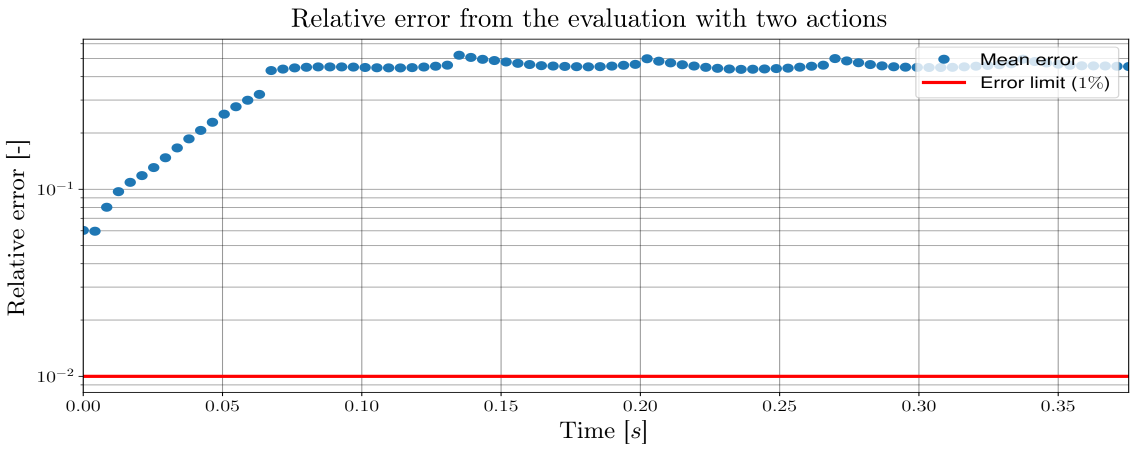
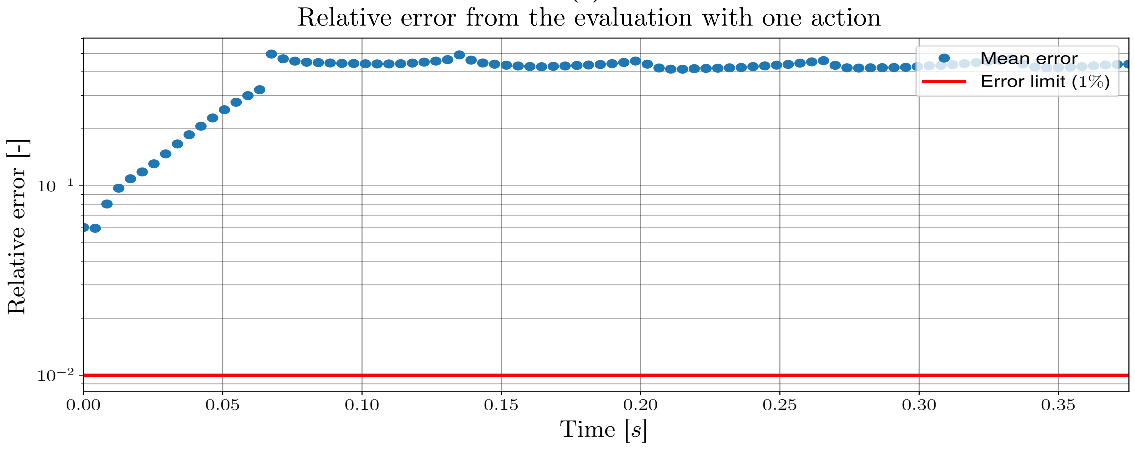


Figure A.13: JS divergence of the longitudinal direction with the ME-PCE model of polynomial order three, seven divisions per RV and an action window of three actions.

