Eötvös Loránd University

Faculty of Science

Institute of Mathematics

# Neural network-based control of a non-linear dynamical system

*Supervisor:*
Dr. Ferenc Izsák
Habil. Associate Professor

*Author:*
Károly Csurilla
Postgraduate student

*Budapest, 2023*

# Abstract

This thesis aims to be an entry point to reinforcement learning control. We are applying the DDPG (Deep Deterministic Policy Gradient) algorithm to the cart pole system, which is a classical non-linear dynamical system often used for benchmarking of control algorithms. The resulting Matlab framework contains the end-to-end system specification from the derivation of its differential equations (as a proxy for a "real" system) to the real-time interactive visualisation of its trajectories. To improve training efficiency, special attention was paid to the trajectory generation and reward structure.

# Contents

# Chapter 1

# Introduction

Control engineering is an engineering discipline that focuses on actuation methods to make a dynamical system converge to a desired state or approximate a prescribed trajectory. Also, a basic requirement for this procedure is the robustness against environmental or modelling uncertainties. Since its formalisation in the late 19th century, control theory turned out to be a melting pot of real and complex analysis, linear algebra and probability theory.

With the resurgence of neural networks in the 1990s, it was natural to use them as a substitute for control principles designed with classical methods. However, embedding them in a learning scheme was prone to the vanishing gradient problem, similar to recurrent neural networks. It took recent breakthroughs in reinforcement learning to provide a stable training environment, which turned out to be especially effective in tackling high output-dimensional, non-linear problems, where human intuition was infeasible.

Chapter 2 aims to give a concise but self-contained introduction to reinforcement learning, covering the basic results necessary for understanding the Deep Deterministic Policy Gradient (DDPG) algorithm, one of the stepping stones in continuous control. It is followed by the environment description in Chapter 3, where the details of the environment, trajectory and reward equations are discussed. Finally in Chapter 4, we collect the training and controller performance results, with a few concluding remarks in Chapter 5.

# Chapter 2

# Reinforcement learning foundations

Reinforcement learning can be recognised as a learning framework, where an agent is conditioned to continuously improve by sequentially interacting with a stochastic environment [1]. As an area of machine learning, it focuses on exploring and exploiting (maximizing) the reward mechanism of the environment, instead of mimicking observation-action relations coming from a supposed optimal policy (as is the case in supervised learning).

Reinforcement learning is such a vast area that it cannot be reasonably covered within the frames of this work (Figure 2.1). Instead, we are focusing on the derivation of the DDPG algorithm, the central method that we are applying to the cart-pole problem. As these reinforcement learning algorithms have numerous hyperparameters, it is crucial to understand the theory behind them.

DDPG is a model-free, online, off-policy actor-critic reinforcement learning

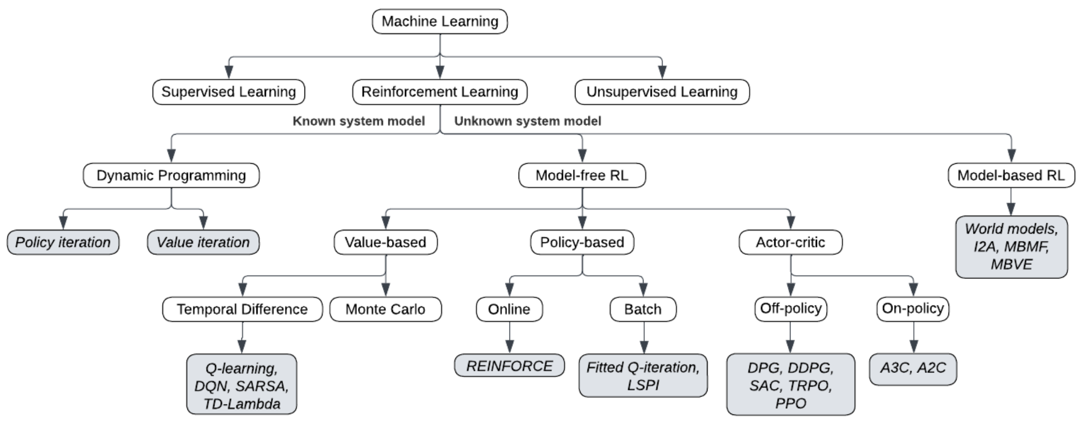


Figure 2.1: Reinforcement learning methodology (based on [2]).

