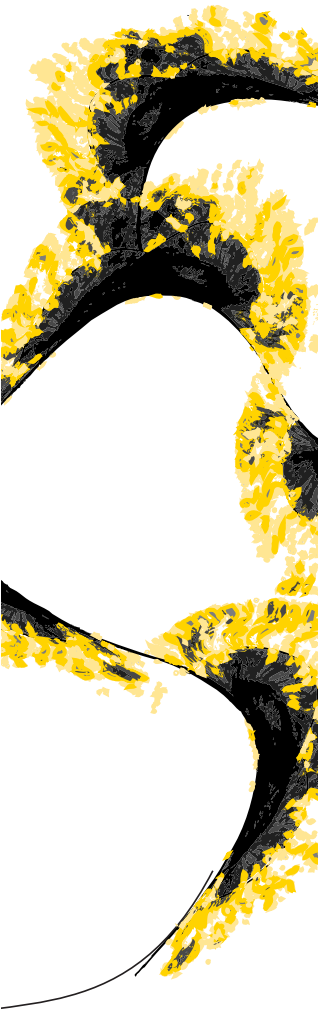


RAM ROBOTICS AND MECHATRONICS

# TOWARDS REDUCING THE SAMPLE COMPLEXITY OF A MODEL-FREE REINFORCEMENT LEARNING AGENT CONTROLLING A SINGLE SEGMENT TENDON-DRIVEN CONTINUUM MANIPULATOR

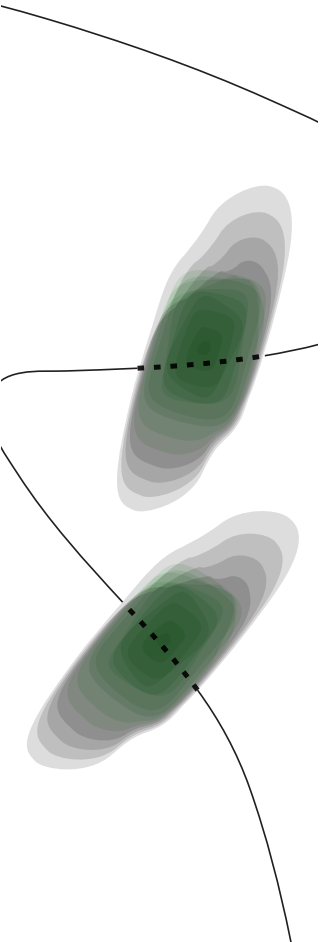
K.J.H. (Kasper) Hendriks

MSC ASSIGNMENT

**Committee:**
dr. ir. F. van der Heijden
Y.X. Mak, MSc
dr. ir. M. Abayazid
dr. E. Mocanu

September, 2020

047RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

# Summary

Soft robotic endoscopes have the potential to play an instrumental role in minimally invasive surgery, which reduces the patients' recovery time and discomfort compared to open surgery. However, control of these manipulators is challenging due to the computational cost of accurate models. Reinforcement learning can alleviate the need for such models by directly learning a policy instead. Unfortunately, model-free reinforcement learning techniques suffer from high sample complexity, limiting their practical use. This work aims to outline an end-to-end process to develop a practically viable reinforcement learning controller based on the soft actor-critic algorithm by reducing its sample complexity. A tendon-driven continuum manipulator is fabricated and then modelled using a non-linear autoregressive exogenous neural network. This model is used to generate a student policy that imitates expert behaviour as well as the policy of a model-free agent trained in simulation. The simulated agent's policy and the student policy are used to initialise a model-free learner, with the intent of reducing the sample complexity by allowing the agent to focus on fine-tuning an already competent policy. The effectiveness of these methods is evaluated by comparing the performance as a function of learning time with that of an agent that was trained without any prior knowledge. Results indicate that while the endoscope is able to learn a reaching task, the sparsity of information about the state-space in the student policy and the model inaccuracies used to develop the simulated agent lead to performances that were similar or worse for a given number of training steps.

## Acknowledgements

I would like to thank my daily supervisor Yoeko Mak for all the time and effort he put into helping me throughout my thesis. Our weekly meetings were a nice opportunity for me to discuss the progress I made (and problems I encountered) in the week prior and figure out a path forward. These meetings were especially nice after having taken a digital form due to the global events, as talking about my thesis work with other people is a great way to stay motivated.

I would also like to thank Momen Abayazid for giving me the opportunity to do this project, providing feedback on my work and for organising biweekly meetings with the medical robotics subject group. These meetings provided an excellent opportunity to hear about the work of others in a similar field and allowed for light-hearted discussions about the progress my peers and myself made.

Finally, I would like to thank Nehal Mathur for her continuous moral support, enabling me to stay motivated throughout my work.

Kasper
Enschede, 16-09-2020

# Contents

# 1 Introduction

Many fields of research draw inspiration from nature. From something as simple as plants sticking to clothing inspiring the invention of Velcro (Stephens, 2007), to improving the aerodynamics of a bullet train based on the kingfisher bird (Kobayashi, 2005). Inspired by nature, a new field of robotics has emerged in the past decades. This branch is called soft-robotics and is characterised by the use of highly compliant materials. The use of these materials has allowed a number of nature inspired soft robots to be created, such as a robotic elephant trunk (Rolf and Steil, 2014) or a soft robotic earthworm (Chatterjee et al., 2018). The inherit compliance of soft manipulators makes them suitable to interact with unknown environments in which their rigid counterpart could potentially cause damage, making them useful in medical applications (Bhagat et al., 2019).

One of the medical applications where the compliance of a soft manipulator is desirable is endoscopy. Endoscopy is an often used form of minimally invasive surgery (MIS) or natural orifice transluminal endoscopic surgery (NOTES). During endoscopic interventions, the endoscope, of which an example can be seen in figure 1.1, is inserted inside of a human. Since this environment is not (fully) known and is dynamic in nature, the compliance of the soft manipulator makes the interaction between the robot and the human safer than with the use of traditional rigid manipulators. Another benefit of soft manipulators is that they can navigate around organs. This flexibility allows the target location to be reached through a small incision in the body, whereas open surgery could be required when using rigid manipulators. This is one of the reasons that lead to MIS and NOTES gradually replacing the need for open surgery, reducing the adverse effects such as patient trauma and decreases the patient's recovery time (Gifari et al., 2019).

**Figure 1.1:** Prototype soft manipulator inspired by the octopus arm used for MIS on a beating heart (Wang et al., 2017).

## 1.1 Problem context

While having an infinite amount of degrees of freedom makes soft continuum manipulators useful in a number of medical tasks, it gives rise to problems in terms of controllability due to

the complexity involved in modelling the system. Different techniques can be used to create a model of soft robotic links, with the most popular ones being the piece-wise constant curvature approximation and Cosserat beam theory (Renda et al., 2017).

The piece-wise constant curvature model provides a way of modelling the statics of a continuum manipulator by approximating it as (a series of) arcs with constant curvature. However, this method is invalid under high external loading that leads to non-circular bending and is unsuited for modelling the dynamic behaviour of the manipulator (Webster and Jones, 2010). Being able to deal with external loads is important for the controller design as the environment in which the manipulator will operate can exert varying loads on the manipulator. Models based on the continuous Cosserat approach do remain valid under external load, but are computationally demanding. This makes them unsuitable for real-time closed-loop control (Renda et al., 2017), (Ho et al., 2018). Instead of solving an online computationally expensive optimisation problem, closed-loop control policies can be learned directly using reinforcement learning.

Reinforcement learning can be split into two different branches: model-based and model-free methods. Model-based reinforcement learning methods utilise or explicitly create a model of the environment, which is then used to plan out which action to take. This is similar to how a human would for example play the game of tic-tac-toe. Before making a move, a player often envisions what the opposing player will do as a result of the move and evaluates the success of the move based on this. Here, the player implicitly creates a model of the environment to predict the outcome of a certain action. In model-free techniques, the effectiveness of a move is evaluated based on experiences gained when previously making the same move (Gläscher et al., 2010). If a certain move has resulted to losing the game in previously played games, the player is less likely to perform that move the next time the same situation presents itself.

These two different ways of learning a policy have an impact on both the time it takes to learn this policy as well as the quality of the policy. In general, the model-based learner has a lower sample complexity, it requires fewer training steps, whereas the model-free learner has a higher asymptotic performance (Gu et al., 2016). The main reason behind this is that the model-based learner is limited by the quality of the model. The model-free learner is not bounded to a model, but exploring the full space of possibilities is a time consuming process. While for some problems computational time might not be a limiting factor, it is often impractical or expensive to train a model-free agent in a real-life setting. On the contrary, creating an accurate system model can be challenging for dynamically complex problems, limiting the accuracy of the final policy.

The complexity of creating accurate and computationally inexpensive models for soft manipulators combined with the impracticality of model-free methods makes it such that real-time dynamic control of this class of manipulators is still largely an open problem. The following section aims to present the current state-of-the-art methods and how their shortcomings lead to the work done in this thesis.

## 1.2 Related work

A number of different control methods have been developed for soft manipulators under quasi-static conditions. Bieze et al. (2018) present a closed-loop controller based on a finite element method model of their system. This controller was tested on a two section pneumatically driven manipulator. Qi et al. (2016) shows a fuzzy based control method for a tendon-based manipulator. The model used for the controller is based on the piece-wise constant curvature model, making it suitable for only kinematic situations. Additionally, the controller is computationally complex, which would make it unsuited for real-time control. While control based on the kinematics of the soft manipulator can seem to provide good control of the robot, this

form of control is not sufficient when interacting with an environment that can cause dynamical situations to occur.

Instead of using a kinematic model, Falkenhahn et al. (2017) present a way to create a controller based on the dynamical model of a soft bionic handling assistant. This is a three section soft manipulator where each section is actuated using three air chambers. The dynamical model of the assistant is created based on the Euler-Lagrangian formalism for a concentrated mass. Based on this model, a cascaded control strategy is used to control the manipulator. In this setup, the control for the air chambers is decoupled from that of the manipulator dynamics, as the dynamics for the bellows is much faster than the dynamics of the manipulator. This model-based controller was shown to achieve good results and was able to damp out an artificially generated swing motion in real-time. However, during the experiments hardware limitations were often reached which made it such that the inner loop pressure could not reach the desired pressures, which lead to a larger end-effector error than predicted.

Even when a sufficiently accurate dynamical model is created, a high amount of sensory data is required to be able to completely define the state of the robot. This is because the state is not fully defined by actuator positions, as the robot's shape is influenced as a result of interactions with the environment. For this reason, different model-free control methods are developed.
The work done by Yip and Camarillo (2014) shows the implementation of a model-free control method on a tendon-driven soft manipulator. The used method bases its ideas on control of rigid robots, where the robot Jacobian is used to define actuator velocities based on desired end-effector velocity. However, in the context of soft manipulators in a constrained environment, the Jacobian cannot fully be measured as a result of the infinite amount of degrees of freedom. Instead, an initial Jacobian is estimated before starting the robot. During operation, the actuator inputs are used to solve a minimisation problem to update the Jacobian based on measurement data. This method was shown to be able to accurately track a reference in free space, but showed a larger error when it was used in constrained environments. Additionally, the controller was found to only be usable in static environments, which makes it unsuitable for some medical interventions that pose a highly dynamic environment.

### Reinforcement learning

Both the aforementioned model-based and model-free methods require solving online optimisation problems. While for small and relatively simple systems this can sometimes be done in real-time, it is difficult to create real-time closed-loop control methods for more complex systems. Instead, Thuruthel et al. (2019) propose to directly learn the closed-loop control policy using reinforcement learning. A two section pneumatic manipulator in which only the first section is actuated is used for the experiments. The forward dynamics of this manipulator are learned by giving a set of random motor inputs and measuring the resulting state. This data is then used to train non-linear autoregressive network. Using these learned dynamics, optimisation techniques can be used to find the control inputs required to drive the robot to a certain state. Despite the computationally efficient model, this optimisation cannot be used in closed-loop form. To obtained the closed-loop policy, a guided policy search combined with trajectory optimisation is used in an iterative fashion, where the set of allowable trajectories changes depending on the currently learned policy. Combing these methods allows a sufficiently well performing closed-loop control policy to be developed in around two hours.

Another method for model-based reinforcement learning is to learn the inverse kinematics of a specific task using programming by demonstration, as presented by Chen et al. (2016). Here, a practitioner operates a single segment tendon-driven soft manipulator to perform a certain task. From this data the inverse kinematics are estimated using Gaussian Mixture Models. To make the model more robust, real-time adaptations to the learned dynamics are performed

using a reinforcement learning algorithm. This procedure is used in simulation to perform two tasks. The results of both these tasks were promising, but they have not been implemented on an experimental setup.

### Reducing sample complexity

To achieve a better performance, model-free reinforcement learning methods can be investigated. However, model-free reinforcement learning methods suffer from a high sample complexity, making them impractical to use. Therefore, different techniques are developed to try and reduce the training time.

Satheeshbabu et al. (2019) present a model-free reinforcement learning technique that is used to control a single section pneumatic soft continuum robot. Deep-Q learning together with experience replay is used to reduce the training time associated with this high dimensional problem and ensure that the policy does not control the system to a local minimum. Instead of training the policy on the experimental setup, it is trained on a simulation based on the Cosserat rod model. The resulting policy was tested in simulation as well as on the experimental setup. The experiments on the setup were done with the policy in open-loop mode. These experiments showed that the system was able to perform a reaching task even with an unmodelled load on the end-effector. However, the error achieved on the experimental setup was larger than that in simulation and improvements can be made by executing the policy in closed-loop form.

Instead of learning a policy in simulation and then executing it on a real robot, another way to reduce the learning time for a certain task is to have multiple agents learn at the same time. Gu et al. (2016) describe a method to learn a certain task quicker in the case time is the limiting factor. This method uses a number of agents that work together to learn a certain task. One of these agents is the central learner that performs the policy updates. The other workers collect data while adhering to the central policy and send this data back to the central learner. The decentralisation of the policy updating makes it such that the worker robots can operate in real-time, as no time has to be spent learning based on the gained experiences. This idea was implemented on rigid serial manipulators, which learned how to open a door.

Apart from using simulation or multiple agents, research is done to find more sample efficient model-free reinforcement learning algorithms. Haarnoja et al. (2018b) present a novel algorithm called soft actor-critic that is designed to fulfil this goal. Unlike most reinforcement learning algorithms, this algorithm does not only try to maximise the cumulative reward, but also the entropy. The combination of trying to maximise reward while being as random as possible has the advantage that it brings structure to the exploration phase. The agent is incentivised to widely explore the state-space, but will give up on regions that result in a poor reward. This algorithm is run on various simulated benchmark environments and is shown to outperform existing state-of-the-art algorithms in both performance as well as sample complexity.

Haarnoja et al. (2018c) uses a slightly modified version of the soft actor-critic algorithm to control the locomotion of the Minitaur, a four legged robot. After approximately two hours of training the agent has learned a policy with which it is able to walk. This policy is shown to generalise well, allowing the robot to walk in unseen environments. The algorithm is also deployed to manipulate a robotic hand with nine degrees of freedom that needs to close a valve. Using purely images as feedback the system is able to learn the task in roughly 20 hours. When feeding in the valve positions directly this time was reduced to three hours, compared to the seven and a half hours another state-of-the-art method needed.

Nagabandi et al. (2017) present a different approach to reducing the sample complexity. In this approach, a model-free reinforcement learning agent is initialised with a model-based pol-

icy. The model-based reinforcement learning method is used to learn a number of simulated benchmark tasks to a sufficient degree. In some cases, the model-based method requires only 14% of the samples the model-free agent requires to reach the same performance. Initialising the model-free agent with an already competent model makes it such that the agent can work on fine-tuning its performance. This is advantageous when it comes to sample complexity. Additionally, starting with a competent model can also reduce damage the experimental setup could cause, e.g. a robot learning how to walk would fall over far less frequently.

## 1.3 Opportunity for research

Different methods have been employed to control a soft continuum manipulator. One class of these methods is the static control, based on the kinematics of the manipulator. While this can achieve satisfactory results in the case where the robot moves quasi-statically, interactions with the environment and the inherit compliance of the robot are likely to cause dynamical situations to occur, decreasing the performance achieved by the controller. This makes it such that these controllers are not suitable to be deploying in a practical setting.

Another class of methods is the dynamical control of the manipulator. This can either be done model-free or model-based. The model-based approaches face a number of challenges. Creating a model of a highly compliant manipulator is complex due to a number of non-linear effects. Even when a sufficiently accurate model is created, high dimensional sensor data is required to accurately define the state of the robot. Additionally, the state-of-the art models are too computationally expensive to be used for real-time control. Model-free techniques have also been used to control the manipulator, but these have only been shown to work in a static environment, making them unsuited to be used in a practical environment.

Model-based reinforcement learning techniques have been employed to control a soft manipulator. The computational complexity of using a model-based technique is circumvented by the use of reinforcement learning techniques as these can be used to learn a control policy directly, rather than having to solve an optimisation at every time step. A model-based reinforcement learning method has been shown to be able to control a two section pneumatically actuated soft manipulator using only two hours of learning time.

Control based on model-free reinforcement learning is mainly focused on decreasing the training time as this is a known limitation for practical implementation. However, in different scenarios it is shown that the model-free learner does achieve a higher performance than the model-based learner, which is why there is still interest in this technique. While a number of different techniques are shown to be effective in decreasing the sample complexity of model-free learning techniques, little research has been done to combine these techniques with the control a soft continuum manipulator.

This leads to the opportunity for research in this field. While model-free fine-tuning as well as the soft actor-critic separately have been proven to be effective at reducing sample complexity, the combination of the two has not been implemented on a tendon-driven endoscope. Therefore, it is interesting to see whether these techniques can be used to achieve a practically viable controller for a tendon-driven continuum manipulator.

## 1.4 Research question

One of the benefits of using model-free reinforcement learning techniques is that the complex modelling of soft robotic manipulators can be skipped. While not needing a model of the system might sound appealing, the model can be used to decrease the sample complexity of the model-free agent to make it practically viable. The aim of this research is to work towards a start-to-end process where both the time spent modelling the system as well as the time

spent executing experiments on the system is minimised. While the framework provided for this process is generally applicable to a wider variety of problems, it is applied to a single segment tendon-driven continuum manipulator. Work towards this goal is done by answering the research question:

*How can model-based techniques be combined with model-free reinforcement learning to reduce the sample complexity of learning to control a single segment tendon-driven continuum manipulator, while simultaneously keeping the time spent modelling the system at a minimum?*

The following sub-questions are used to answer this research question:

*How can a single segment tendon-driven continuum manipulator be modelled using a non-linear autoregressive exogenous model for the purpose of initialising a model-free agent?*

*How can the model-free reinforcement learning algorithm soft actor-critic be used to control a single segment tendon-driven continuum manipulator to move between points in its workspace?*

*To what degree is the sample complexity of a model-free reinforcement learning agent reduced when initialised based on the obtained system model?*

## 1.5 Approach and thesis structure

The rest of this thesis is structured to answer the aforementioned research questions.
*Chapter 2 - Background* contains the theoretical background of the rest of this work. This starts with the theory on multilayer perceptron networks. This is then used to describe the non-linear autoregressive exogenous (NARX) network, which uses a similar structure. Reinforcement learning is discussed next, with the emphasis on deep reinforcement learning and the soft actor-critic algorithm. Finally, the theory supporting the behaviour cloning algorithm is explained.
*Chapter 3 - Methods* contains the methods used throughout this work. This chapter starts with an explanation of the experimental setup and a (re)design of the endoscope. Afterwards, the methodology used to create a system model is discussed. The methods used to create expert demonstrations using this model, as well as the method used to train a policy using these demonstrations is outlined next. The chapter concludes with the method used to train and evaluate a model-free agent, both in simulation as well as on the experimental setup.
*Chapter 4 - Results* presents the results following the same structure of the previous chapter. This starts with demonstrating the hysteresis effects resulting from redesigning the endoscope, then shows the competence of the created model and finally demonstrates the performance of the RL agent when initialised randomly or using the model-based methods.
*Chapter 5 - Discussion* gives an interpretation and discussion regarding the achieved results. After this, a few points of improvements with respect to the acquisition of the results are discussed.
*Chapter 6 - Conclusion* presents and answer to the research question and concludes this thesis with recommendations for future work.

# 2 Background

This chapter provides the background information necessary to understand the used methods. This starts with a section on multilayer perceptron networks, as these are used in almost every other part of the thesis. The second section builds on the theory of multilayer perceptron networks and shows how these can be used to form a non-linear autoregressive exogenous (NARX) network. The third section gives a (brief) overview of reinforcement learning, focussing on the soft actor-critic algorithm. The last section shows the concepts behind behaviour cloning, which forms the bridge between the created system model and model-free reinforcement learning in the context of this research.

## 2.1 Multilayer perceptron

Neural networks are used throughout this work. This section provides the theory regarding neural networks relevant to understanding the rest of this thesis. The content of this section is based on the book *Pattern recognition and machine learning* by Bishop (2006).

A multilayer perceptron (MLP) provides a (non-linear) mapping between a vector of inputs and outputs. The general architecture for such a network is shown in figure 2.1. This figure shows three different types of layers: the input layer, the output layer and a variable number of hidden layers. These layers consist of a number of nodes and are connected using weights, which are represented as circles and links respectively. The two additional blue nodes connected to the hidden layer and output layer are the biases for these layers that act as an offset, conceptually similar to how $b$ acts as an offset in $y = ax + b$.