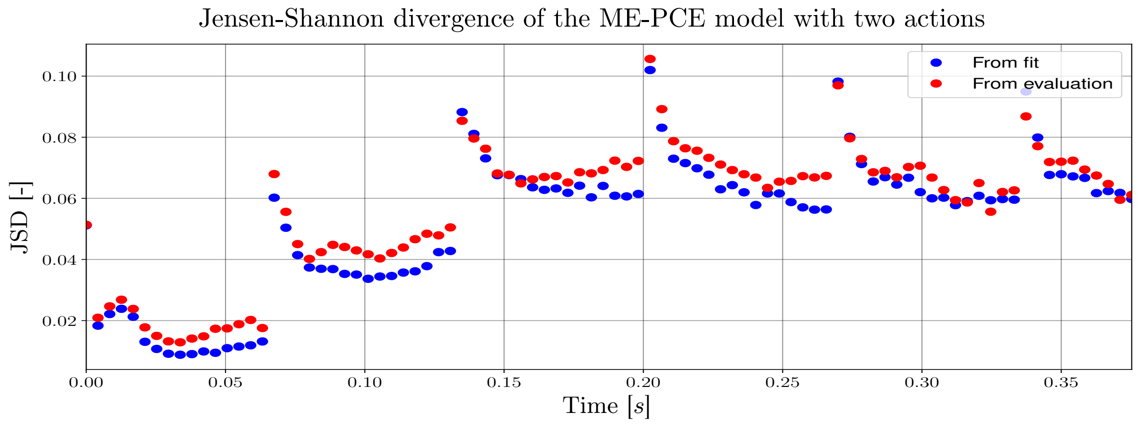


(a)

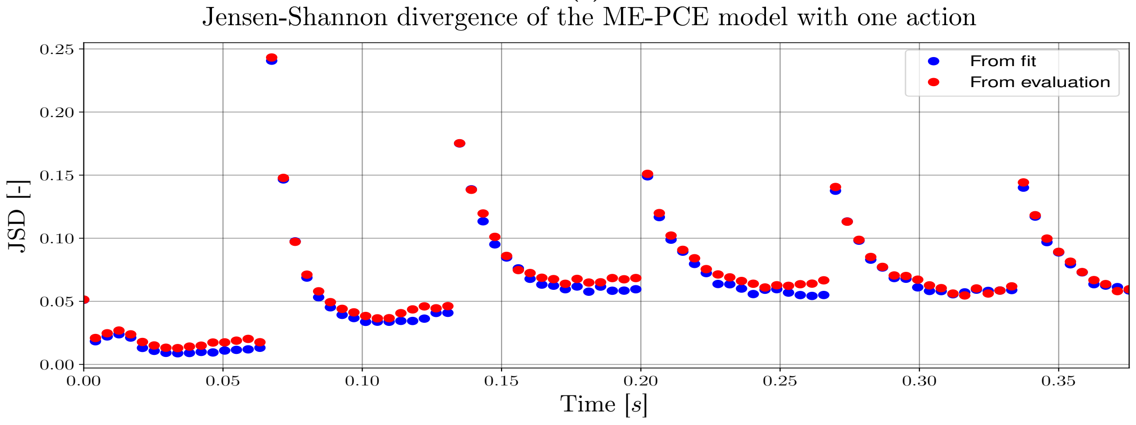


(b)

Figure A.14: Relative error of the estimation of the longitudinal velocity with an action window of two or one respectively.



(a)



(b)

Figure A.15: JS divergence from the estimate of the longitudinal velocity. The top plot shows the estimate from a ME-PCE model with an action window of two actions, the bottom plot shows the estimate from a ME-PCE model with one action in the action window.