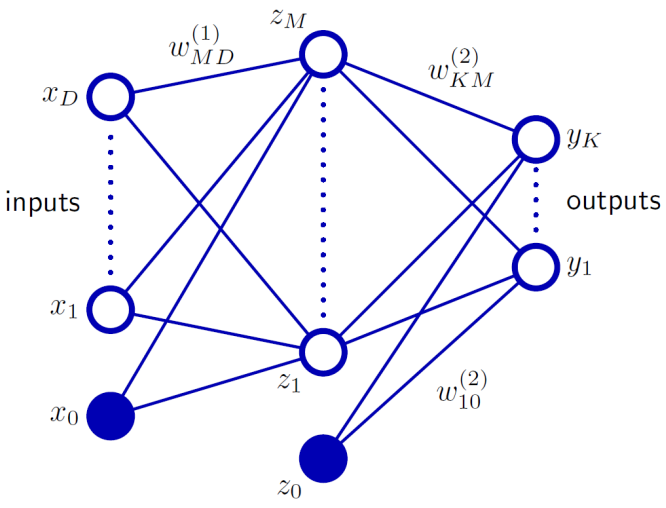



**Figure 2.1:** Network architecture of a fully connected multilayer perceptron network (Bishop, 2006).

Finding how the input vector is mapped to the output is done by calculating the values of each of the nodes in the network. The value of the $j$th node in the $N$th layer that is connected to $D$ nodes in previous layer can be calculated as:

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(N)} x_i + w_{j0}^{(N)}\right) \tag{2.1}$$

In this equation, $h(\cdot)$ is the activation function associated with neuron $z_j$, $w_{ji}^{(N)}$ are the weights associated to the $i$ nodes in the previous layer and $w_{j0}^{(N)}$ is the connected bias node. This equation holds for every node in the network and can be chained together to find the value of any node in the network.

This equation can also be written in matrix-vector form to obtain the vector $Z \in \mathbb{R}^{(M\times 1)}$ that contains the activations for the neurons in a layer. The activation function can be applied to this vector in element wise fashion to determine the value of each neuron. Finding the values of a vector of nodes $Z$ that is connected with a vector of nodes $X$ using weights $W$ can be done using:

$$Z = WX \tag{2.2}$$

$$\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} w_{10} & \cdots & w_{1D} \\ \vdots & \ddots & \vdots \\ w_{M0} & \cdots & w_{MD} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_D \end{bmatrix} \tag{2.3}$$

Where $Z \in \mathbb{R}^{(M\times 1)}$, $W \in \mathbb{R}^{M\times(D+1)}$ and $X \in \mathbb{R}^{(D+1)\times 1}$.

### 2.1.1 Training a network

While it is shown how a network forms a mapping between the input and output, this mapping is unlikely to yield any meaningful results given a random set of weights. To use a neural network in practice, the network has to be trained. Given a set of inputs and corresponding targets, the network is trained to minimise some error function that is dependent on the weights of the network, $E(\mathbf{w})$. Minimising this function is done iteratively, where each step is generally divided in two parts.

The first parts is evaluating the gradient of the error with respect to the weights, $\nabla E(\mathbf{w})$. This is most often done using some form of backpropagation. The backpropagation algorithm consists of two passes through the network. During the forward pass the activations of all hidden and output neurons is determined. Errors based on these activations are then propagated back through the network to evaluate the gradient of the error function.

The second step is to use this gradient to update the network weights. One method to train the weights of a neural network is stochastic gradient descent. In stochastic gradient descent, the network weights are updated as:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E_n\left(\mathbf{w}^{(t)}\right) \tag{2.4}$$

where $\eta$ is the learning rate and $E_n\left(\mathbf{w}^{(t)}\right)$ the error function evaluated for data point $n$. This method updates the weights by 'stepping' in the direction where the gradient of the error function decreases steepest, driving the error function to a (local) minimum. While many different training algorithms have been developed, these algorithms often utilise these two core concepts.

## 2.2 NARX

System identification can be done using a NARX model. In this section, the theory behind the NARX network is explained. The contents of this section are based on the work of Boussaada et al. (2018).

A NARX network is a type of recurrent neural network that can be used for the prediction of a non-linear time series, making it a useful tool for predicting non-linear system dynamics. The network has two distinct modes of operation: series-parallel and parallel. A simplified architecture of both networks is shown in the figure 2.2.

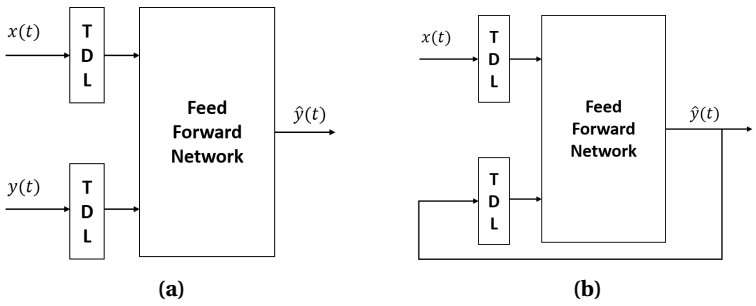



**Figure 2.2:** Different network architectures of a NARX network (Boussaada et al., 2018). (a) Series-parallel architecture. (b) Parallel architecture.

In this figure, the input to the network is denoted as $x(t)$, the output as $y(t)$ and the estimated output as $\hat{y}(t)$. The TDL blocks signify tapped delay lines. Assuming these delay lines have an input $v_t$, the output is $\{v_{t-i},\ldots,v_{t-j}\}$, where $i$ is the minimum delay and $j$ the maximum delay. The limits of these tapped delayed lines are problem specific hyperparameters. The difference between the two architectures is the role of the predicted output. In the series-parallel configuration, the output is predicted based on a set of known input data and previous states as:

$$\begin{aligned}\hat{y}(t+1) = F\big(&x(t-d_{x,i}), x(t-d_{x,i}-1),\ldots,x(t-d_{x,j}),\\ &y(t-d_{y,i}), y(t-d_{y,i}-1),\ldots,y(t-d_{y,j})\big)\end{aligned} \tag{2.5}$$

Where $d_{x,i}$ and $d_{x,j}$ are the minimum and maximum input delay and $d_{y,i}$ and $d_{y,j}$ the minimum and maximum output delay. This architecture is used to reduce the single-step prediction error, as it uses both the true input and output data.

The parallel architecture uses its own predicted state in the prediction of the future states as:

$$\begin{aligned}\hat{y}(t+1) = F\big(&x(t-d_{x,i}), x(t-d_{x,i}-1),\ldots,x(t-d_{x,j}),\\ &\hat{y}(t-d_{y,i}), \hat{y}(t-d_{y,i}-1),\ldots,\hat{y}(t-d_{y,j})\big)\end{aligned} \tag{2.6}$$

This makes the parallel architecture suitable for long-term time series prediction as the network can, given a list of inputs, predict not only the output at the next step, but also at all future steps for which input is available.

Training the network is done in two stages, each stage utilising the strengths of one of the outlined network architectures. First, the series-parallel network is trained utilising both input $u$ and state $y$ data. After achieving a satisfactory performance, the network is closed, converted to the parallel architecture, and trained further. This time, only the input data is required, as the estimated states $\hat{y}$ are fed back into the network for estimations of future states. When deployed, only the parallel archetype is used given the unavailability of the states $y$.

## 2.3 Reinforcement learning

This section provides the theoretical background behind reinforcement learning. After an introduction to the important concepts, the focus is shifted to deep reinforcement learning techniques, with the emphasis on the soft actor-critic algorithm. This section is based on the book *Reinforcement learning: An Introduction* by Sutton and Barto (2018) and *OpenAI Spinning up* by OpenAI (2018).

Reinforcement learning is a branch of machine learning where an agent learns how to act through interaction with the environment. The agent interacts with the environment by taking actions, which transition the agent to certain states and are associated with a certain reward. This reward gives the quality of a state-action pair and is used as a basis for the agent to formulate correct behaviour. The overall goal for the agent is to maximise the total reward it accumulates over an episode. The agent maximises this goal by developing a policy, which is a mapping between the environment state and a corresponding action.

### 2.3.1 Environment interaction

The image in figure 2.3 shows the basic interaction between an agent and its environment. The agent is able to take actions that are part of its action space, which can be either discrete or continuous. An example of a discrete action space is an agent that needs to swing up a pendulum but can only do so by applying maximum torque in either direction or no torque at all. The continuous variant of this action space would allow the agent to also apply any torque in between the two extrema. After applying action $a_t$, the state of the environment changes. This change is fed back to the agent by means of a state $s_t$ and a reward $r_t$. The state provides a description of the agent in its environment. To prove convergence, this state needs to obey the Markov property. This means that the state transition probability is completely determined by the current state and does not require information about previous states. The reward $r_t$ is given to the agent as a result of its previous action. In practice, this reward is prescribed by a (hand-engineered) reward function $R(s, a, s')$ which maps the current state, the current action and state resulting from this action to a scalar value.

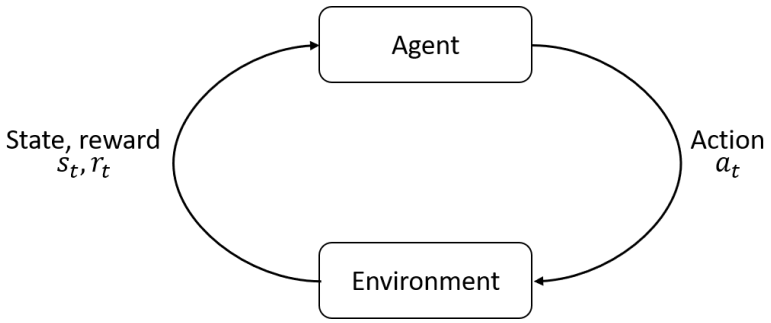



**Figure 2.3:** Agent-environment interaction (OpenAI, 2018).

### 2.3.2 Policy

In order to fulfil the goal of maximising cumulative rewards, the agent needs to learn how to act in every possible situation. In other words, the agent needs to develop a policy $\pi$. This policy forms the mapping between the current state and the action the agent should take in that state. In tabular Q-learning this policy is a look-up table, while in deep reinforcement learning parametrised policies are used. These are policies which depend on a set of parameters $\theta$, such as, for example, a neural network that depends on a set of weights. Throughout the reinforcement learning process, the goal is to adjust the parameters of the policy to be able to obtain a maximum accumulated reward or return.

When considering the class of episodic tasks, an episode or rollout is denoted by the letter $\tau$. The episode return is the sum of rewards gained throughout each step of the episode.

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{2.7}$$

Since the goal of the agent is to optimise this return, reinforcement learning algorithms often make use of discounting when calculating the expected future return. This is done using a discount factor $\gamma \in (0,1)$. While this formally helps with the convergence of the algorithm, it also makes intuitive sense. Having a $\gamma$ close to one makes it such that the agent takes future expected rewards into account heavily, whereas a low $\gamma$ leads to a greedy agent that only cares about immediate reward without much consideration for its long term future. Adding the discount factor changes the episode return to:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{2.8}$$

The optimal policy is the policy that maximises the expected return when acting according to that policy:

$$\pi^* = \arg\max_{\pi} J^{\pi} \tag{2.9}$$

Where $(\cdot)^*$ denotes the optimal variant of a certain variable or function approximator and $J^{\pi}$ denotes the expected return acting according to policy $\pi$.

### 2.3.3 Value functions

While equation 2.9 shows the formal definition of the optimal policy, it gives no way of acquiring this policy. Finding the optimal policy is done by making use of value functions. Value functions indicate how good (or bad) a certain state or state-action pair is.

The on-policy value function $V^{\pi}(s)$ describes the value of a state as the expected return if the agent were to start in state $s$ and act according to its current policy $\pi$.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s\right] \tag{2.10}$$

The action-value function, or Q-function, is defined similarly. This function describes the value of a certain state-action pair as the expected return given the agent starts in state $s$, takes action $a$ (which can be off-policy) and from then onwards acts according to its current policy:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s, a_0 = a\right] \tag{2.11}$$

Closely related to these functions are the optimal value function and optimal Q-function, which are similar but assume the agent acts according to the optimal policy:

$$V^*(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s\right] \tag{2.12}$$

$$Q^*(s,a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s, a_0 = a\right] \tag{2.13}$$

These optimal value functions are related to one another by:

$$V^*(s) = \max_a Q^*(s,a) \tag{2.14}$$

The value functions can be used by the agent as a basis for developing its behaviour. The optimal action-value function can be used to find the optimal action by taking the action that leads to the highest expected return:

$$a^* = \arg\max_a Q^*(s,a) \tag{2.15}$$

The four mentioned value functions follow the Bellman equations. These equations are used throughout reinforcement learning as they help in finding the value of a certain state $s$ when the value of the next state $s'$ is known. This can be done by noting that the value of a state is dependent on the reward the agent is likely to receive as a result of being in that state combined with the value of the state you will transition into:

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{s' \sim P} \left[ r(s,a) + \gamma V^*\left(s'\right) \right] \tag{2.16}$$

where $s' \sim P$ denotes that the resulting state $s'$ is sampled from a distribution $P$ describing the environment. While the obtained reward is $R(s,a,s')$, it is abbreviated to $r(s,a)$ for compactness.

Similarly, the Bellman equation for the optimal action-value function is found by noting that the optimal Q-value of a state is the expected value of the received reward combined with the expected Q-value as a result of the optimal future action.

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim P} \left[ r(s,a) + \gamma \max_{a'} Q^*\left(s', a'\right) \right] \tag{2.17}$$

### 2.3.4 Soft-actor critic

While this section has so far outlined the theory on which most reinforcement learning algorithms are based, no way of practically implementing these ideas has been discussed. Many different algorithms are developed that help implement reinforcement learning in a practical setting. This section aims to outline common ideas that are prevalent throughout most of these algorithms, but focusses on showing the soft-actor critic (SAC) algorithm, as originally published by Haarnoja et al. (2018b). While newer implementations of the SAC algorithm are different than this version, the presented version is the one that is implemented in Stable Baselines (Hill et al., 2018) and is used throughout this research.

Soft-actor critic, is a deep reinforcement learning algorithm that allows the agent to develop a stochastic policy in an off-policy way. A fundamental concept that is used when doing this is the use of entropy regularisation. The entropy of a distribution gives a measure of the randomness of that distribution. The entropy of a variable sampled from a uniform probability distribution is highest, as the outcome is most difficult to predict. On the other hand, flipping a heavily weighted coin is the example of a process with low entropy, as the outcome is usually unsurprising. Given a continuous random variable $x$ sampled from a probability density function $P$, the entropy is defined as:

$$H(x) = \mathop{\mathbb{E}}_{x \sim P} \left[ -\log\left(P(x)\right) \right] \tag{2.18}$$

While in many algorithms the optimal policy is the one that maximises the expected return, the optimal policy when using entropy regularisation does not only maximise the expected return, but also the entropy of the policy, such that:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\in\pi}\left[\sum_{t=0}^{T}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\big(\pi(\cdot|s_t)\big)\Big)\right] \tag{2.19}$$

The trade-off between maximising expected return and entropy is denoted by the parameter $\alpha$, the temperature parameter. The 'regular' reinforcement learning objective can be recovered from this augmented form by setting the temperature equal to 0. Tuning this parameter is not only task dependent, but the desired value can even change during training for one specific task. This is a result of the policy evolving over time, changing the expected return throughout a training session. This in turn changes the relative importance of maximising expected return and entropy, possibly leading to adverse effects on the learning process. To prevent this, the temperature coefficient can be automatically adjusted, as shown in (Haarnoja et al., 2018c). The effect of properly weighing the importance of expected return and entropy of the policy is that the agent will still quickly give up on parts of the state-space that yield little reward. Additionally, the promising parts of the state-space are broadly explored, by also taking suboptimal paths every so often. This is shown to lead to an improved exploration, reducing the agent's overall sample complexity.

As the algorithm's name suggests, the agent makes use of an actor-critic structure. This means that the agent makes use of two structures: an actor and a critic. The actor dictates how the agent should behave. The behaviour of the actor is influenced by the critic, which provides feedback to the actor based on its behaviour. This structure is shown in figure 2.4.

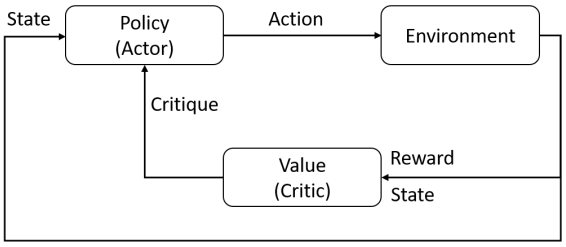



**Figure 2.4:** General structure for actor-critic based methods (Morozs, 2019).

Solving a reinforcement learning problem using the actor-critic structure comes down to optimising both the actor and the critic. For the actor to properly learn from its behaviour, the critic has to be able to properly assess the quality of a certain action in a given state. This core idea holds throughout all actor-critic based algorithms, but the precise form and implementation is variable.

### Critic networks

In soft actor-critic, the critic network is optimised using a replay buffer, $\mathscr{D}$. The replay buffer is a set of previous experiences of the agent. Whenever the agent takes an action, the state transition tuple containing the initial state, the action, the resulting reward, the resulting state and whether this state is terminal, is stored into the replay buffer. The size of the replay buffer is problem dependent and often needs to be tuned. Whenever the replay buffer is too small, only recently acquired data is used and the agent is likely to overfit. On the other hand, when the buffer is too large there may be many transition tuples that were generated by a policy far inferior to the current one, which can slow down learning. The fact that the SAC algorithm

optimises networks based on this buffer makes it such that it is an off-policy algorithm, as the transition tuples can be generated by an older version of the agent's policy.

The critic is trained using two mean-square based loss functions. The first loss function is based on the Q-value and uses equation 2.17 as a basis. The loss function gives an indication of how close the Q-value function under network parameters $\phi$ is to satisfying the Bellman equation. This loss function is defined as:

$$\text{target}_Q = r(s,a) + \gamma(1-d)V_\psi^{\pi_\theta}(s') \tag{2.20}$$

$$\mathscr{L}_Q(\phi,\mathscr{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}\left[\left(Q_\phi(s,a) - \text{target}_Q\right)^2\right] \tag{2.21}$$

A similar loss function is defined for the value function, this time using the Bellman equation for the value function as in equation 2.16 as a basis:

$$\text{target}_V = Q_\phi(s,a) - \alpha\log\left(\pi(a|s)\right) \tag{2.22}$$

$$\mathscr{L}_V(\psi,\mathscr{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}\left[\left(V_\psi^{\pi_\theta} - \text{target}_V\right)^2\right] \tag{2.23}$$

A problem that arises when updating the network parameters using equation 2.20 is that the target is a function of the network parameters of the value function, $\psi$, which in turn is a function of the Q-value network $\phi$. This can make the process of minimising these unstable. To prevent this, a second network is created that is similar to the primary network approximating the value function. This target network changes more gradually than the primary network, counteracting this instability. In the soft actor-critic algorithm Polyak averaging is used to update the target network every time the primary network is updated. The update rule for this is

$$\psi_{\text{target}} = \rho\psi_{\text{target}} + (1-\rho)\psi \tag{2.24}$$

where $\psi_{\text{target}}$ and $\psi$ are the network parameters of the target network and the primary network respectively and $\rho$ is a hyperparameter determining the similarity with the previous iteration of the target network parameters.

Another problem that occurs often is that the critic network tends to overestimate the actual value function. This can lead to problems, as basing actions on an incorrect value function can lead to undesired behaviour. In order to counter this, SAC uses two separate networks to approximate the Q-value function. The minimum of the two Q networks is used when determining the target in equation 2.22. Regressing to the smaller of the two Q-values helps preventing overestimating the value function.

**Actor network**

The policy parameters $\theta$ are determined such that they maximise the value function at each state. Since the value function is augmented with the entropy term, the policy should maximise both the expected future returns as well as the expected future entropy:

$$\max_\pi \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^\pi(s,a) - \alpha\log\left(\pi(a|s)\right)\right] \tag{2.25}$$

Ideally, equation 2.25 should be solved by performing gradient ascent with respect to the policy parameters. However, the expectation over actions, $\mathbb{E}_{a\sim\pi}$, depends on the policy parameters. This makes it often impossible to find the gradient of this expectation. To work around this issue, the reparametrisation trick is used. This entails creating a function that allows the expectation over actions to be written as something that is independent of the policy parameters, such that its gradient can be determined. In soft actor-critic, this is done using a squashed Gaussian:

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta \odot \xi\right) \tag{2.26}$$

Where $\mu_\theta$ and $\sigma_\theta$ are the mean and standard deviation of the Gaussian respectively and $\xi \sim \mathcal{N}(0, I)$. Using this, equation 2.25 can be rewritten to

$$\max_\pi \mathop{\mathbb{E}}_{\xi\sim\mathcal{N}} \left[ Q^\pi\left(s, \tilde{a}_\theta(s,\xi)\right) - \alpha \log\left(\pi_\theta\left(\tilde{a}_\theta(s,\xi)|s\right)\right)\right] \tag{2.27}$$

for which the gradient of the expectation can be found because the expectations over the actions is now rewritten to something independent of policy parameters (Kingma and Welling, 2014).

On top of this, the Q-value used in equation 2.27 should be replaced with one of the critic networks. In the case of SAC, the first Q-value network is used. This leads to the final update rule to be used in gradient ascent:

$$\max_\theta \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{N}}} \left[ Q_{\phi_1}(s, \tilde{a}_\theta) - \alpha \log\left(\pi_\theta(\tilde{a}_\theta|s)\right)\right] \tag{2.28}$$

While SAC learns a stochastic policy, Haarnoja et al. (2018b) shows that during evaluation better results are achieved when using a deterministic variant of the policy. This deterministic variant is achieved by taking the mean of the action distribution as predicted by the actor.

The pseudocode for the algorithm implemented in Stable Baselines is shown in algorithm 1. The equations used to update the Q-value and value functions are approximated by sampling transition tuples from the replay buffer.

**Pseudocode soft actor-critic algorithm**

**Initialise:**
Emtpy replay buffer: $\mathscr{D}$
Actor network: $\pi_\theta$
Critic networks: $Q_{\phi_1}$, $Q_{\phi_2}$, $V_\psi$
Target value network: $V_{\psi_{\text{target}}}$

**while** *not converged* **do**
Observe state $s$ and select action $a \sim \pi_\theta(\cdot|s)$
Execute $a$ in the environment
Observe resulting state $s'$, reward $r$ and done signal $d$
Store transition tuple $(s, a, r, s', d)$ into the replay buffer $\mathscr{D}$
If $s'$ is terminal, reset the environment state

**if** *time to update* **then**
**for** *each iteration* **do**
Randomly sample a batch $B$ of transitions tuples from the replay buffer

Update value function using one step of gradient descent:

$$\nabla_\psi \frac{1}{|B|} \sum_{s \in B} \frac{1}{2} \left( V_\psi(s) - \min_{i=1,2} Q_i^{\pi_\theta}(s, \tilde{a}) - \alpha \log\left(\pi_\theta(\tilde{a}|s)\right) \right)^2 \tag{2.29}$$

Update the Q-functions using one step of gradient descent:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,s',a) \in B} \frac{1}{2} \left( Q_{\phi_i}^{\pi_\theta}(s, a) - \left( r(s, a) + \gamma V^{\pi_\theta}(s') \right) \right)^2 \tag{2.30}$$

Update policy using one step of gradient ascent:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left( Q_1^{\pi_\theta}\left(s, \tilde{a}_\theta(s)\right) - \alpha \log\left(\pi_\theta(\tilde{a}_\theta(s)|s)\right) \right) \tag{2.31}$$

Update target value network:

$$\psi_{\text{target}} = \rho \psi_{\text{target}} + (1 - \rho)\psi \tag{2.32}$$

**end**
**end**
**end**

**Algorithm 1:** Pseudocode for the soft actor-critic algorithm. Adapted from (OpenAI, 2018) based on (Haarnoja et al., 2018b).

## 2.4 Behaviour cloning

The sample complexity of a model-free learner can be decreased by initialising the agent with prior knowledge about the task rather than having to learn it from the ground up (Nagabandi et al., 2017). One way of giving the agent prior information is to use behaviour cloning. This section outlines the theory for the specific behaviour cloning procedure used in this work.

During behaviour cloning, it is the goal for a student to clone the behaviour of an expert. To do this, the expert executes a task on the environment recording all action and observations. The goal of the student is to learn how to act in the same way as expert, by taking the same action given the same observation. The actor network is trained to act similarly to the expert in the context of an actor-critic architecture. Treating the actions as labels and the observations as inputs, this problem can be formulated as a supervised learning problem. The goal when doing supervised learning is to minimise a defined loss function. This loss function gives a measure of how different the policy of the student is compared to the behaviour of the expert. While a number of different loss functions could be used (Pajarinen et al., 2018), the specific implementation used in this thesis minimises the mean square difference between the action shown by the expert demonstrations and the action predicted by the policy. Assuming the student network is parametrised using $\theta$, the goal of behaviour cloned is defined as

$$\arg\min_{\theta} \sum_{(\mathbf{s}_t, \mathbf{a}_t) \in D} \left(\mathbf{a}_t - \mathbf{a}_t^*\right)^2 \tag{2.33}$$

Where $D$ is a dataset containing expert transitions and $\mathbf{a}_t^*$ the action predicted by the student policy.

# 3 Methods

This chapter provides an overview of the methods used to answer the research questions. The first section shows a schematic diagram representing the general outline of the process. The second section shows the experimental setup and its capabilities. The third section shows the methods used to develop a system model using minimal modelling time. The fourth section shows how this system model can be used to create a policy with the aim of reducing learning times. The last section outlines the methods used for deploying the soft actor-critic algorithm in simulation and on the experimental setup.

## 3.1 Approach

An overview of the entire end-to-end process demonstrated in this research is show in figure 3.1. This image shows three phases:

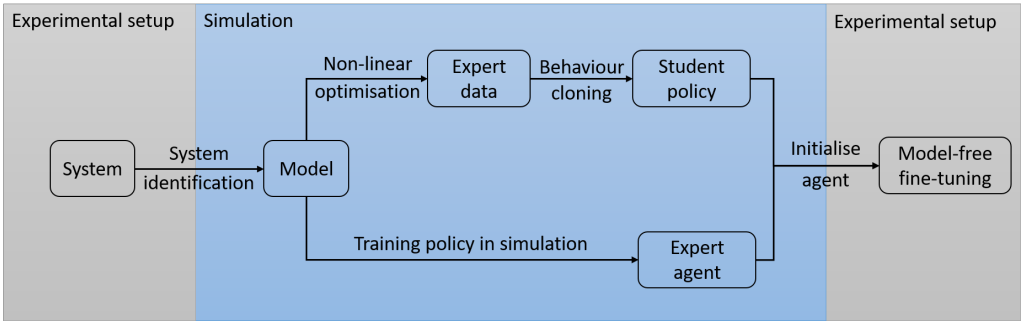



**Figure 3.1:** Overview of the end-to-end process.

**Phase 1: Creating a system model**
A system model is created using system identification. In this work, the system is modelled using a NARX neural network trained with experimental data acquired using the setup.

**Phase 2: Creating model-based policies**
Policies are developed based on the created system model. This is done in two different ways. The first method entails creating expert demonstrations by using non-linear optimisation to find the optimal trajectory between points in the endoscope's workspace. These demonstrations can then be used in combination with behaviour cloning to develop a student policy that mimics this behaviour. In the second method an agent is trained directly on the obtained system model, resulting in a policy capable of fulfilling the desired task in simulation.

**Phase 3: Fine-tuning the policy**
Rather than initialising the model-free agent using random weights, the model-based policies can be used to warm-start the agent. This allows the agent to focus on fine-tuning the obtained policies rather than learning form zero knowledge, aiming to reduce the sample complexity. The effectiveness of warm-starting the agent is evaluated by comparing the sample efficiency of the different methods of initialisation with the sample efficiency of the agent when trained without prior knowledge.

## 3.2 Experimental setup

The experimental setup designed and constructed by Hoitzing (2020) is used as a basis for the experiments throughout this research. This section aims to outline the already existing and newly added features of this setup. The section concludes with a naming convention and setup specific definitions.

### 3.2.1 System overview

The setup consists out of three main parts: a tendon-driven continuum manipulator, the endoscope, a construction to actuate and measure the tendons driving the endoscope and an electromagnetic (EM) based position tracking system manufactured by NDI Medical, see table 3.1 for the hardware specifics. Figure 3.2 shows an overview of the setup containing these three elements.

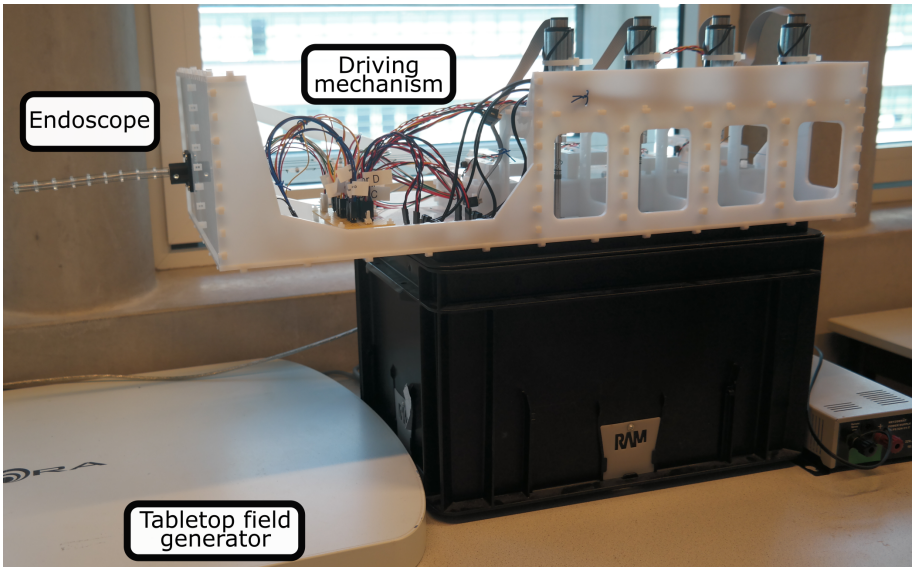



**Figure 3.2:** Total system overview including the endoscope, driving mechanism and tabletop field generator that measures the endoscope's tip position.

#### Endoscope

The main point of interest for this research is the endoscope. The endoscope is held in place by a holder and actuated by four tendons, as shown in figure 3.3. Spacers are used to route the tendons along the endoscope. These tendons terminate in a knot such that force can be applied to the end of the endoscope, allowing control of the endoscope's tip position.

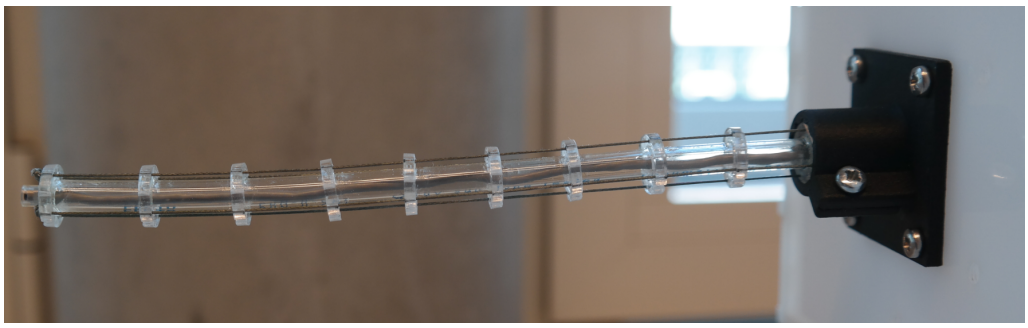


**Figure 3.3:** Hollow endoscope with EM tracker routed through the backbone mounted on the setup.

Hoitzing (2020) designed, fabricated and tested a 3D printed endoscope. The endoscope's tip position was sensed using two 5D EM probes attached to the endoscope using a mounting piece. However, due to the fragile nature of these sensors in combination with the high acceleration motion used in this project, these sensors were deemed unsuitable. Instead, a single, sturdier, 6D probe is used. This probe is used in conjunction with a tabletop field generator that generates a varying electromagnetic field. The varying field induces currents in the tip of the sensor that are used to determine the 6D pose of the sensor at a fixed frequency of 40 Hz, see table 3.1 for the specific hardware.

The disadvantage of using this more robust sensor is that it has bulkier cable compared to the two 5D probes. While the added weight should not inhibit an RL agent from developing an adequate policy, the implications on the state transitions might. Mounting the sensor on the endoscope's tip using a mounting piece will leave the sensor's wire hanging in the air. Violent changes in direction or oscillatory behaviour of the endoscope will result in this cable moving. Given its weight, this is likely to influence the dynamics of the endoscope in a chaotic manner. This means that the state transition becomes partly dependent on the state of the cable, which is hard to deduce from the available sensor data. This additional uncertainty is undesired as it will slow down the learning process. To prevent this, the endoscope's design is changed such that the sensor's cable can be routed through the backbone.

In contrast to the existing endoscope, a backbone with a sufficiently large hollow centre and an outer diameter compatible with the rest of the setup cannot be 3D printed. Instead, pneumatic tubing made from polyurethane was used. The outer diameter of the backbone is 6 mm and the inner diameter is 4 mm. The diameter of the probe is 1.8 mm which makes it such that the sensor can be removed and inserted easily. A connector piece is used to fix the sensor to the end of the backbone, limiting all motion of the sensor with respect to the backbone.

The spacers were designed similarly to Hoitzing (2020), but were laser cut out of 2 mm thick acrylic instead of being 3D printed. Due to the inability to 3D print the endoscope, these spacers were manually attached to the backbone. A slit was made in the spacers to ensure the orientation of each spacer is the same. The resulting endoscope is shown in figure 3.3.

**Driving mechanism**

Figure 3.4 shows a top-down view of the driving mechanism. The image shows four DC motors that are used to move the tendons. These motors are equipped with quadrature encoders that measure the shaft angle, which is converted to displacement of the tendons. The tension of the tendons is measured using four load cells. These load cells use strain gauges to measure the force with which the tendon pushes onto the pulley, allowing a measurement of the tension at a rate of up to 80 Hz. The tension readings are amplified before being sent to the microcontroller. The low level control is done on a microcontroller, which controls the motors via motor driver breakout board and communicates with a laptop running the high level control using ROS.

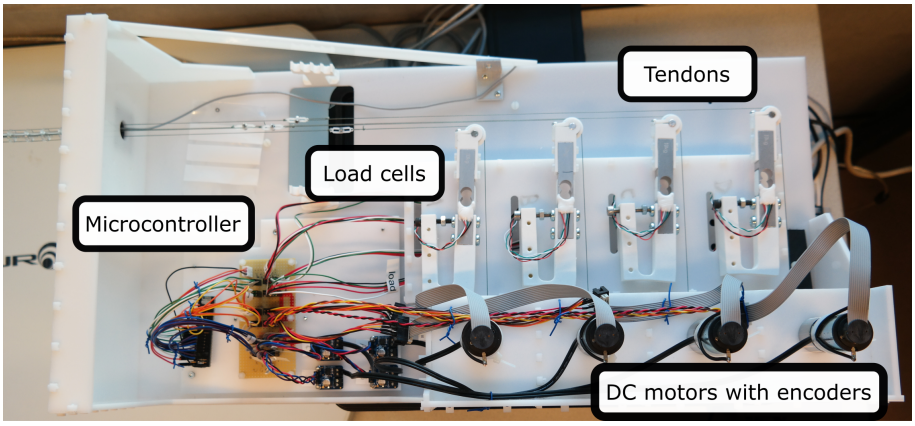
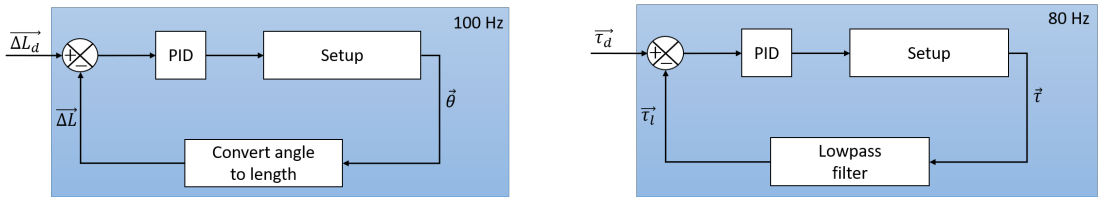



**Figure 3.4:** Top view of the endoscope's actuation mechanism.

### Control modes

The low level control of the setup can be done using one of two mechanisms at any given time: control based on the tendon displacement as adapted from Hoitzing (2020) or based on the tendon tension. The block diagrams for both low level control schemes are shown in figure 3.5.




**(a)** **(b)**

**Figure 3.5:** Low level control schemes used to control the setup. (a) Low level tendon displacement control. (b) Low level tendon tension control.

The tendon displacement controller is the primary controller used to move the endoscope around. This controller can quickly and accurately reach a desired setpoint. The tendon tension controller is used to reset the setup to its neutral state. This means that the tendon tension is the same after each reset, but the endoscope's position can differ as a result of effects such as hysteresis. In the context of this thesis, resetting to an equal tendon tension is more important than resetting to the same location. This is because the high level control maps tendon displacement to endoscope position, which changes for varying initial tendon tension.

### 3.2.2 Setup specific definitions

Instead of presenting the actual value measured by the EM tracker, normalised coordinates are used wherever possible. These coordinates are obtained by mapping the endoscope position between a value of 0 and 1 using the extrema of the endoscope's position, which are measured as a result of the maximum tendon displacement of 8 mm in both the positive and negative direction. On one hand, the goal is for the obtained results to be independent of the absolute position. While there might be position dependent effects, expressing this dependency in the coordinates of the EM tracker carries little physical meaning. On the other hand, the use of normalised coordinates allows for better interpretation of variables such as the error signal as their significance is better interpreted when compared to the size of the endoscope's workspace.

While the model used throughout this report is data-driven and makes few assumptions about the experimental setup, interpretation of the results benefits from a more thorough description

of the system. Figure 3.6 shows the tip of the endoscope. The tendons exerting force on the tip are named A through D. Throughout this research, these tendons are driven as antagonistic pairs. This means that whenever tendon A is displaced with $x$, tendon D is displaced with $-x$. This makes the system go from an overactuated system (four tendons and three-dimensional positions), to an underactuated system. This choice is made because the setup can be adapted to drive a two segment endoscope by making each motor drive a pair of tendons rather than a single one. The demonstrated methods should also be applicable when controlling a two segment endoscope in this way.

Figure 3.6 also shows the orientation of the EM tracker's coordinate frame with respect to the endoscope's tip. While this orientation is in principle variable, the endoscope is kept on a fixed location at the right side of the tabletop field generator throughout the experiments. Note that only the orientation of the shown coordinate frame is accurately depicted as the origin of the EM tracker's coordinate frame is located at the centre of the field generator.

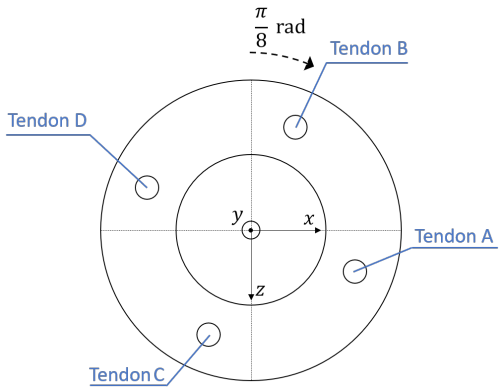



**Figure 3.6:** Tendon identifiers and their relative orientation with respect to the EM tracker's frame.

### Hardware

Table 3.1 shows an overview of the hardware used in the experimental setup.

| Hardware | Source |
|---|---|
| Tabletop field generator | NDI medical - Part number 090024 https://www.ndigital.com/medical/products/aurora/ |
| EM probe | NDI medical - Part number 610016 https://www.ndigital.com/medical/products/tools-and-sensors/ |
| Microcontroller | PJRC - Teensy 3.5. https://www.pjrc.com/store/teensy35.html |
| Motor drivers | Adafruit - Part number DRV8871. https://www.adafruit.com/product/3190 |
| DC motor | Maxon - Part number DC-max B7A2257A0377 https://www.maxongroup.com/maxon/view/category/motor |
| Load cells amplifier | SparkFun - Part number HX711 https://www.sparkfun.com/products/13879 |
| Load cells | HTC-sensor - TAL220 http://www.htc-sensor.com/products/94.html |

**Table 3.1:** Overview of the hardware used in the experimental setup.

## 3.3 System identification

A model of the system is created with the goal of increasing the RL agent's learning speed. The process of creating this model, system identification, is outlined in this section. The system identification procedure is divided in stages: data acquisition, data pre-processing and model training.

### 3.3.1 Data acquisition

Data about the system is required to be able to train a NARX network. In this case, a set of three-dimensional position measurements $X = \{\mathbf{x}[1], \ldots, \mathbf{x}[T]\}$ as a result of a four-dimensional tendon displacements $U = \{\mathbf{\Delta L}[0], \ldots, \mathbf{\Delta L}[T-1]\}$ is required. This data is acquired using the control scheme shown in figure 3.7.

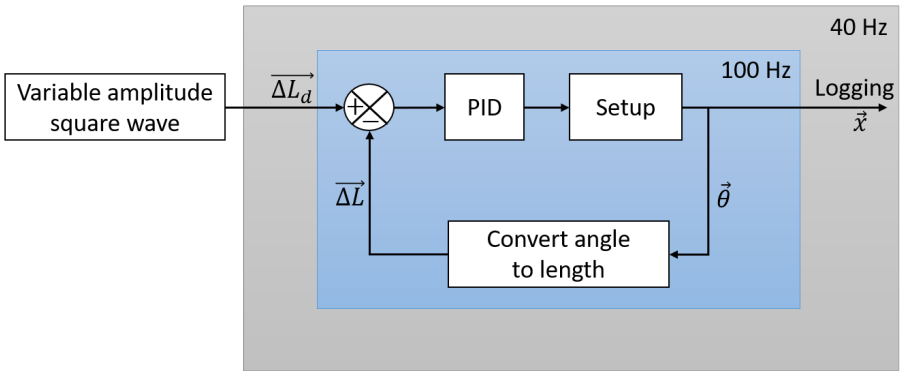



**Figure 3.7:** Block diagram of control architecture during system identification.

#### Input signal

One of the requirements when doing system identification is that the input signal used should be sufficiently exciting. To find a suitable input signal, the step response of the system is examined. A Fourier transform of this input showed that the frequencies up to 5 Hz can reasonably be expected to be excited during normal operation. Exciting these frequencies is done using a square wave with varying amplitude. This type of signal is used because it most closely resembles the input the reinforcement learning agent will be allowed to give. During normal operation, the agent will directly send desired tendons displacements, $\mathbf{\Delta L}$ to the lower level controller. In the worst case, the input signal provided by the reinforcement learning controller would thus resemble a square with varying amplitude. While in principle there are four degrees of freedom due to the four available tendons, the system is controlled to move using only two independent actuators, as explained in *Section 3.2.2 - Setup specific definitions*. An independent sequence of square wave amplitudes is generated for both pairs.