A.3 Results MOE

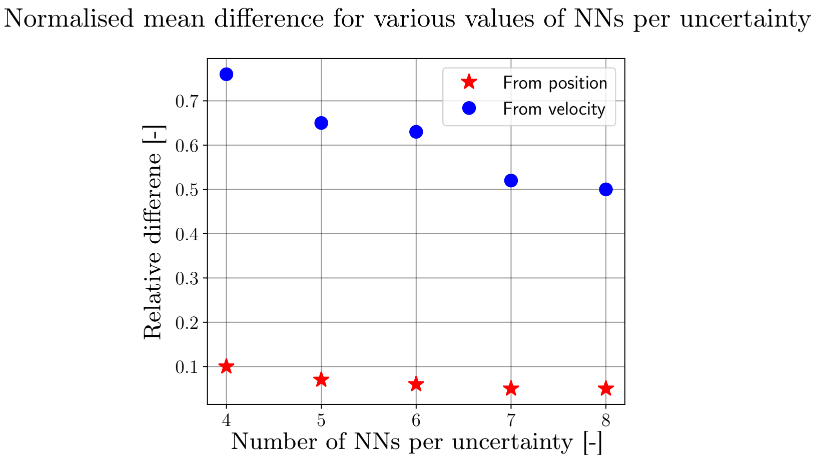


Figure A.16: Normalised mean difference between the estimated mean by the model ensemble and the sampled data for velocity and position in longitudinal direction, for multiple numbers of NNs model per uncertainty.

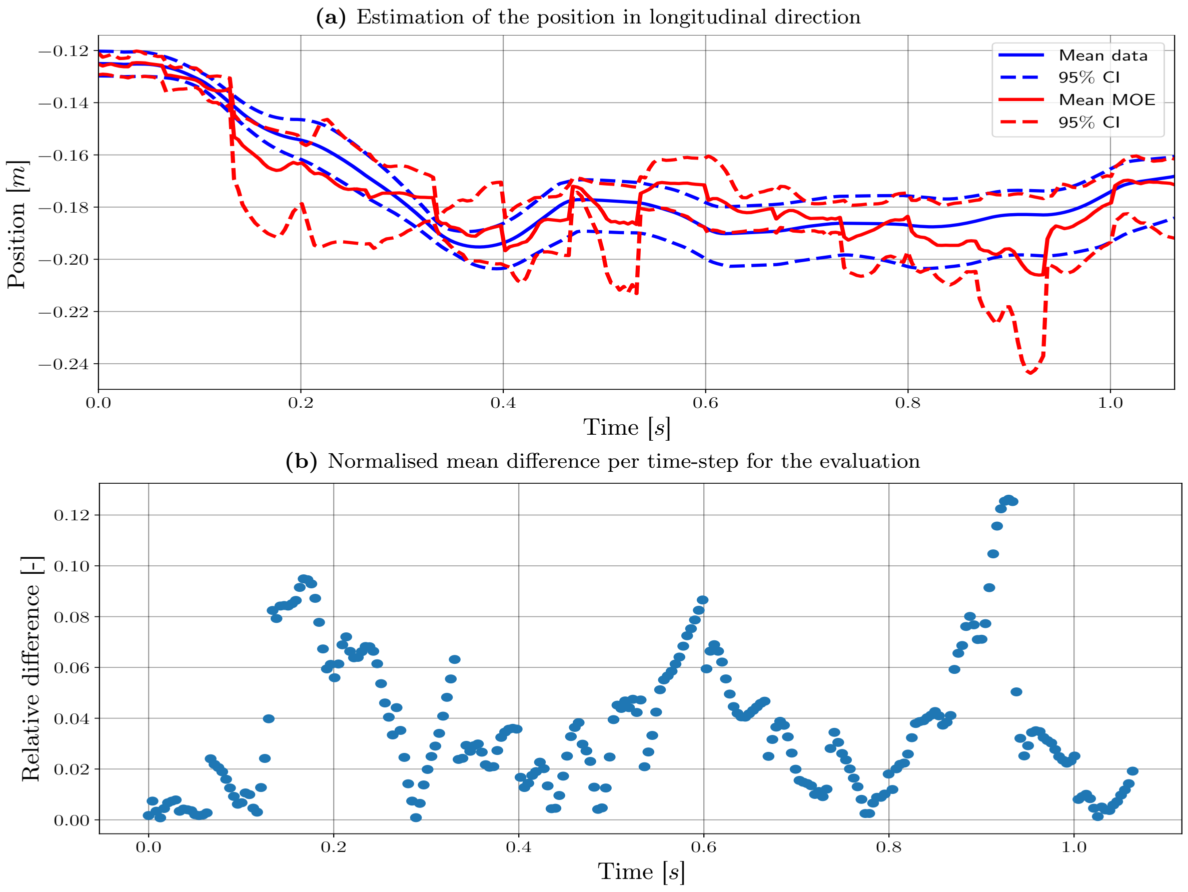
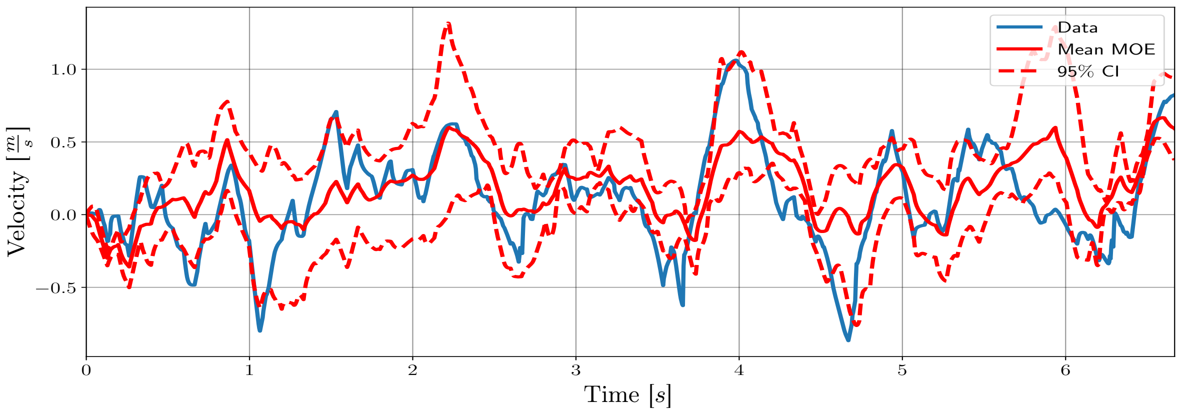