The limits of the amplitude are empirically determined to a maximum of 8 mm, meaning that each tendon can be displaced by a maximum of 8 mm in both the negative and positive direction. The amplitude of the square wave are sampled from a uniform distribution and change every 10 position measurements, making it such that frequencies up to around 5 Hz are likely to be excited. Using the block diagram as shown in figure 3.7, an identification sequence of 15 minutes, 36000 samples, as well as a verification sequence of 2 minutes, 4800 samples, are executed on the experimental setup.

### 3.3.2 Preparing data for training

**Sensor data**

The endoscope's tip position is read at a frequency of 40 Hz using the EM tracker. Every time a position measurement $\mathbf{x}$ is received, a new input $\mathbf{\Delta L}$ is sent to the setup. Assuming that the frequency with which the EM tracker publishes position data is stable, this results in a set of positional data as a result of input data with a fixed $\Delta t$ between each sample. However, at high levels of acceleration the EM tracker does not give accurate position readouts, returning an invalid position instead. While the tracker can be set up to wait until a valid position is found, this would yield to a sequence of states that is unevenly spaced in time. Since the NARX network assumes a constant time difference between states, the invalid position measurements are reconstructed in a post-processing step.

The system identification process is an open-loop process in the sense that the system behaviour is not dependent on position provided by the EM tracker. This makes it such that a complete identification run can be done without rectifying any missing measurements. After all samples have been collected, the invalid transformations are corrected based on the valid measurements temporally surrounding it. This correction is based on a second order Taylor expansion where the derivatives are estimated using backward finite difference method. Rather than determining a point based on only the past, information from the future, which is available since this is a post-processing step, can be added to yield a better result. Given a set of correctly measured positions $X = \{\mathbf{x}[k-1], \mathbf{x}[k-2], \mathbf{x}[k+1]\}$ the position of $\mathbf{x}[k]$ is found using:

$$\mathbf{x}[k] = \frac{2}{5}\mathbf{x}[k+1] + 2\mathbf{x}[k-1] - \frac{1}{2}\mathbf{x}[k-2] \tag{3.1}$$

Since equation 3.1 uses three known values to compute a single unknown value, this principle can be extended to correct for three consecutively failed measurements, albeit at the cost of a larger estimation error. The reconstruction method for two and three consecutively missing points can be found in *Appendix A - Reconstructing sensor data*.

In the case a larger number of measurements fails consecutively, the missing positions are interpolated using a second order polynomial based on the two successful measurements before and the first successful measurement after the set of failed measurements.

**Preprocessing data**

In addition to post-processing the sensor data, both the input data and position data is preprocessed before being used to train the NARX network. Both these sets of data are normalised to a range from 0 to 1 using the extrema found in the training set. Instead of normalisation, standardisation of the data was also tested. This is a pre-processing step in which the data is transformed to have 0 mean and unit standard deviation. This showed little differences in performance. Normalisation was chosen as the preferred method since the mean of the data was prone to vary over time due to hysteresis effects, as will be discussed in *Section 4.1 - Hysteresis*. Methods to reduce the dimensionality of the problem such as principle component analysis and selecting features based on a correlation heatmap were tested but showed worse results.

### 3.3.3 NARX training

The system model is created using a NARX neural network. The set of NARX models is a subset of the set of non-linear autoregressive moving average with exogenous inputs (NARMAX) models. The set of NARMAX models is widely used in black-box system identification due to its formulation covering a large set of non-linear systems (Rannen et al., 2016). Despite not explicitly taking noise into account, like the NARMAX models, NARX models have been shown

to sometimes perform equally well for some situations, while providing a simpler model structure (Acuna et al., 2012). Additionally, NARX models have been proven effective in modelling non-linear systems and they are shown to be better at discovering long term behaviour than regular recurrent neural networks (Diaconescu, 2008).

The advantage and disadvantage of a data-driven approach is the abstraction from the physical system. While the outlined method is applied to the setup shown in figure 3.2, little to no information about the system is used in order to identify it. This makes it such that this method could be applied to a wide variety of systems, requiring only little system specific fine-tuning. However, the drawback of this approach is that little physical insight in the system is gained performing this system identification. This makes picking suitable control algorithms or making changes to the system based on desired dynamics more difficult.

### NARX parameters

The system identification data is used to determine the input delays and feedback delays used when training. The autocorrelation of the measured positions is used to determine the amount of feedback delays, as this gives an indication how many previous states are relevant when determining the next one. Similarly, the cross-correlation between the actuation and the measured position is used to find the amount of input delays that are relevant. The resulting plots of autocorrelation of the x position as well as the cross-correlation between the x position and tendon D are shown in figure 3.8.

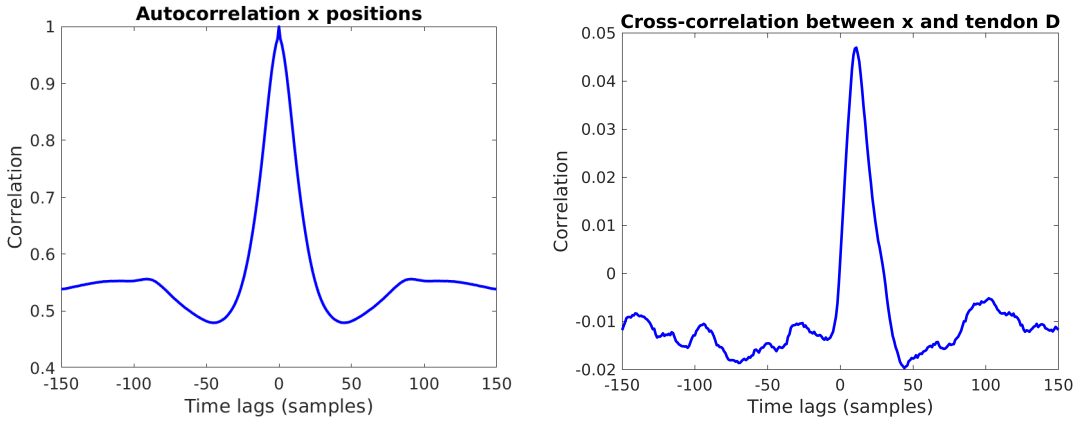



**Figure 3.8:** Autocorrelation and cross-correlation plots used to determine NARX parameters. (a) Autocorrelation of the tip's x position. (b) Cross-correlation between the x position and displacement of tendon D.

Figure 3.8a, shows a wide peak around 0. For this reason, the feedback delays that are used are 1 through 10. While a larger range could have been considered based on the autocorrelation plot, training was slowed down significantly or sometimes made impossible due to the larger memory footprint.

The figure showing the cross-correlation, figure 3.8b, shows a peak at 11 samples delay in the cross-correlation between the x position and tendon D. This indicates that the measured position correlates most with the input 11 samples ago. Similar behaviour is observed when looking at the cross-correlation between the other tendons and directions. For this reason, the input delay was initially chosen to be between 8 and 12 samples. However, after performing system identification the obtained results were better when changing to input delays between 1 through 10.

### 3.3.4 Network architecture

The NARX network is constructed and trained using MATLAB. A simplified network architecture is shown in figure 3.9. This figure shows only one node per input or output dimension to prevent cluttering.

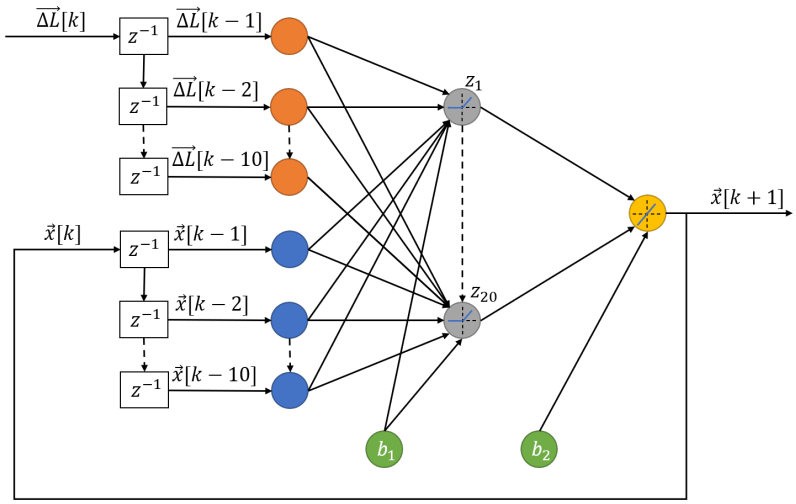



**Figure 3.9:** Simplified parallel NARX architecture. Adapted from (Kumar and Murugan, 2018).

As can be seen in figure 3.9, the network has an input layer that takes a two-dimensional input. The input features physically represent the displacement of tendon A and B. The displacement of tendon C and D are left out as these provide no new information and can be deduced from the displacements for tendon A and B. The next layer is a hidden layer with 20 hidden neurons that uses a rectified linear unit (ReLU) as the activation function. The layer after this is the output layer that has a linear activation function and produces a three-dimensional position. In principle the EM tracker's orientation measurement could have been included in the output vector. However, this proved to degrade the performance of the position estimation.

### 3.3.5 Training procedure

The training of the network is done in two stages. During the first stage, the network is used in the series-parallel architecture. This training aims to minimise the single-step prediction error. Training is done using Bayesian regularisation, which minimises the weighted sum of the squared network weights and the squared error using Levenberg-Marquardt optimisation. In addition to this weight regularisation, the network training is stopped early when a certain performance is achieved or the gradient is too low. The combination of these methods aims to counter the weights from becoming too large, which is often a sign of overfitting on the training data, degrading the network's ability to generalise.

In the second stage the network is closed, maintaining the weights found during training for single-step prediction, and trained further. This training is meant to decrease the multi-step prediction, which is required to simulate the endoscope dynamics. In the multi-step prediction situation, training is done using Levenberg-Marquardt backpropagation with the data split up into 70%, 15% and 15% training, validation and test set respectively. The data is not shuffled when splitting, as this would remove the temporal correlation in the dataset. In addition to early stopping, the network's performance is also tested on a physically different validation set. This is done to see if the network has overfit on some behaviour or offset that was specific to the system identification run.

### 3.3.6 Validation experiments

The network performance is validated after it has been trained. To do so, the network is given a sequence of inputs for which the movement of the endoscope needs to be predicted. This sequence of inputs is run on the setup during a separate validation experiment. This will give insight on the network's ability to predict the endoscope's trajectory even when the endoscope has been reset in between. In addition to predicting the validation sequence, a secondary validation test is run where the network is used to predict a step response. This validation step aims to give additional insight in the model accuracy.

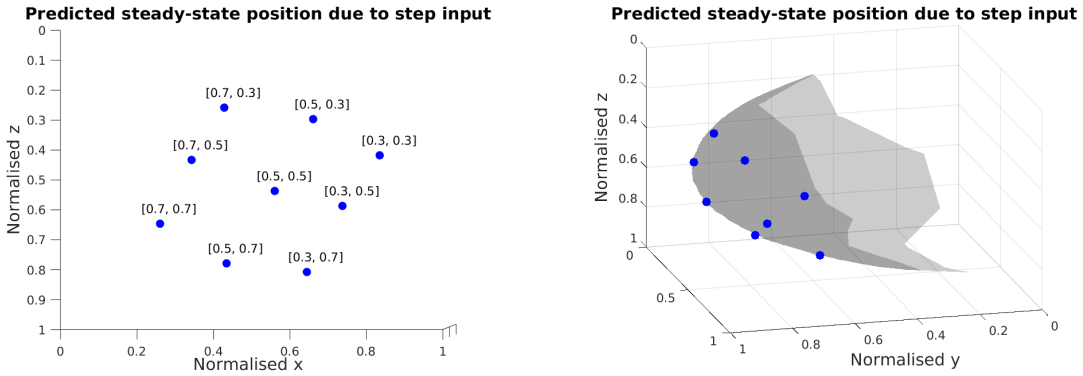
## 3.4 Behaviour cloning

The soft actor-critic algorithm is a model-free reinforcement learning algorithm, which means that a model of the system is not required for the algorithm to work. However, the model created in *Section 3.3 - System identification* can be used to initialise the model-free learner in an attempt to reduce its learning time (Nagabandi et al., 2017). This section outlines the method for generating the expert data and demonstrates how a policy is created that mimics this behaviour.

### 3.4.1 Generating expert trajectories

Expert demonstrations are a sequence of inputs that execute the given task sufficiently well. The source of these expert trajectories can vary widely: from a human operating the system to a proficient (PID) controller. In this case it is chosen to use a non-linear optimisation routine to find the optimal trajectory. An advantage of this method is that the routine can be instructed to minimise a cost function that is in accordance with the agent's reward function. Additionally, such an optimisation routine has the same data-driven nature as a NARX model. This makes this method widely applicable without the need to develop a competent, system specific, controller to acquire the expert demonstrations. The latter is especially useful in the case of soft continuum manipulators, where controllers are difficult to develop given the unavailability or complexity of the system model.

An important consideration when generating the expert trajectories is the information that is to be transferred using the expert trajectories. In this case, the goal of the expert demonstrations is to give the student a general idea of how to behave throughout the workspace. This behaviour consists of two parts: moving from point A to B and standing still anywhere in the workspace. Ideally, a large number of expert trajectories would be generated densely covering the entire workspace. However, this is practically not feasible due to the computational resources this would cost. Instead, a number of points is chosen in the endoscope's workspace between which trajectories are going to be generated. In this case a trajectory is considered to go from point A to point B as well as to go from point A to point A, the latter demonstrating how to stand still. The points between which to generate trajectories are shown in figure 3.10a. Ideally these points are spread through the entire workspace. Unfortunately, some iterations of the NARX network were improperly defined at the edges of the workspace. This lead to heavy oscillations for constant input at some locations in the workspace. This behaviour was non physical, as retraining the network shifted or removed regions in which this phenomenon was observed. To minimise the underlying model as a source of error for the generated trajectories, the points were picked closer to the origin. The implication of this is that the student policy cannot reasonably be expected to develop proper behaviour outside of the region spanned by the chosen points. While for practical application of this method it should be ensured that the points are spread through the entire workspace, the points used can serve as a proof of concept.




**(a)** **(b)**

**Figure 3.10:** Steady-states as a result of different step inputs. (a) Front view showing predicted steady-state position as a result of the displacement of tendon A and tendon B. (b) Side view. Gray surface represents the measured data.

In order to generate the expert trajectories a cost function must be defined. Since the goal is for these trajectories to help accelerate the learning of the student policy once it is deployed, the cost function used for generating the expert trajectories should be in line with the reward function used in the reinforcement learning algorithm. If this were to not be the case, the supervised learning procedure to learn the expert behaviour would still work, but the generated policy would not optimise the expected return when deployed in the reinforcement learning environment.

The duration of a trajectory $T$, the control horizon, is set to 1 second. This duration allows the endoscope to reach any desired location and get to a stand-still. The total cost of a trajectory $\tau_i = \{\mathbf{x}_{i_0}, \ldots, \mathbf{x}_{i_T}\}$ is the sum of the Euclidean distance between the endoscope's tip position $\mathbf{x}_t$ and the goal position $\mathbf{x_d}$ at each time during the trajectory:

$$L(\tau_i) = \sum_{t=0}^{T} \|\mathbf{x}_{i_t} - \mathbf{x_d}\| \tag{3.2}$$

However, using this cost function will yield a single optimal trajectory between two points. Having a larger set of trajectories between points is beneficial, as this increases the locations in which the student is shown how to behave. A secondary term is added to the cost function to accommodate this. This term tries to maximise the minimum distance between the already existing trajectories, up to a limit $\alpha$:

$$\begin{aligned} L(\tau_i) &= \sum_{t=0}^{T} \|\mathbf{x}_{i_t} - \mathbf{x_d}\| - \min([e_{\tau_1}, \ldots, e_{\tau_N}, \alpha]), \\ \text{where } e_{\tau_j} &:= \sum_{t=0}^{T} \|\mathbf{x}_{i_t} - \mathbf{x}_{j_t}\| \end{aligned} \tag{3.3}$$

This limit for $\alpha$ is empirically determined to be 1, with higher values leading to a wider spread in trajectories and lower values leading to a more narrow distribution.

The expert trajectories are generated using MATLAB's fmincon() function running the 'interior-point' algorithm. This function can be used to solve a constrained non-linear multivariable function. The general form of these equations is (MathWorks, 2020):

$$\arg\min_{x} f(x) \text{ s.t.} \begin{cases} c(x) \leq 0 \\ c_{\text{eq}}(x) = 0 \\ A \cdot x \leq b \\ A_{\text{eq}} \cdot x = b_{\text{eq}} \\ lb \leq x \leq ub \end{cases} \tag{3.4}$$

Where $A$ and $A_{\text{eq}}$ are matrices, $b$ and $b_{\text{eq}}$ are vectors, $lb$ and $ub$ are the lower and upper bound for $x$ respectively and $f(x)$, $c(x)$ and $c_{\text{eq}}(x)$ are (non-linear) functions. For this case, the matrices $A$, $A_{\text{eq}}$ and vector $b$ and $b_{\text{eq}}$ are set to zero. The lower and upper bound are set at a maximum of 8 mm tendon displacement in both the positive and negative direction. The function $f(x)$ is the non-linear function as defined in equation 3.3. This function is used to generate three trajectories between all possible combinations of the points shown in figure 3.10a, resulting in a total of 243 trajectories.

Some of the generated trajectories are tested on the experimental setup to investigate their robustness against model inaccuracies. These trajectories are run five times to get insight on their repeatability. Note that these are run in open-loop configuration, so the trajectory is not recalculated based on the sensor information acquired during the rollout.

### 3.4.2 Behaviour cloning

The result of the generated expert trajectories is a set of inputs corresponding to a set of endoscope positions. However, for the agent to be able to develop a policy based on the expert data, the data should be a combination of action and observations. To obtain observation-action pairs from the optimal actions, the expert inputs are run on the modelled environment. This environment is explained in detail in *Section 3.5.1 - Gym environment*. Executing the expert actions on the environment yields the same trajectory the expert experienced, but ensures the observations are in line with those defined by the gym environment. Having to deploy the expert trajectories on the RL environment could have been avoided if the MATLAB optimisation routine was extended to yield the observations associated to the expert trajectory. However, it is chosen to keep the reinforcement learning specific choices contained in the gym environment.

The set of action-observations pairs is used to train a policy using the supervised learning scheme outlined in *Section 2.4 - Behaviour cloning*. The trained student policy is then executed on the experimental setup to find its performance.

## 3.5 Reinforcement learning

The high level control is done using reinforcement learning. The used methods are tested using the model created in *Section 3.3 - System identification* and then executed on the experimental setup. The agent's policy is initialised using different methods to find the effects of the initialisation on training time. The control scheme for the high level controller is shown in figure 3.11.

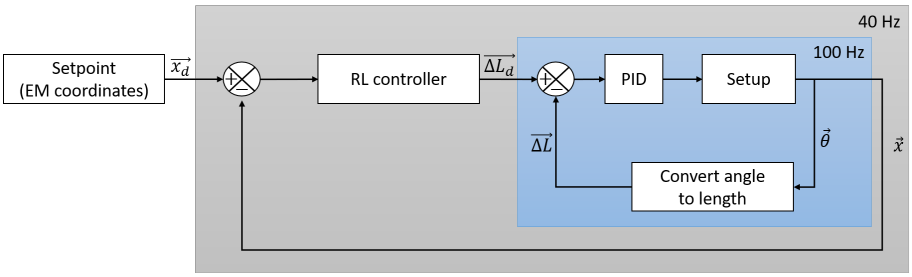



**Figure 3.11:** Block diagram of high level control architecture.

### 3.5.1 Gym environment

An implementation of the SAC algorithm shown in algorithm 1 is provided by Stable Baselines (Hill et al., 2018). Stable Baselines is written in Python and is a fork of OpenAI Baselines that provides the implementation for several state-of-the-art deep reinforcement learning algorithms. Within Stable Baselines, all deep reinforcement learning algorithms make use of gym environments. Gym environments form an interface between the (simulated) dynamics and the reinforcement learning algorithm. This separation allows for the comparison of different algorithms or a multitude of training sessions of the same algorithm without any major changes to the code. This section summarises the most important aspects of the gym environment used throughout the reinforcement learning experiments.

#### Environment physics

The core of the gym environment is allowing the RL agent to step through time. To this end, the NARX network obtained in section *Section 3.3 - System identification* is used. Using Python to call MATLAB functions predicting the network's output resulted in slow simulations. Instead, the network parameters were exported and a Python function utilising these parameters in matrix form as shown in equation 2.2 was used to predict the system state. This resulted in an increase of speed by almost 100 times compared to the initial method, making training more viable.

#### Observation space

The observation space defines what part of the system the agent is able to observe. While in simulation there often are few hidden system parameters, in practice the observation space is usually limited by available sensor data. Therefore, only data that would be available on the experimental setup is chosen to be included in the observation space. At its basis, the observation space is:

$$\mathbf{o}[k] = \left[\mathbf{x}[k], \mathbf{u}[k-1], \hat{\mathbf{g}}[k], d[k]\right] \tag{3.5}$$

where $\mathbf{x}$ is the endoscope's position in Cartesian coordinates, $\mathbf{u}$ the agent's previous action, $\hat{\mathbf{g}}$ the unit vector pointing from the endoscope's tip to the goal position and $d$ the distance to the goal.

A more state invariant observation space can be achieved by leaving out the absolute position. However, due to position dependent effects, such as stiction between the tendons and the spacers, using only the direction to the goal provides the agent with insufficient information about the environment.

The observation in equation 3.5 makes the system non-Markovian, as the future observation is dependent on more than the current observation. To more closely approximate a Markov decision process, the observation vector is augmented with previous observations. This allows for approximations of for example velocity and acceleration (Haarnoja et al., 2018a), (Mnih et al., 2013). Given the knowledge that the NARX network governing the endoscope's behaviour uses a maximum of 10 delayed states, the agent is given the past 10 observations to be the full observation.

**Reward function**

The focus of this research is on showing a reduction in training time by initialising the policy using the created model. This is why a simple reward function is chosen over an intricately shaped one, despite the latter likely allowing for a decrease in sample complexity compared to the former. The chosen reward function is:

$$R(\mathbf{x}) = -\|\mathbf{x} - \mathbf{x_d}\| \tag{3.6}$$

This reward function is monotonically increasing and has a maximum of zero when $\mathbf{x} = \mathbf{x_d}$, driving the system towards the desired state.

**Integration with the setup**

When deployed on the experimental setup, the environment dynamics are not governed by the NARX network. Instead, the observations are based on sensor input. To accommodate for this, the gym environment is changed such that the learning algorithm is synchronised with the EM tracker's readings. Directly after receiving a value, the algorithm publishes the action determined based on the previous observation. The computation based on the received value is done in the remaining time until the next value is received. While this makes it such that the agent operates on a single step delay, this reduces the jitter caused by varying computation times.

As mentioned in *Section 3.3.2 - Sensor data*, the sensor is not always able to publish a valid measurement at a rate of 40 Hz. While the data could be reconstructed using a post-processing step, a different method must be used for online reconstruction. Whenever a policy is executed that is based on simulated data, the missing values are estimated using:

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mathbf{x}[k] - \mathbf{x}[k-1] + \frac{1}{2}\big(\mathbf{x}[k] - 2\mathbf{x}[k-1] + \mathbf{x}[k-2]\big) \tag{3.7}$$

with the output being clipped between the maximum coordinates of the workspace. This method is used despite the fact that the assumption of constant acceleration between steps is violated. A safer alternative could be to use the previous data point rather than estimated one using this method. However, the former method cannot be used because the simulated policies can develop an I-like behaviour, increasing the magnitude of the output if the previous action resulted in insufficient motion. When a policy is trained without being given any prior knowledge, the EM tracker is set up to only return valid data points. Despite the resulting measurements having a varying frequency in regions of high acceleration, this resulted in better performance.

### 3.5.2 Training

Two different policies are distinguished when training an agent: a local and a global policy. A local policy allows the agent to from the neutral position to a single end position, whereas the global policy allows the agent to go from any start position to any position in its workspace.

#### Training in simulation

For the agent to develop a global policy, the full workspace has to be explored. The training procedure is set up to facilitate this. When training a global policy, the start and goal locations are randomly generated for every episode from the list of known state/action combinations shown in figure 3.10a. The start and goal location are not necessarily a different location, meaning that some episodes are focussed on learning how to stand still. Each episode has a fixed length of 200 samples, corresponding to five seconds of real time.

The policy's performance is evaluated 100 times per training session. During evaluation, the endoscope has to follow a predefined trajectory consisting out of 12 goal locations, which are similar, but unequal, to the points used throughout training. This evaluation trajectory takes a total of 60 seconds and is essentially a series of episodes chained together that form a path throughout the entire workspace spanned by the points trained on.

While this reference trajectory gives insight in how well the agent can move through the workspace, it is not going to be used in practice. A different reference trajectory is used to test the practical usefulness of a learned policy. This trajectory uses the same points as the aforementioned trajectory, but interpolates between the points using skew sines. This resembles someone using for example a joystick to move the reference position.

### 3.5.3 Decreasing sample complexity using simulation

The effects of the different methods of initialisation are investigated by training a local policy on the experimental setup. The training process is similar to training in simulation, albeit this time without randomising a starting location and goal location for each episode and using the adaptations to the gym environment outlined in *Section 3.5.1 - Integration with the setup*.

The agent is trained on the setup using three different methods of obtaining the initial policy. The first method is the standard method, where the policy is initialised randomly. This method forms a baseline for the learning speed and performance. The second method is using the student policy acquired in *Section 2.4 - Behaviour cloning*. The hypothesis is that this method allows the agent to reach a higher performance in the same number of training samples, as the initial learning is sped up due to the observed expert demonstrations. The third method uses the fully trained policy obtained in simulation. This method is expected to have a similar effect to the method based on behaviour cloning, but should provide even better results because the ideal behaviour is determined for an even larger part of the state-space. Even though the latter method is theorised to be superior in term of performance to the former, the former circumvents the need for extensive training sessions in simulation. Despite not being done in this work, the expert trajectories could also be deployed on the experimental setup when acquiring observations associated to the expert actions to reduce the model bias as suggested by Thuruthel et al. (2019).

Two sets of experiments are done to verify whether the learning process can be sped up using the created model. The first set of experiments aims to evaluate the performance of the policies used to initialise the model-free agent. This is done by running both the student policy as well as the simulated RL expert on the setup. During this experiment, both agent's are given a single goal location. This location is part of the set of points determined using the NARX network. The disadvantage of this is that these points can be physically unreachable due the modelling

errors in the network. However, the ability to reach these points gives an idea of how the robust the created policies are against modelling errors.

In the second set of experiments the agent is initialised using the three described methods. A local policy is then trained for each of these three methods for 300 episodes each with a length of 200 steps. The agent's performance is evaluated 13 times throughout the training session. This is done by running an episode in which the deterministic version of the agent's policy is used to go to the goal location. The evolution and final performance of the agents that were warm-started is compared to the baseline performance of the randomly initialised agent to determine the effect of the different methods of initialisation on the sample complexity.

### 3.5.4 Global policy on the experimental setup

A global policy is trained on the experimental setup. This policy is trained without any prior knowledge and serves as a proof of concept demonstrating the agent's ability to control the endoscope throughout the workspace. The method of training a global policy is similar to the method used in simulation. However, instead of having a predefined series of start and goal locations, this list is generated on the experimental setup at the start of a training session. This is done by measuring the position resulting from a number of tendon displacements. These inputs are chosen similarly to those shown in figure 3.10 to ensure a large coverage of the endoscope's workspace. At the start of each episode, an initial tendon displacement and goal position are generated from the created list. The evaluation trajectory is constructed out of this same list, using skew sines to interpolate between the points. Since a three-dimensional evaluation trajectory can consist out of unreachable positions due to the method of constructing the evaluation trajectory, only the x and z directions are used when rewarding the agent.

The agent is trained for 500 episodes each with a random start and goal location and a length of 200 samples. The agent is evaluated 20 times over the course of its training.

# 4 Results

This chapter shows the results achieved following the start-to-end process of creating a reinforcement learning controller. This includes the observed hysteresis and the performance of the obtained model, the policies used to initialise the reinforcement learning controller and the reinforcement learning controller when trained without prior knowledge.

## 4.1 Hysteresis

The main point of improvement for the redesigned endoscope was to make sure the sensor could be routed through the backbone. However, the new material used to fabricate the endoscope showed memory-like effects. An experiment is performed to show these effects. In this experiment, the endoscope is reset five times, then held in an upright, stressed, position. After an hour in this position, the endoscope is reset another five times. The average resetting position for the endoscope before and after this hour are shown in figure 4.1.

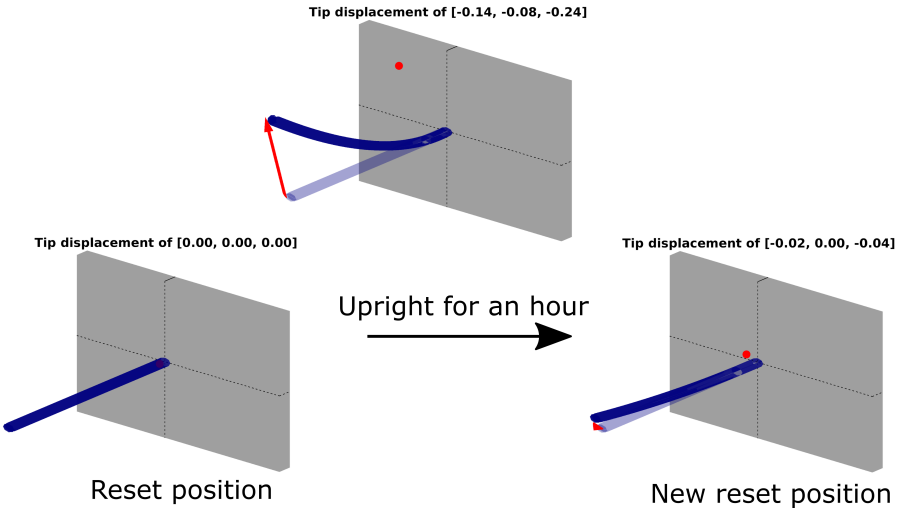


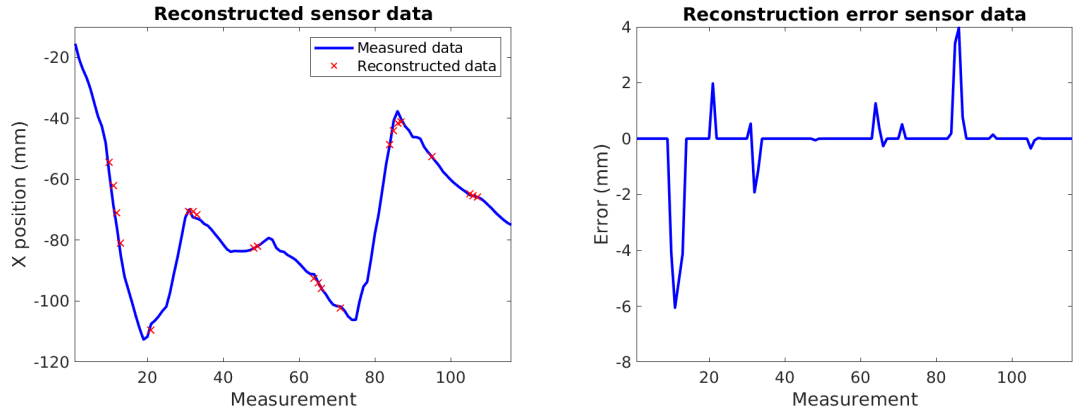

**Figure 4.1:** Effect of hysteresis present in the newly designed endoscope.

## 4.2 NARX results

This section shows the performance of the methods shown in *Section 3.3 - System identification.* This is done by first showing the error induced by the construction of the failed position measurements, after which the performance of the NARX is evaluated.

### 4.2.1 Interpolation sensor data

The invalid sensor readouts are constructed following the procedure in *Section 3.3.2 - Sensor data.* The performance of this post-processing step is determined by reconstructing a number of data points out of a series successful measurements. These results are shown in figure 4.2.




**(a)** **(b)**

**Figure 4.2:** Performance of the sensor data construction in the post-processing step. (a) Set of successful measurements in which a number of data points is reconstructed. (b) Error for the reconstructed data points.

### 4.2.2 NARX network

The network is tested on its ability to predict the endoscope's tip position given a sequence of tendon displacements. Figure 4.3 shows measured position and the predicted position using both the series-parallel and parallel network architectures. These predictions are for a physically different run with data that was not used in the training of the network. Invalid sensor readouts that were constructed in the post-processing step are marked. To ensure visibility of the targets and predictions, figure 4.3 only shows the starting portion of the two minute validation run. The full run results are shown in figure B.1.

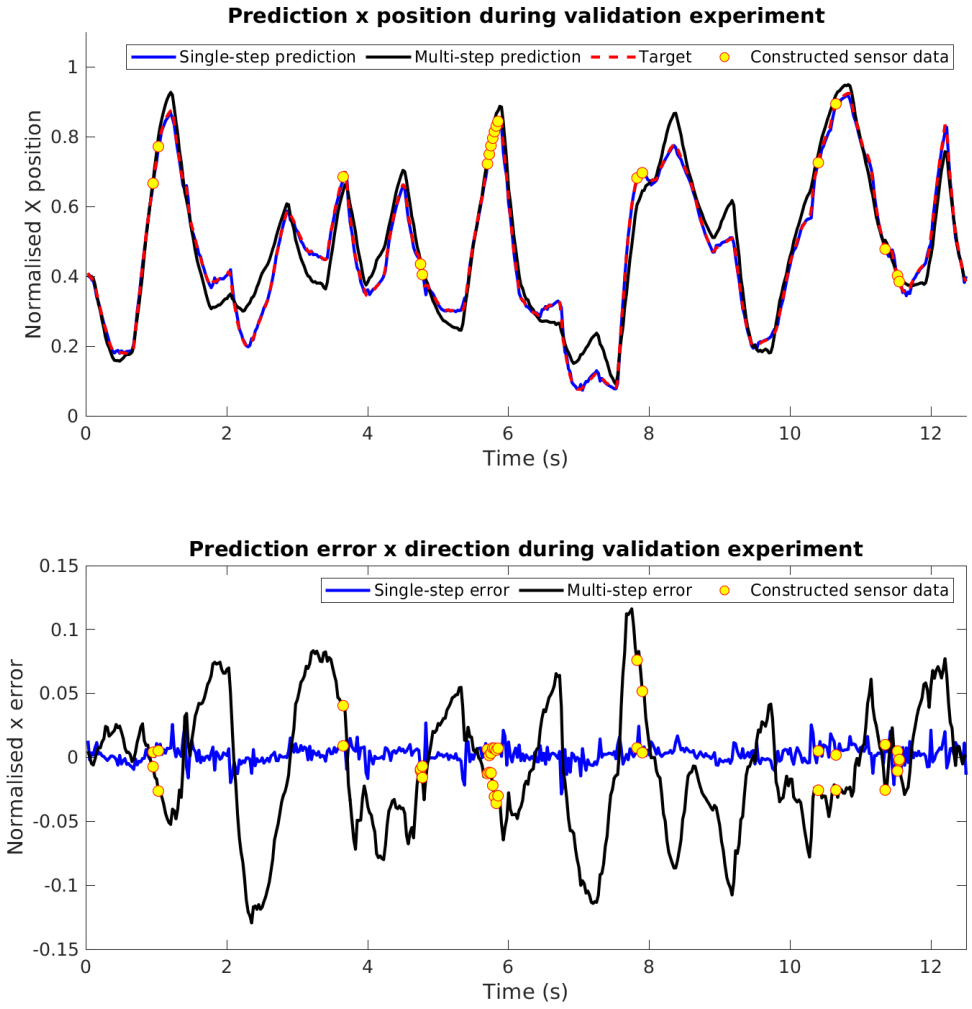



**Figure 4.3:** Performance of the NARX network using the single-step predictions, multi-step predictions and targets. Top: prediction for x direction. Bottom: prediction error.

The NARX network is also used to predict a step input. A comparison between the simulated and measured step response is shown in figure 4.4.

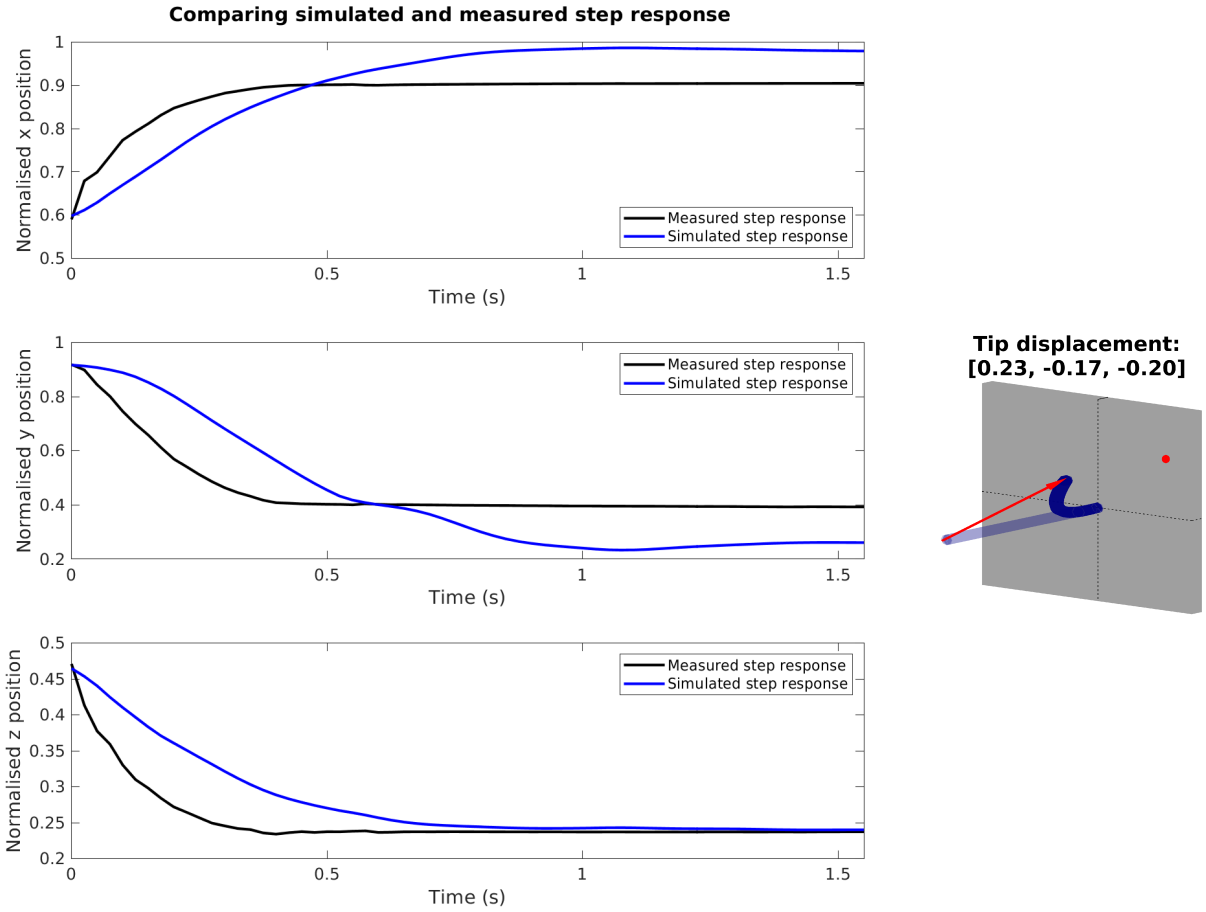



**Figure 4.4:** Comparing the simulated and measured step response.

## 4.3 Expert policies

This section shows the results achieved by the policies used to initialise the model-free agent. The expert trajectories used to base the student policy on are first tested in simulation. The resulting student policy is then evaluated on the experimental setup in both open-loop and closed-loop mode. The expert RL agent is evaluated the same way, although the simulated results are shown in *Section 4.4 - Reinforcement learning*.

### 4.3.1 Expert trajectories in simulation

Figure 4.5 shows three expert trajectories that have been generated between two points in the endoscope's workspace. These trajectories are obtained in simulation.