Figure A.17: Estimation of the position in the longitudinal direction by the model ensemble. The top plot is the estimated trajectory and the bottom plot the relative error between the estimated mean and trajectory. The figures were made with the data used for evaluation.

(a) Velocity in the longitudinal direction with varying actions



(b) Normalised mean difference per time-step for the evaluation

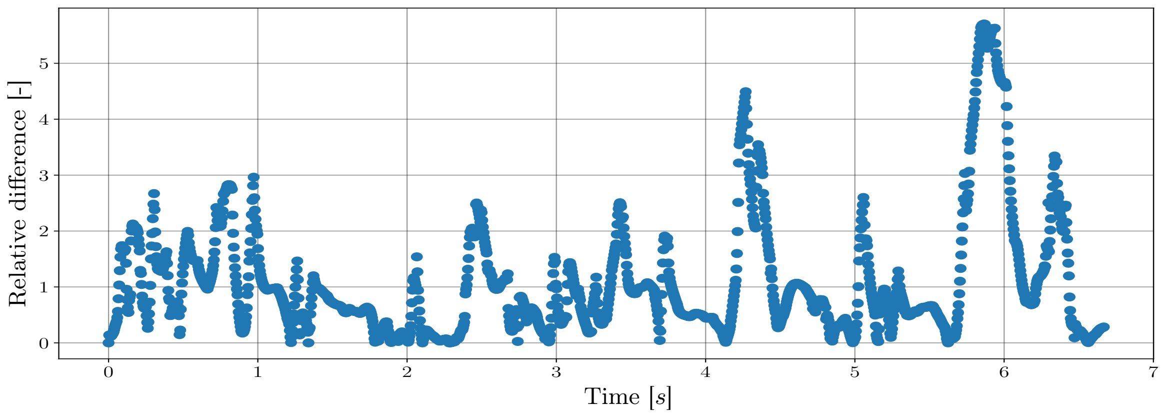
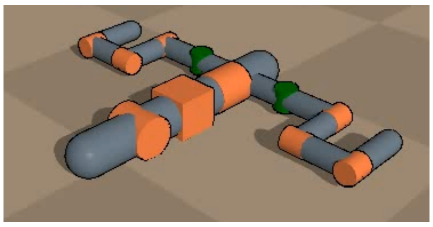
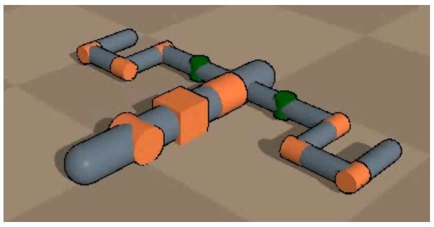