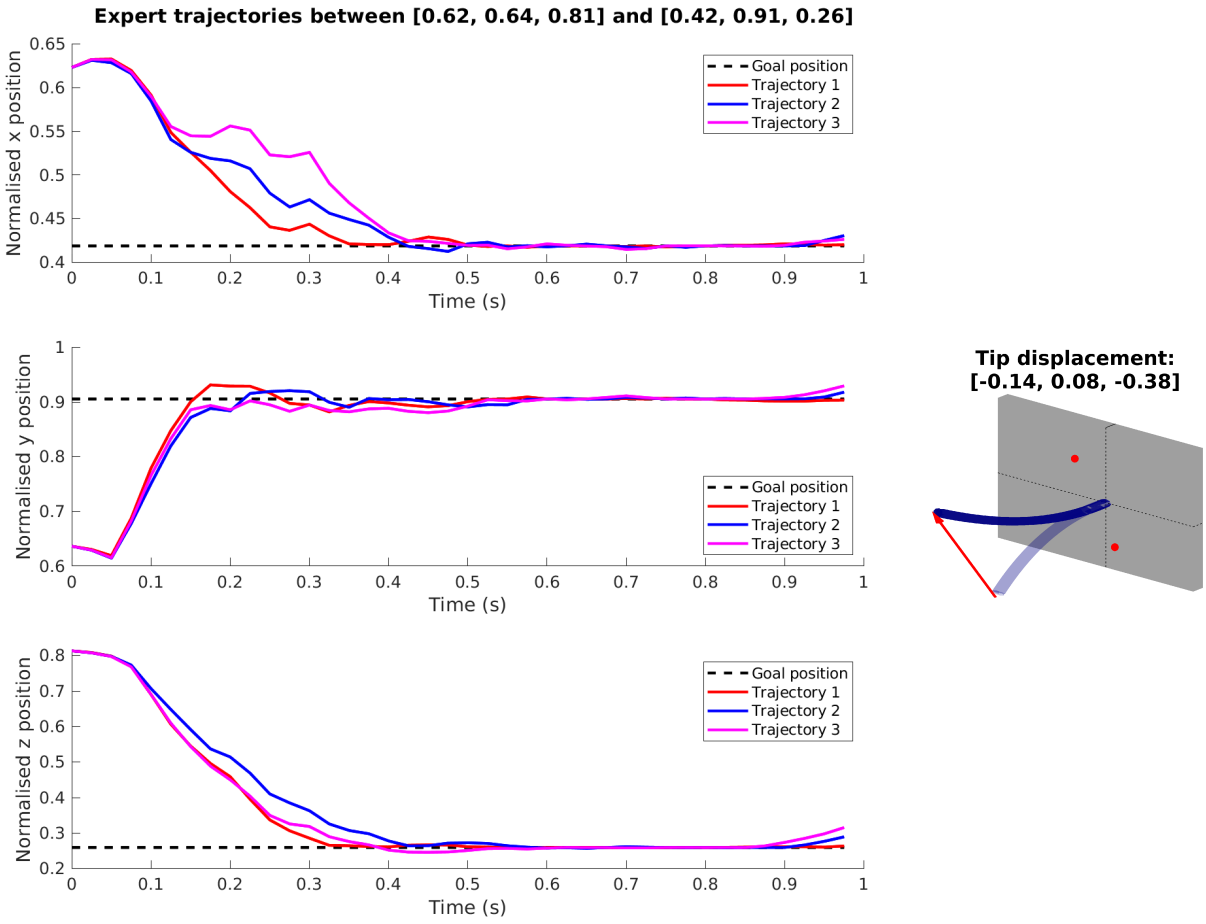



**Figure 4.5:** Generated expert trajectories between two points in the endoscope's workspace.

### Expert trajectories on the experimental setup

An expert trajectory was executed on the experimental setup five times. This expert trajectory was not recalculated based on observations and the sequence of inputs was thus the same for every rollout. The resulting positions as well as the goal position is shown in 4.6.

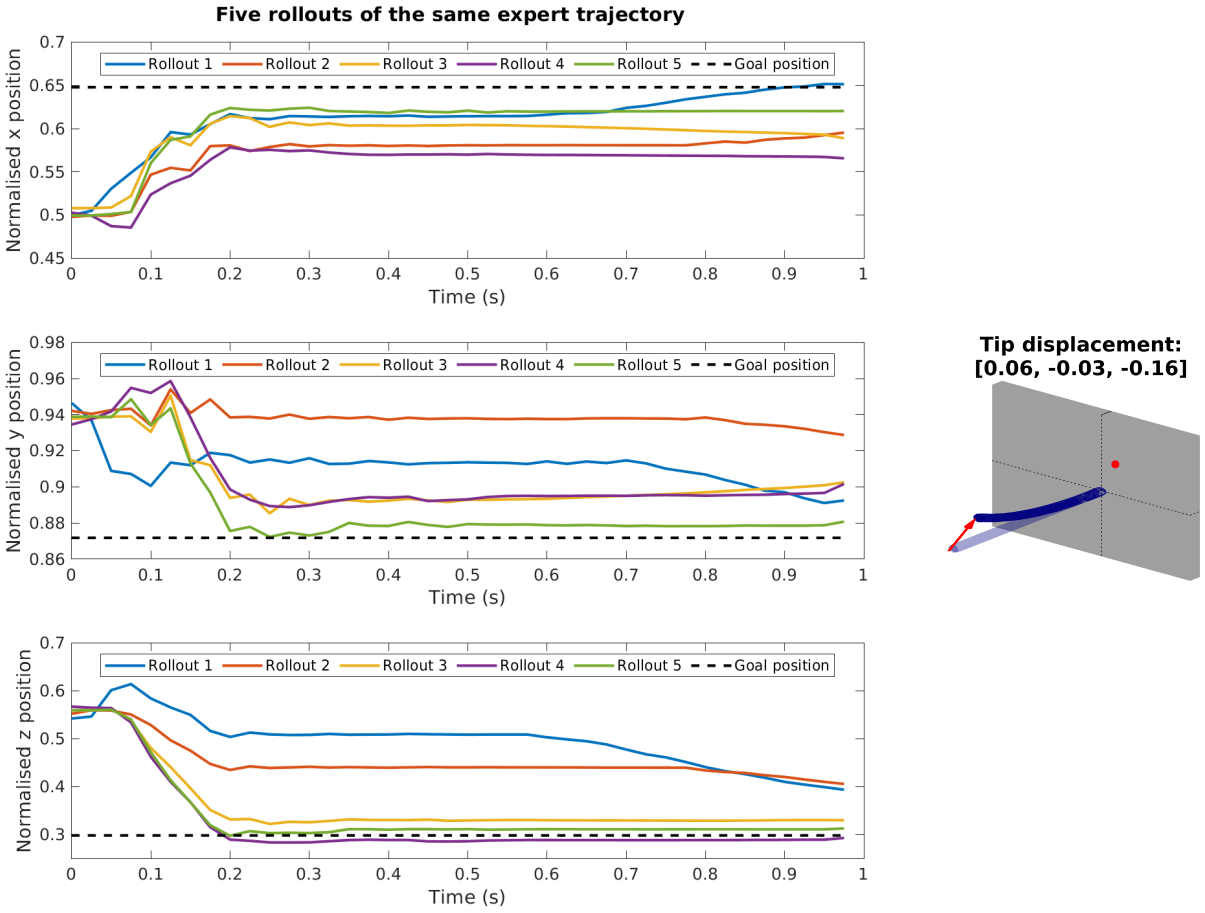



**Figure 4.6:** One expert trajectory run on the experimental setup five times.

The simulated RL expert is used to determine an expert trajectory between the two same points as used in figure 4.6. The predetermined sequence of inputs was run on the experimental setup for five runs. These results are shown in figure 4.7.

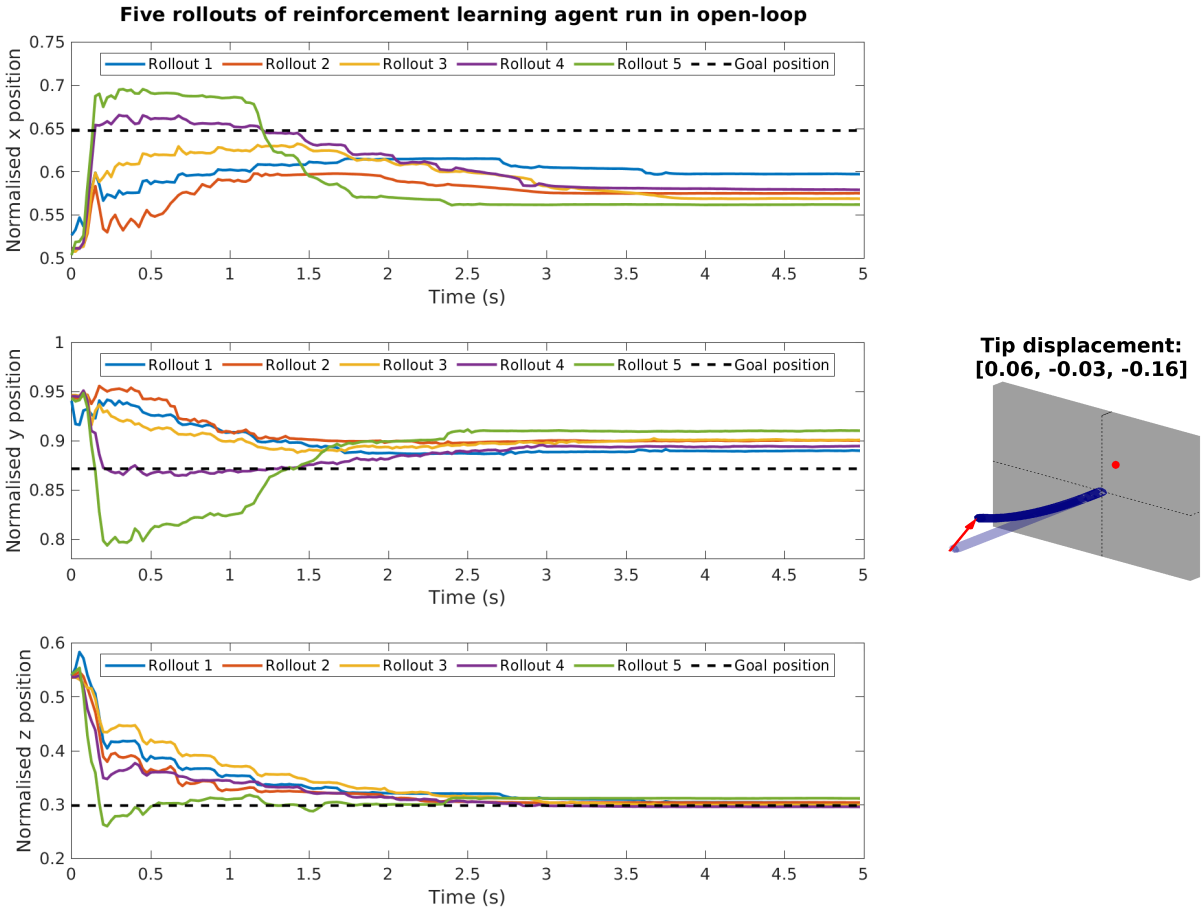



**Figure 4.7:** One RL agent generated trajectory run on the experimental setup five times.

### 4.3.2 Initial policies

The behaviour cloning algorithm is used to create a student policy. An agent utilising this policy in closed-loop mode is used to control the experimental setup. The resulting rollout is shown in figure 4.8.

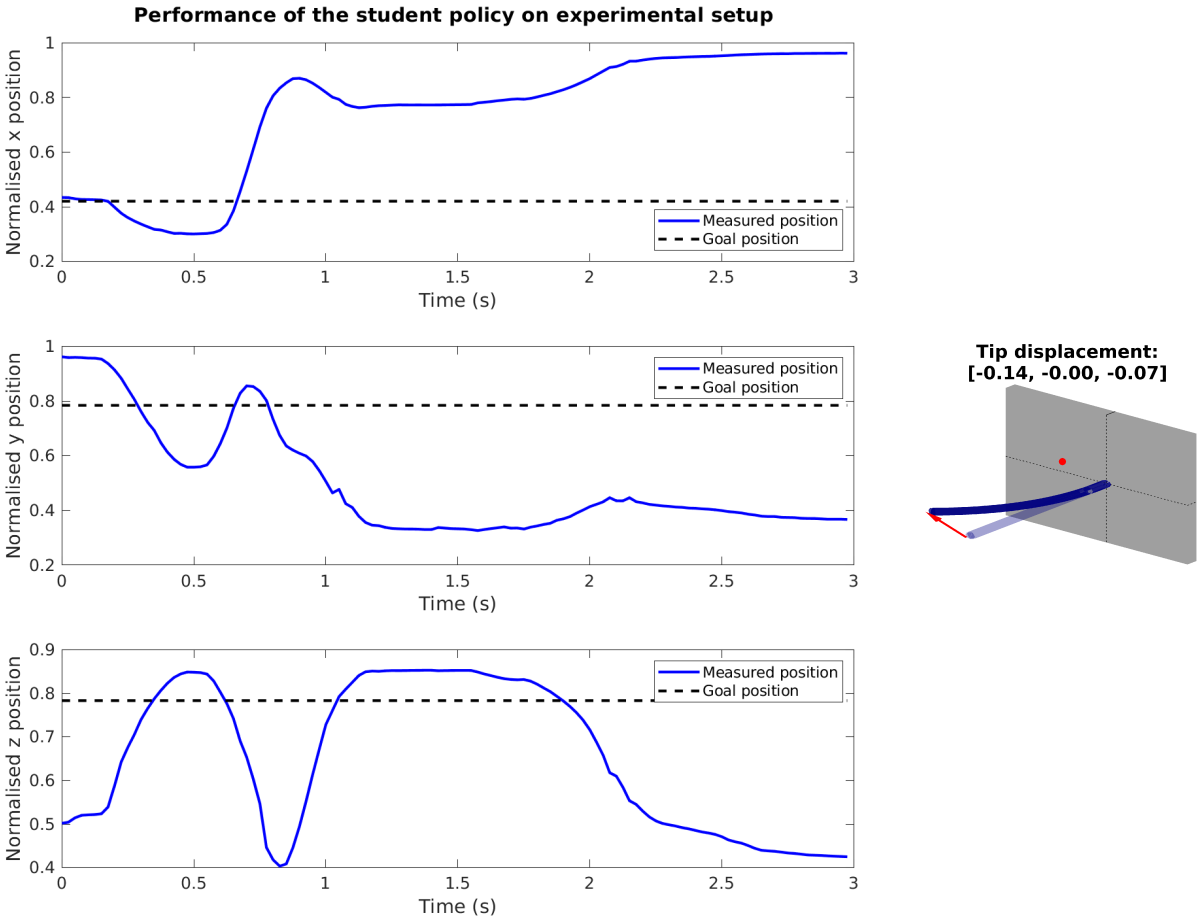



**Figure 4.8:** Performance of the student policy on the setup.

The simulated RL expert is also used to initialise the model-free RL agent. The performance of this agent is shown in figure 4.9.

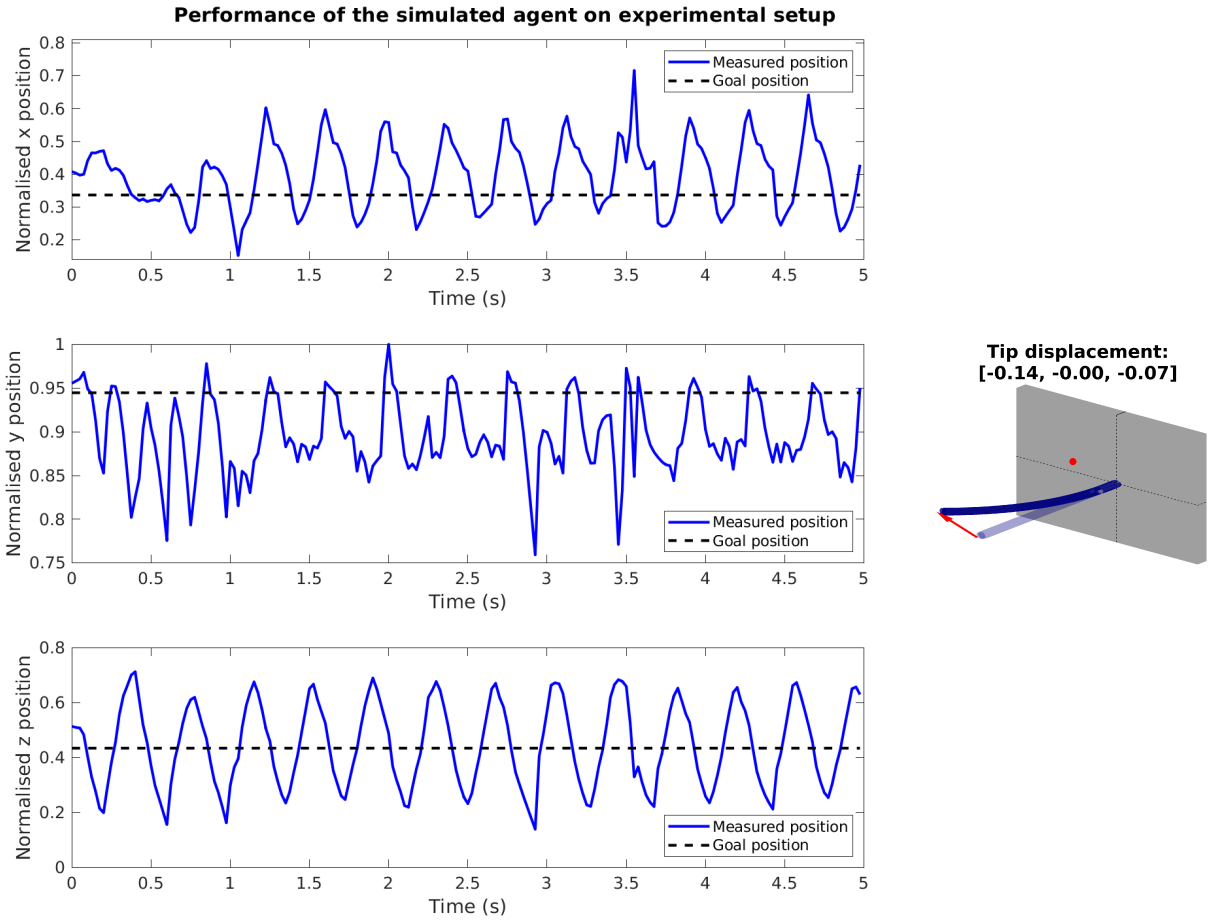



**Figure 4.9:** Performance of the global RL expert policy on the setup.

## 4.4 Reinforcement learning

This section shows the results of the reinforcement learning controller. The controller is first shown to work in simulation, after which the results on the experimental setup are shown. These results include training the agent from scratch as well as initialising it using the student policy and RL expert's policy. After showing the effects on the sample complexity, the performance of an agent that has learned a global policy without any prior knowledge is shown.

### 4.4.1 Simulated RL expert

A global policy is trained in simulation. Figure 4.10 shows how the performance of the agent evolves over time.

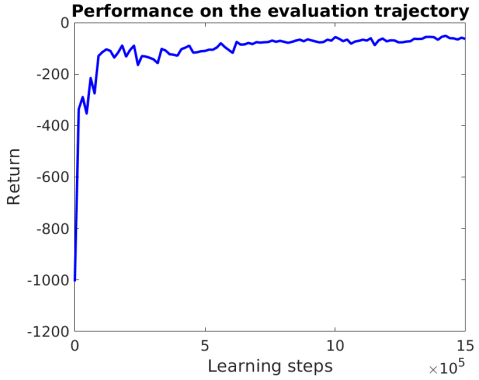



**Figure 4.10:** Performance of the simulated agent throughout a single training session.

The rollout with this highest return on the validation trajectory is shown in figure 4.11.

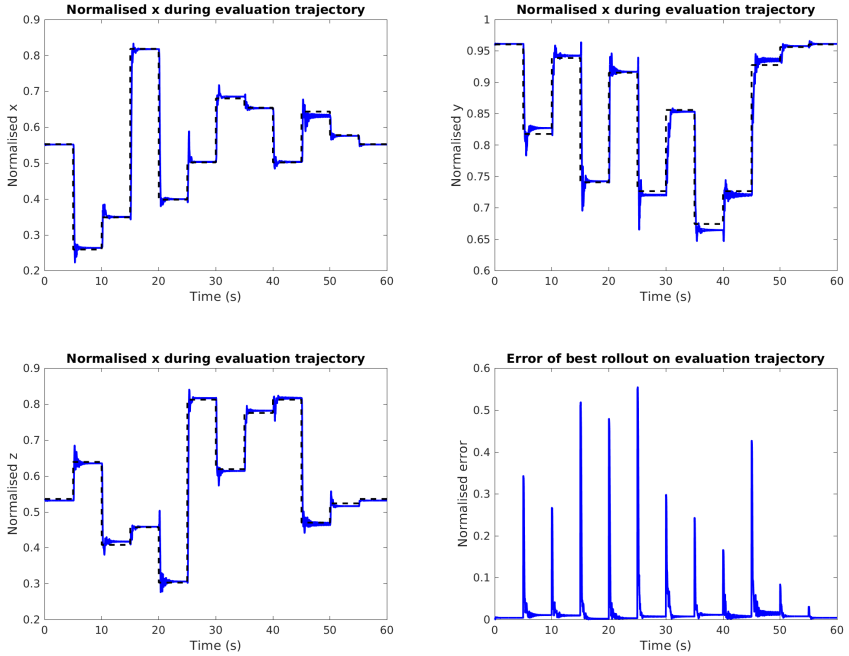



**Figure 4.11:** Performance of the rollout with the highest return on the validation trajectory. Top-left) Trajectory data for the x-direction. Top-right) Trajectory data for the y-direction. Bottom-left) Trajectory data for the z-direction. Bottom-right) L2 norm of difference between goal and endoscope's tip position.

The agent's policy that lead to the highest return on the validation trajectory is used to track a more practically relevant reference profile. Figure 4.11 shows the agent's capability to follow a trajectory that passes through the same points as the trajectory shown in figure 4.11, but this time the trajectory between points is interpolated using skew sines.

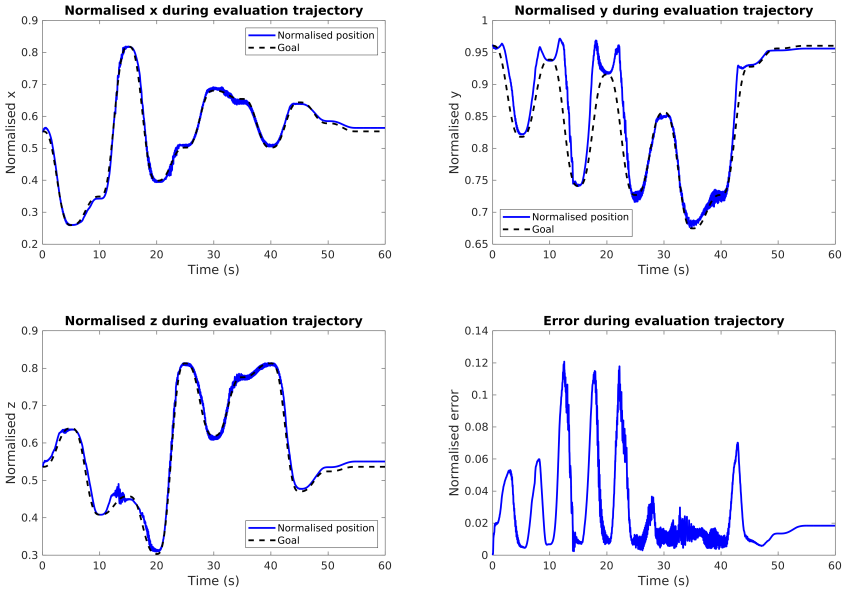



**Figure 4.12:** Performance of the best rollout on the validation trajectory consisting interpolated with skew sines. Top-left) Trajectory data for the x-direction. Top-right) Trajectory data for the y-direction. Bottom-left) Trajectory data for the z-direction. Bottom-right) L2 norm of difference between goal and endoscope's tip position.

### 4.4.2 Different methods of initialisation

The performance of the randomly initialised agent is compared to that of the agent initialised with the student policy and the one with the simulated RL expert. The evolution of the performance over training time for all these three methods is shown in figure 4.13.

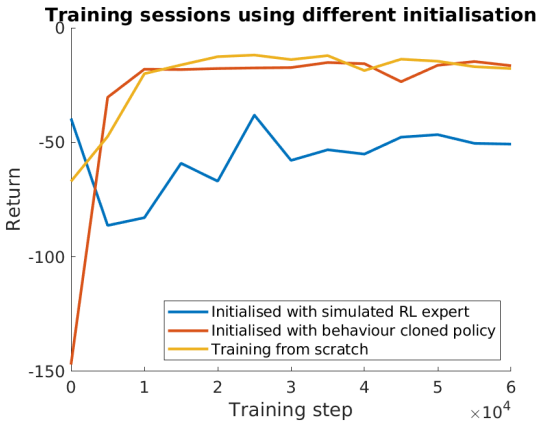



**Figure 4.13:** Comparing different methods of initialisation.

The last rollout of the evaluation trajectory for each of the differently initialised agents is shown in figure 4.14.

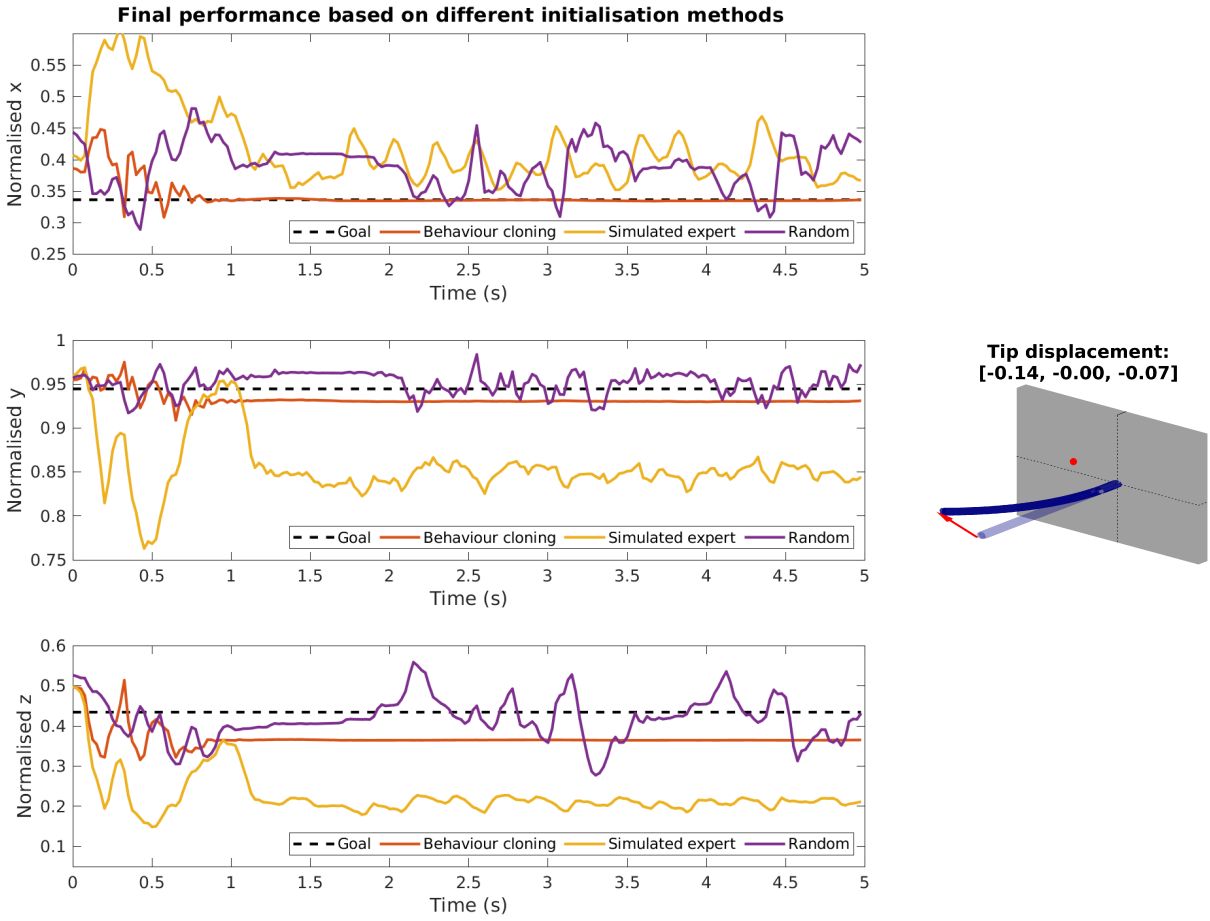



**Figure 4.14:** Comparing the policy performance after 300 episodes of training for different initialisation methods.

### 4.4.3 Learning without prior knowledge

An agent is trained without prior knowledge to judge its ability to reach a certain location in space. The evolution of its performance is shown in figure 4.15 and the best rollout is shown in figure 4.16.

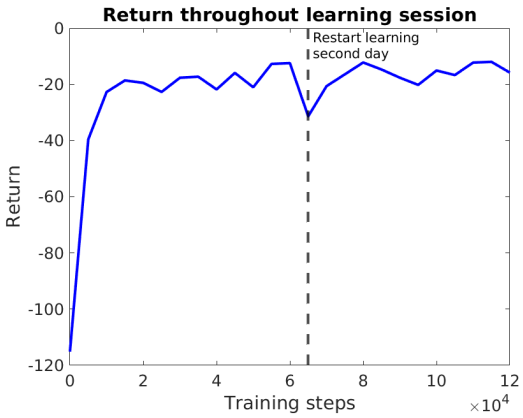



**Figure 4.15:** Performance throughout a training session developing a local policy on the experimental setup.

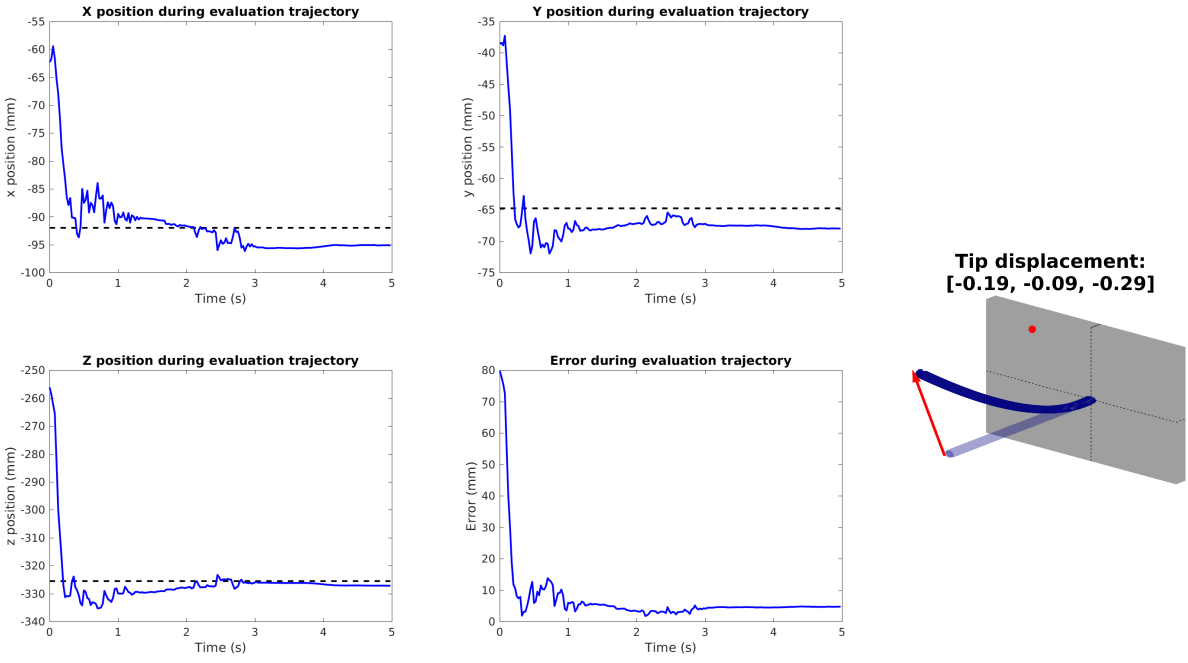



**Figure 4.16:** Performance of an agent following a local policy after training without any prior knowledge for 500 episodes.

A different agent is trained without any prior knowledge to develop a global policy. The training results are shown in figure 4.17.

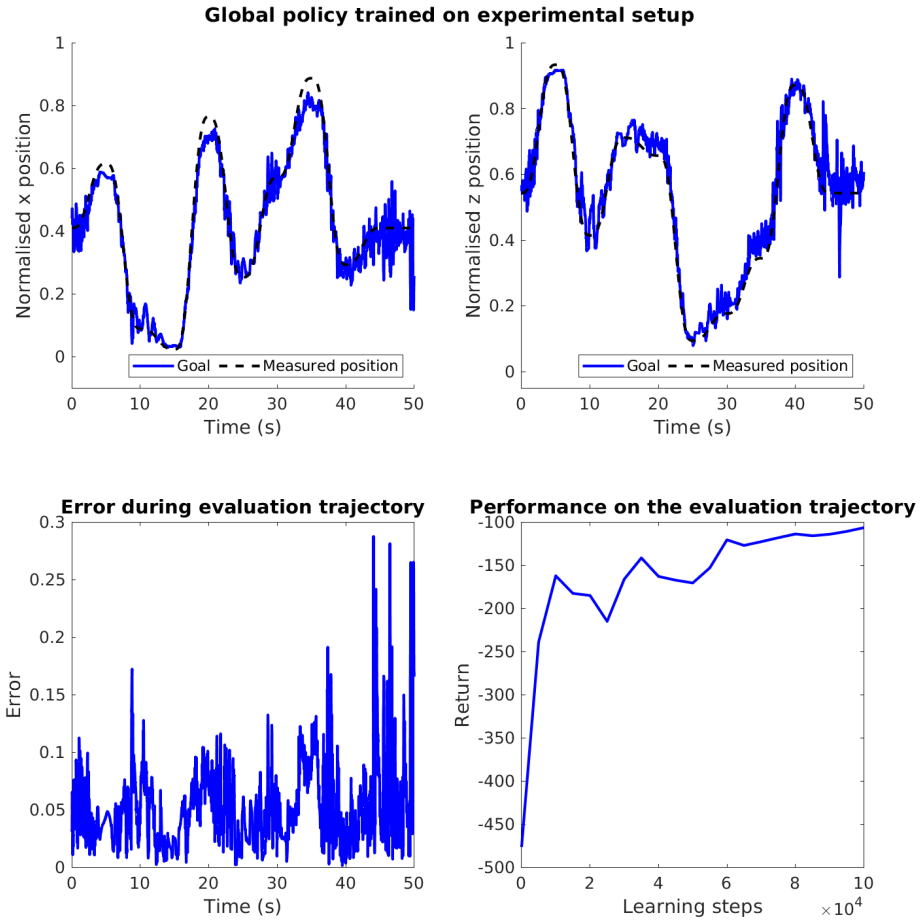



**Figure 4.17:** Results of training a global policy on the experimental setup. Top-left) Normalised x position during evaluation. Top-right) Normalised z position during evaluation. Bottom-left) Error during the evaluation trajectory. Bottom-right) Policy performance during training.

# 5 Discussion

This chapter discusses the results obtained in *Chapter 4 - Results*. This is done by interpreting the achieved results as well as giving a few points of improvement with respect to the acquisition of the shown data.

## 5.1 Interpretation of results

### 5.1.1 Endoscope design

*Section 3.2.1 - System overview* outlined the design and fabrication of the endoscope. The underlying reason for changing the endoscope's design was to decrease the number of training steps needed by reducing the stochasticity of the system. Routing the sensor through the body of the endoscope likely helped in this regard, but the endoscope's design should be improved on even further. As can be seen in figure 3.3, the spacers are not mounted perfectly perpendicular to the endoscope's body, which results in an increased likelihood of the tendon getting stuck on the sharper edges of the holes through which it is routed. This affects the dynamics of the endoscope in an unpredictable manner, which increases the sample complexity of the agent.

As shown in figure 4.1, the system inhibits a memory-like effect, or hysteresis with a long time constant. Holding the endoscope in an upright position changed its mechanical properties, as resetting to a specific tension results in a different resting position after having been in the upright position for a prolonged period of time. The delay lines of the NARX network do not take sufficiently many samples into account to model this behaviour, meaning that this effect remains unmodelled. However, assuming the endoscope has been in a resting position for a sufficiently long time, this should have little effect on the system identification experiments.

A more important consequence of the hysteresis is its effect on the reinforcement learning agent. When developing a local policy, the agent is around its goal position for long enough to give rise to the hysteresis effects. This makes it that the optimal action varies over time, increasing the learning time and potentially causing convergence issues.

### 5.1.2 System identification

#### Constructing sensor data

An error in the endoscope's position is introduced as a result of having to reconstruct invalid sensor readouts. Figure 4.2a shows that this error is mostly within 2 mm when using the method based on the Taylor expansion. However, the error can be seen to increase when more than three readouts are missing consecutively. While adding some noise can help a network generalise, an error of 6 mm in a single direction is likely large enough to deteriorate the generalisation of the network and thus decrease the accuracy of the prediction position.

#### Validation trajectory

The results in figure 4.3 combined with those in figure B.1 show that the maximum prediction error for multi-step prediction error is below 20% of the coordinate's range. The prediction error is largest when changing direction near the edges of the workspace, with the model showing a tendency to overshoot the actual trajectory. The reason for this is likely to be a combination of the quality and the quantity of the data regarding these regions. Due to the input signal being centred around zero input, more data is available around the region close to the resting position than there is around the limits of the workspace. Additionally, figure 4.3 shows that

the sensor fails to output a valid reading more often when close to the edges of the workspace. The combination of having a reduced quality and quantity of data makes training the network harder in these regions, resulting in a larger error. However, because the average error per direction is close to 0 and the error signal is bounded, the model can still be used to train a reinforcement learning agent, although the resulting policies will need to be fine-tuned on the experimental setup.

Figure 4.4 shows the simulated step response compared to the estimated step response. This indicates the damping of the simulated system is higher than that of the physical system. This is in line with the observations made about figure 4.3, where the network's predicted position does not change quickly enough when changing direction. A possible reason for this is that the input signal used for system identification did not excite sufficiently high frequencies, meaning that only slower, low frequency motion could be learned properly.

### 5.1.3 Initial policies

**Expert trajectory in simulation**

Figure 4.5 shows an example of three expert trajectories. This figure shows that the expert reaches the final position within the allotted time and that the different rollouts trace a different path. While the expert reaches a steady-state position, it can be seen that it moves away from this position near the end of the episode. The reason for this is that the solver is not given sufficiently many iterations to fully converge. Therefore, some parts of the trajectory, especially visible at the end, are not optimal. While the number of iterations could be increased to get closer to convergence, this will make it more likely that the optimiser seeks out improperly defined regions of the NARX network and 'abuse' these where possible.

This phenomenon is experimentally demonstrated using figure 4.6 and 4.7. From these figures it can be observed that the spread in final position is larger when running the expert trajectory compared to the trajectory generated by the RL agent, though the former has more accurate tracking in simulation. This indicates that the trajectory found by the reinforcement learning agent is more robust to modelling errors and variations in the starting position. The reason for this is that the objective of the reinforcement learning agent is to optimise a combination of the expected return and expected entropy. This combination results in the agent preferring safer but less rewarding regions of the state-space over regions that are highly rewarding but difficult to traverse. Since the optimiser seeks out the optimal path, it is likely the found path requires a precise set of inputs to go from A to B. While this path is a valid solution according to the NARX network, this trajectory can abuse parts of the NARX that are ill-defined. This makes these trajectories more difficult, or sometimes impossible, to execute on the experimental setup, giving rise to the larger spread in the final position.

**Simulated policies on the setup**

Figure 4.8 gives an indication of the performance of the student policy when deployed on the setup. While this behaviour shows the student policy has some basic understanding of how to move towards the goal position, the overall performance is poor. The reason for this is likely the method with which the policy was trained. During the generation of the expert trajectories an effort was made to enforce spread between these trajectories. Despite this, the expert trajectories do not cover the workspace entirely, meaning that the state-space of the student policy is largely unexplored. This translates to poor robustness against modelling errors as even a small deviation from the predicted path cannot be compensated.

The performance of the simulated RL expert on the experimental setup can be seen in figure 4.9. This agent is more robust to modelling errors in the sense that a sensible action is

available for a larger number of positions. Unfortunately, due to the identified system having a larger damping than the physical system, the learned actions lead to oscillation around the goal position when executed on the experimental setup.

### 5.1.4 Reinforcement learning

#### Global policy in simulation

Figure 4.11 and figure 4.12 show the ability of the simulated agent to follow the evaluation trajectory in simulation. In the case of figure 4.11 the reference can be seen to be tracked well, although there are some places where the agent has a small steady-state error. Overall, the performance on this trajectory is rather good, especially given the intrinsic difficulty that comes from tracing a sequence of steps.

A more practical trajectory is followed in figure 4.12. This figure shows that the x and z-direction can track the reference trajectory well, albeit with small errors and sometimes oscillatory behaviour. It is possible that some of these imperfections could be improved on with further training, but it can also be that this behaviour is caused by the method of interpolation between the points or the NARX network itself. The endoscope can be seen to deviate from the reference trajectory in the y-direction around 10 and 20 seconds. This effect seems position dependent and can be observed with other reference trajectories that pass through the same points. One explanation for this behaviour is that the endoscope is physically unable to follow the reference trajectory. To prevent this from happening, the interpolation method used to generate the evaluation trajectory can be changed to generate trajectories that are part of the endoscope's workspace. Alternatively, the reference trajectory could be made to only exist out of the x and z-direction. If similar effects are still observed after changing the evaluation trajectory, the reason for the behaviour is likely to be as a result of the NARX network being ill-defined at the edges of the endoscope's workspace.

#### Training with different methods of initialisation

Figure 4.13 shows the performance throughout a training session when initialising the policy in three different ways. This figure shows that initialisation using the simulated RL expert increases the sample complexity, whereas initialising the agent with the student policy has little effect. In the early stages of training the agent warm-started using the student policy does seem to improve quicker than the agent trained from scratch. While this could suggest that the method of initialisation indeed reduces the sample complexity, this effect is more likely to be attributed to the fact that the initial evaluation of the agent initialised with the student policy showed worse performance than the agent trained from scratch. After 10.000 steps both agents reach their asymptotic performance and perform similarly. This is in line with the performance observed for the student policy and the simulated RL expert, as shown in figure 4.6 and 4.7. Due to the insufficiently well explored state-space, initialisation with the student policy has little effect on the training time because little information is transferred using this initialisation. On the other hand, initialisation with the simulated RL expert transfers a well explored state-space, but the transferred information is incorrect.

A rollout of each of the policies is shown in figure 4.14. Here, the performance of the policy based on the simulated RL expert is poorest and shows oscillatory behaviour, in line with earlier observations. While figure 4.13 suggests a similar performance of agent initialised using the student policy and the agent initialised randomly, the shown behaviour is different. The randomly initialised agent shows some ability to reach the target location, but fluctuates heavily. The policy initialised with student policy shows different behaviour where the endoscope's tip barely moves, but has a steady-state position that is unequal to the goal position for the y and z-direction. The difference is that this behaviour is not attributed to the method of initialisa-

tion, but rather to chance. The reward function shown in equation 3.6 does not favour either of these modes of behaviour as it is only concerned with the cumulative error. Therefore, the order of the ability to move to a location and the ability to stand still can be learned at different times. However, both policies should, given sufficient training time and a physically reachable goal location, be able to develop a policy that is able to stand still at the correct location.