Figure A.18: Estimation of the velocity in the longitudinal direction with varying actions by the model ensemble. The top plot is the estimated trajectory and the bottom plot the relative difference between the estimated mean and trajectory. The figures were made with the data used for evaluation.

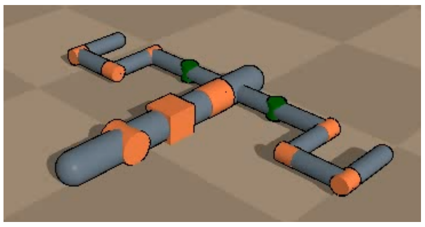
A.4 Results TRPO



(a) Link length: 0.15 m



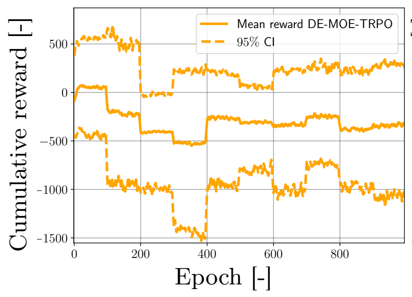
(b) Link length: 0.17 m



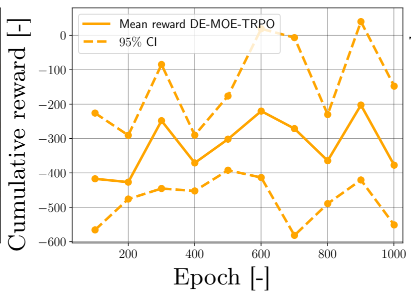
(c) Link length: 0.20 m

Figure A.19: The used robot with three different values for the length of the link.

(**a**) Cumulative reward during training in the DE-MOE



(**b**) Cumulative reward the policy, evaluated in the original environment



(**c**) Cumulative reward of DE-MOE-TRPO

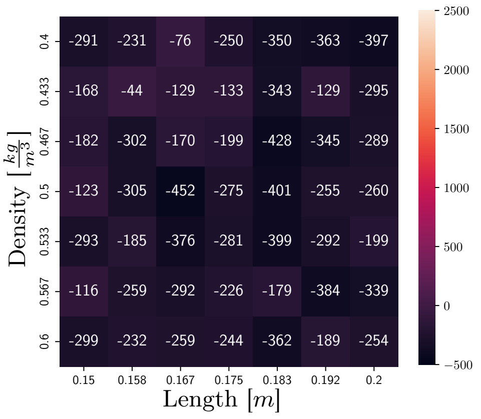


Figure A.20: Training and evaluation performance of ten policies trained with DE-MOE-TRPO. The left figure gives the cumulative reward during training. The middle figure is the evaluation of the policy in the original environment, evaluated every 100 epochs. The right figure gives the performance of the final policy for the distribution of density and length.

A.5 Artificial Intelligence used in this Report

For this report, QuillBot has been used for grammar checks and improve sentence structure. It is retrieved from `https://quillbot.com/grammar-check` on 30-05-2024.