**Training without prior knowledge**

Instead of focussing on the different methods of initialisation, the performance of the model-free agent trained without any prior knowledge is evaluated. The goal position used in figure 4.14 was generated using the NARX network and can thus be physically unreachable due to the modelling errors. The goal location of the local policy trained without any prior knowledge was selected using the experimental setup, ensuring that it was physically reachable. Figure 4.16 shows the rollout of an agent trained to go to this location. The evolution of this agent's performance over training time is shown in figure 4.15. The performance over training time has a similar trend to that shown in figure 4.10, but can be seen to have a decrease in performance at 64.000 steps. This moment was right after continuing the training session on a different day. The reason for this decrease in performance is because the endoscope was left in its neutral position overnight, 'resetting' the hysteresis effects resulting from trying to reach an upright position during training. The final performance of the agent is reasonably accurate, with a steady state error of less than 5 mm.

Figure 4.17 shows the performance of the agent acting according to a global policy. The agent is able to follow the reference with moderate success, generally following the reference position, but oscillating heavily. This oscillation is undesired and has to be removed before the controller is of use for medical applications. Examining figure 4.17 suggests that the agent was not yet at its asymptotic performance. Due to limited lab availability the agent could not be trained until it reached its final performance. However, while the performance is expected to increase, it is unlikely that the observed oscillations would be removed entirely. Depending on the desired specifications of the final controller, the changes to the training process or way the RL based controller is used have to be made.

## 5.2 Improvements

**Repeating measurements**

Due to limited availability of the experimental setup, many measurements were done only once. However, since every training session in reinforcement learning differs, training the agent once is insufficient to provide conclusive results. This is especially relevant when it comes to showing the effects of initialisation with respect to the learning time, as randomly visiting the correct states early on in the learning process can accelerate the learning significantly. To do a proper analysis of the influence on the learning time, the same training session should be run multiple times to determine with certainty that a method does or does not have an effect on the training time. However, given the decisive degradation in performance due to initialisation with the simulated RL expert and the sparsely explored state-space of the student policy, the outlined methods are unlikely to prove effective.

**Comparing initialisation methods**

The current method of initialisation compares the performance of an agent based on a randomly initialised policy to that of an agent based on a student policy. The problem with this is that some of the difference between the performance of these two is attributed to the random initialisation of each of these policies, as demonstrated by a varying performance when evaluated before training. A better comparison would be to use the same seed to generate the initial

policy as well as the basis for the student policy. This is less of a problem for the agent based on the simulated RL expert as virtually the entire state-space gets changed, meaning that the exact seed used in initialising the random policy only affects the learning time in simulation.

Another issue with the current comparison of initialisation methods is that the amount of episodes between each performance evaluation is rather large. In figure 4.13 it can be seen that a performance close to the asymptotic performance is reached after two evaluations. Decreasing the total number of episodes or evaluating the agent more frequently in the early stages could provide additional insight on the effectiveness of the initialisation, as these are likely to be most prominent early in the training process.

# 6 Conclusion

The aim of this work was to develop an end-to-end process where a system model is created and used to decrease the sample complexity of a model-free agent using soft actor-critic controlling a single segment tendon-driven continuum manipulator. A NARX neural network is used to develop a system model that maps tendon displacement to endoscope position. This is done using less than 20 minutes of measurements on the the experimental setup. The obtained model was shown to give an unbiased estimate of the endoscope's position with a maximum error of 20% in any given direction.

The soft actor-critic algorithm was used to develop a reinforcement learning controller that is able to move the endoscope to a specified location in space by determining the necessary tendon displacements. A local policy is trained in two and a half hour of training time that is able to move and maintain the endoscope's tip position to within 5 mm of the desired location. A global policy trained for the same duration is shown to be able to follow a trajectory through the workspace but oscillates too heavily to be of practical use.

The obtained system model is used to generate expert demonstrations of the endoscope moving through its workspace. These demonstrations are used to train a student policy that is used to warm-start the model-free agent. An agent that developed a global policy in simulation is used for the same purpose. Unfortunately, both these methods showed ineffective in reducing the sample complexity whilst learning a local policy. While the presented method proved ineffective in reducing the sample complexity, this work presents a step towards using reinforcement learning to control a tendon-driven continuum manipulator.

## 6.1 Future work

Further work can be done in two different directions. The first direction is to maintain focus on reducing the sample complexity of the model-free agent. If sample complexity is not a limiting factor, the focus of further work could also be on increasing the performance of the global policy in order to achieve a controller with sufficient performance to control the endoscope in a medical application.

While the achieved results show that the sample complexity is not reduced using the different methods of initialisation, the idea behind these methods is shown to work in literature, as for example in the work of Nagabandi et al. (2017). Therefore, the focus on improving the end-to-end process should be on the system identification step. Further research could be done for a more suitable input signal. The most promising avenue for this work would be to ensure a larger range of frequencies is excited during the identification experiments. The frequency with which the square wave changes could be changed, or a different type of input signal that sweeps over a range of frequencies as suggested by Liu and Song (2015) could be used. Alternatively, different methods of obtaining the system model, such as using Cosserat rod theory, could be used. While these models are too slow to be used for real-time control, they can be used to generate expert trajectories for the purpose of behaviour cloning.

The main source of improvement for the agent's performance is likely to come from shaping the reward function. This function currently does not explicitly penalise overshoot or oscillation. This is something that could be incorporated into the reward function by punishing the agent for motion or actuator action. Additionally, if this method were to be extended to control an endoscope with multiple segments, the reward function should be changed accordingly. A change that could be made is giving a penalty for joint motion, which would prevent the first link from moving unnecessarily, which is something that is not restricted by the current reward function. Changing the reward function in this manner should increase the performance of the agent, taking another step towards achieving a practically viable controller.

# Bibliography

Acuna, G., C. Ramirez and M. Curilem (2012), Comparing NARX and NARMAX models using ANN and SVM for cash demand forecasting for ATM, in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1–6, ISBN 978-1-4673-1490-9, doi:10.1109/IJCNN.2012.6252476.
http://ieeexplore.ieee.org/document/6252476/

Bhagat, S., H. Banerjee, Z. T. H. Tse and H. Ren (2019), Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges, **vol. 8**, no.1, pp. 1–36, ISSN 22186581, doi:10.3390/robotics8010004.

Bieze, T. M., F. Largilliere, A. Kruszewski, Z. Zhang, R. Merzouki and C. Duriez (2018), Finite element method-based kinematics and closed-loop control of soft, continuum manipulators, **vol. 5**, no.3, pp. 348–364, ISSN 21695180, doi:10.1089/soro.2017.0079.
https://doi.org/10.1089/soro.2017.0079

Bishop, C. M. (2006), *Pattern Recogniton and Machine Learning*, Springer-Verlag New York, ISBN 978-0-387-31073-2.

Boussaada, Z., O. Curea, A. Remaci, H. Camblong and N. M. Bellaaj (2018), A nonlinear autoregressive exogenous (NARX) neural network model for the prediction of the daily direct solar radiation, **vol. 11**, no.3, ISSN 19961073, doi:10.3390/en11030620.

Chatterjee, S., R. Niiyama and Y. Kawahara (2018), Design and Development of a Soft Robotic Earthworm with Hydrostatic Skeleton, *2017 IEEE International Conference on Robotics and Biomimetics, ROBIO 2017*, **vol. 2018-Janua**, pp. 1–6, doi:10.1109/ROBIO.2017.8324385.

Chen, J., H. Y. Lau, W. Xu and H. Ren (2016), Towards transferring skills to flexible surgical robots with programming by demonstration and reinforcement learning, *Proceedings of the 8th International Conference on Advanced Computational Intelligence, ICACI 2016*, pp. 378–384, doi:10.1109/ICACI.2016.7449855.

Diaconescu, E. (2008), The use of NARX neural networks to predict chaotic time series, **vol. 3**, no.3, pp. 182–191, ISSN 19918755.

Falkenhahn, V., A. Hildebrandt, R. Neumann and O. Sawodny (2017), Dynamic Control of the Bionic Handling Assistant, **vol. 22**, no.1, pp. 6–17, ISSN 10834435, doi:10.1109/TMECH.2016.2605820.

Gifari, M. W., H. Naghibi, S. Stramigioli and M. Abayazid (2019), A review on recent advances in soft surgical robots for endoscopic applications, **vol. 15**, no.5, ISSN 1478-5951, doi:10.1002/rcs.2010.
https://onlinelibrary.wiley.com/doi/abs/10.1002/rcs.2010

Gläscher, J., N. Daw, P. Dayan and J. P. O'Doherty (2010), States versus Rewards: Dissociable Neural Prediction Error Signals Underlying Model-Based and Model-Free Reinforcement Learning, **vol. 66**, no.4, pp. 585 – 595, ISSN 0896-6273, doi:10.1016/j.neuron.2010.04.016.
http://www.sciencedirect.com/science/article/pii/S0896627310002874

Gu, S., E. Holly, T. P. Lillicrap and S. Levine (2016), Deep Reinforcement Learning for Robotic Manipulation, *CoRR*, **vol. abs/1610.0**.
http://arxiv.org/abs/1610.00633

Haarnoja, T., S. Ha, A. Zhou, J. Tan, G. Tucker and S. Levine (2018a), Learning to Walk via Deep Reinforcement Learning.
http://arxiv.org/abs/1812.11103

Haarnoja, T., A. Zhou, P. Abbeel and S. Levine (2018b), Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, *CoRR*, **vol. abs/1801.0**.
http://arxiv.org/abs/1801.01290

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel and S. Levine (2018c), Soft Actor-Critic Algorithms and Applications, *CoRR*, **vol. abs/1812.0**.
http://arxiv.org/abs/1812.05905

Hill, A., A. Raffin, M. Ernestus, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor and Y. Wu (2018), Github - Stable Baselines.
https://github.com/hill-a/stable-baselines

Ho, J., K.-H. Lee, W. Tang, K.-M. Hui, K. Althoefer, J. Lam and K.-W. Kwok (2018), Localized online learning-based control of a soft redundant manipulator under variable loading, *Advanced Robotics*, **vol. 32**, pp. 1–16, doi:10.1080/01691864.2018.1528178.

Hoitzing, W. B. W. (2020), *Deformation Prediction in Single-Segment Tendon- Driven Flexible Endoscope for Compensation of Gravity and External Loads Using Cosserat Rod Theory (MSc thesis)*.

Kingma, D. P. and M. Welling (2014), Auto-encoding variational bayes, in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR.

Kobayashi, K. (2005), JFS Biomimicry Interview Series: No.6 "Shinkansen Technology Learned from an Owl?" - The story of Eiji Nakatsu.
https://www.japanfs.org/en/news/archives/news{_}id027795.html

Kumar, D. A. and S. Murugan (2018), Performance analysis of NARX neural network backpropagation algorithm by various training functions for time series data, **vol. 3**, no.4, p. 308, ISSN 2053-0811, doi:10.1504/ijds.2018.096265.

Liu, H. and X. Song (2015), Nonlinear system identification based on NARX network, in *2015 10th Asian Control Conference: Emerging Control Techniques for a Sustainable World, ASCC 2015*, Institute of Electrical and Electronics Engineers Inc., ISBN 9781479978625, doi:10.1109/ASCC.2015.7244449.

MathWorks (2020), Find minimum of constrained nonlinear multivariable function - MATLAB fmincon.
https://www.mathworks.com/help/optim/ug/fmincon.html

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller (2013), Playing Atari with Deep Reinforcement Learning.
http://arxiv.org/abs/1312.5602

Morozs, N. (2019), *Accelerating Reinforcement Learning for Dynamic Spectrum Access in Cognitive Wireless Networks (PhD thesis)*.
https://www.researchgate.net/publication/293815876{_}Accelerating{_}Reinforcement{_}Learning{_}for{_}Dynamic{_}Spectrum{_}Access{_}in{_}Cognitive{_}Wireless{_}Networks

Nagabandi, A., G. Kahn, R. S. Fearing and S. Levine (2017), Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning, *CoRR*, **vol. abs/1708.0**.
http://arxiv.org/abs/1708.02596

OpenAI (2018), Spinning up in deep RL.
https://spinningup.openai.com/en/latest/

Pajarinen, J., G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, T. Osa, J. Pajarinen, G. Neumann, J. Andrew Bagnell, P. Abbeel and J. Peters (2018), An Algorithmic Perspective on Imitation

Learning, **vol. 7**, no.2, pp. 1–179, doi:10.1561/2300000053.

Qi, P., C. Liu, A. Ataka, H. K. Lam and K. Althoefer (2016), Kinematic Control of Continuum Manipulators Using a Fuzzy-Model-Based Approach, **vol. 63**, no.8, pp. 5022–5035, ISSN 02780046, doi:10.1109/TIE.2016.2554078.

Rannen, S., C. Ghorbel and N. B. Braiek (2016), , no.2, pp. 2–6.

Renda, F., F. Boyer, J. M. M. Dias and L. D. Seneviratne (2017), Discrete Cosserat Approach for Multi-Section Soft Robots Dynamics, *CoRR*, **vol. abs/1702.0**. http://arxiv.org/abs/1702.03660

Rolf, M. and J. J. Steil (2014), Efficient exploratory learning of inverse kinematics on a bionic elephant trunk, **vol. 25**, no.6, pp. 1147–1160, ISSN 21622388, doi:10.1109/TNNLS.2013.2287890.

Satheeshbabu, S., N. K. Uppalapati, G. Chowdhary and G. Krishnan (2019), Open loop position control of soft continuum arm using deep reinforcement learning, *Proceedings - IEEE International Conference on Robotics and Automation*, **vol. 2019-May**, pp. 5133–5139, ISSN 10504729, doi:10.1109/ICRA.2019.8793653.

Stephens, T. (2007), How a Swiss invention hooked the world. http://www.swissinfo.ch/eng/velcro{_}how-a-swiss-invention-hooked-the-world/5653568

Sutton, R. S. and A. G. Barto (2018), *Reinforcement learning : an introduction*, ISBN 9780262039246.

Thuruthel, T. G., E. Falotico, F. Renda and C. Laschi (2019), Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators, **vol. 35**, no.1, pp. 124–134, ISSN 1552-3098, doi:10.1109/TRO.2018.2878318. https://ieeexplore.ieee.org/document/8531756/

Wang, H., R. Zhang, W. Chen, X. Wang and R. Pfeifer (2017), A cable-driven soft robot surgical system for cardiothoracic endoscopic surgery: preclinical tests in animals, **vol. 31**, no.8, pp. 3152–3158, ISSN 14322218, doi:10.1007/s00464-016-5340-9.

Webster, R. J. and B. A. Jones (2010), Design and kinematic modeling of constant curvature continuum robots: A review, **vol. 29**, no.13, pp. 1661–1683, ISSN 02783649, doi:10.1177/0278364910368147.

Yip, M. C. and D. B. Camarillo (2014), Model-Less Feedback Control of Continuum Manipulators in Constrained Environments, **vol. 30**, no.4, pp. 880–889, ISSN 1552-3098, doi:10.1109/TRO.2014.2309194. http://ieeexplore.ieee.org/document/6776556/

# A Reconstructing sensor data

In *Section 3.3.2 - Sensor data* the algorithm for reconstructing a single missing data point is shown. This principle can be extended to correct for up to three consecutively failed measurements. Assuming a set of successful position readouts $X = \{\mathbf{x}[k-2], \mathbf{x}[k-1], \mathbf{x}[k+3]\}$, the following set of equations can be written:

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mathbf{x}[k] + \mathbf{x}[k-1] + \frac{1}{2}\big(\mathbf{x}[k] - 2\mathbf{x}[k-1] + \mathbf{x}[k-2]\big) \tag{A.1}$$

$$\mathbf{x}[k+2] = \mathbf{x}[k+1] + \mathbf{x}[k+1] + \mathbf{x}[k] + \frac{1}{2}\big(\mathbf{x}[k+1] - 2\mathbf{x}[k] + \mathbf{x}[k-1]\big) \tag{A.2}$$

$$\mathbf{x}[k+3] = \mathbf{x}[k+2] + \mathbf{x}[k+2] + \mathbf{x}[k+1] + \frac{1}{2}\big(\mathbf{x}[k+2] - 2\mathbf{x}[k+1] + \mathbf{x}[k]\big) \tag{A.3}$$

This can be solved for the missing data $\bar{X} = \{\mathbf{x}[k], \mathbf{x}[k+1], \mathbf{x}[k+2]\}$ as:

$$\mathbf{x}[k] = -\frac{17}{49}\mathbf{x}[k-2] + \frac{58}{49}\mathbf{x}[k-1] + \frac{8}{49}\mathbf{x}[k+3] \tag{A.4}$$

$$\mathbf{x}[k+1] = -\frac{18}{49}\mathbf{x}[k-2] + \frac{47}{49}\mathbf{x}[k-1] + \frac{20}{49}\mathbf{x}[k+3] \tag{A.5}$$

$$\mathbf{x}[k+2] = -\frac{11}{49}\mathbf{x}[k-2] + \frac{26}{49}\mathbf{x}[k-1] + \frac{34}{49}\mathbf{x}[k+3] \tag{A.6}$$

Similar steps can be followed to arrive at a set of equations in case there are three correct readings $X = \{\mathbf{x}[k-2], \mathbf{x}[k-1], \mathbf{x}[k+2]\}$ and the two missing ones $\bar{X} = \{\mathbf{x}[k], \mathbf{x}[k+1]\}$ are to be estimated:

$$\mathbf{x}[k] = -\frac{5}{17}\mathbf{x}[k-2] + \frac{18}{17}\mathbf{x}[k-1] + \frac{4}{17}\mathbf{x}[k+2] \tag{A.7}$$

$$\mathbf{x}[k+1] = -\frac{4}{17}\mathbf{x}[k-2] + \frac{11}{17}\mathbf{x}[k-1] + \frac{10}{17}\mathbf{x}[k+2] \tag{A.8}$$

$$\tag{A.9}$$

# B NARX validation

Figure B.1 shows multi-step prediction error achieved using the NARX network when predicting the endoscope's position through the complete validation trajectory.

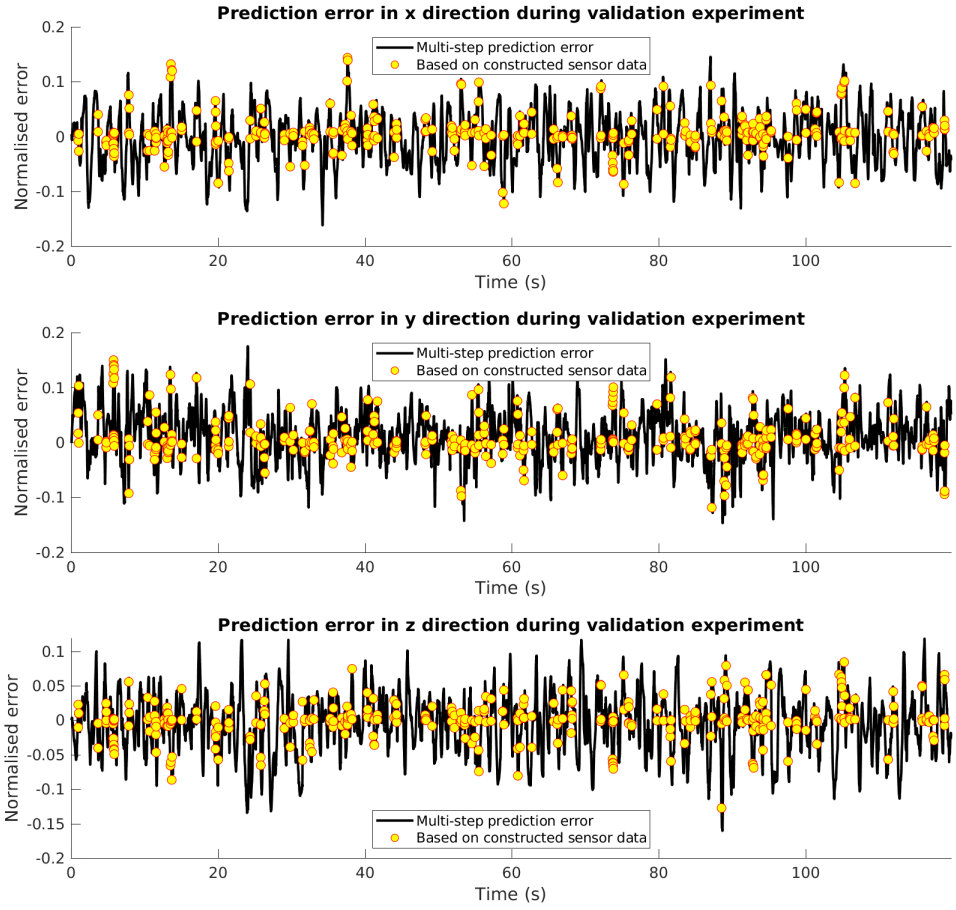



**Figure B.1:** The multi-step prediction errors in each direction for the full validation run.