method with deterministic and continuous action space [3]:

- **Policy-based**: Actions are not derived from the estimated cumulative reward function, but have their own approximator that we optimize for.
- **Actor-critic**: A DDPG agent consists of two function approximators, an actor interacting with the environment, and a critic trying to estimate the potential cumulative reward the actor can achieve from a given state.
- **Model-free**: The agent does not learn the environment action-observation relations directly, even though they are captured in the estimated reward function.
- **Online**: The agent parameters are updated each time a new measurement arrives, it does not need to wait until the end of an episode (simulation cycle).
- **Off-policy**: To boost action space exploration during training, the environment is excited with actions other than what the current policy would allow, usually perturbed by a noise model.
- **Deterministic and continuous action**: The actor is approximating the observation-action relations as a map instead of a probability density function, and is not restricted to a finite number of distinct actions.

After this high-level overview, we introduce the mathematical terms and concepts leading up to the DDPG algorithm in the subsequent sections.

## 2.1 Markov decision process

In a model-free reinforcement learning setting, the environment is modelled as a Markov Decision Process (MDP) with state space $\mathcal{S} \in \mathbb{R}^n$, in which it assumes an initial state with probability density $p\left(s_1\right)$ [1] [4]. The environment can be influenced by the external action $a$ in the action space $\mathcal{A} \in \mathbb{R}^m$, and as a result, it undergoes a state transition according to the stationary transition dynamics distribution with conditional density $p\left(s_{t+1}|s_t, a_t\right)$ satisfying the Markov property

$$p\left(s_{t+1}|s_1, a_1, \ldots, s_t, a_t\right) = p\left(s_{t+1}|s_t, a_t\right) \tag{2.1}$$

for any sequence of state-action pairs $\{s_1, a_1, \ldots, s_t, a_t\}$.

[1] This is the shorthand notation for $p(s_1 = s)$, with the random variable $s$ dropped, which is also reflected in the marginalisation differentials. We will be using this simplification throughout the document to avoid symbol duplication, unless we want to denote a specific substitution value.

The method of defining an action $a_t$ at a certain time instance based on the knowledge of the actual state $s_t$ is called the policy. We restrict our talks on deterministic policies $a_t = \mu_\theta(s_t)$ performed by an agent function approximator with smooth parametrisation $\theta \in \mathbb{R}^d$. As a result of the environment-agent interaction, a state trajectory $\tau_{1:t} = \{s_1, s_2, \ldots, s_t\}$ is generated, whose probability for specific states is given by

$$p(\tau_{1:T}) = p(s_1) \prod_{t=1}^{T-1} p(s_{t+1} | s_t, a_t = \mu_\theta(s_t)) . \tag{2.2}$$

Based on this, we denote the density at state $s_T$ after transitioning for $T-1$ time steps starting from state $s_1$

$$p(s_1 \to s_T, \mu_\theta) = \int_{\mathcal{S}} \ldots \int_{\mathcal{S}} p(\tau_{1:T} | s_1) \, \mathrm{d}s_2 \ldots \mathrm{d}s_{T-1}. \tag{2.3}$$

During training phase, the environment provides feedback to the agent about its performance in form of a deterministic reward sequence

$$r_t = R(s_t, a_t) . \tag{2.4}$$

The agent's goal is to maximise this reward over a certain period. While the transition dynamics in Equation (2.1) are bound by a physical law or modelling considerations, the definition of the reward function is completely in the hands of the designer, and it has profound implications for the stability and effectiveness of the training process. The environment-agent interaction is summarised in Figure 2.2.

## 2.2 Value functions

The individual reward values in Equation (2.4) have too high variance to serve as a basis for agent parameter updates. Moreover, for many problems, it pays off taking suboptimal actions in the short term in exchange for arriving at a more beneficial solution in the long run. To mitigate these issues, we define the cumulative discounted reward on an infinite horizon

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k, \tag{2.5}$$

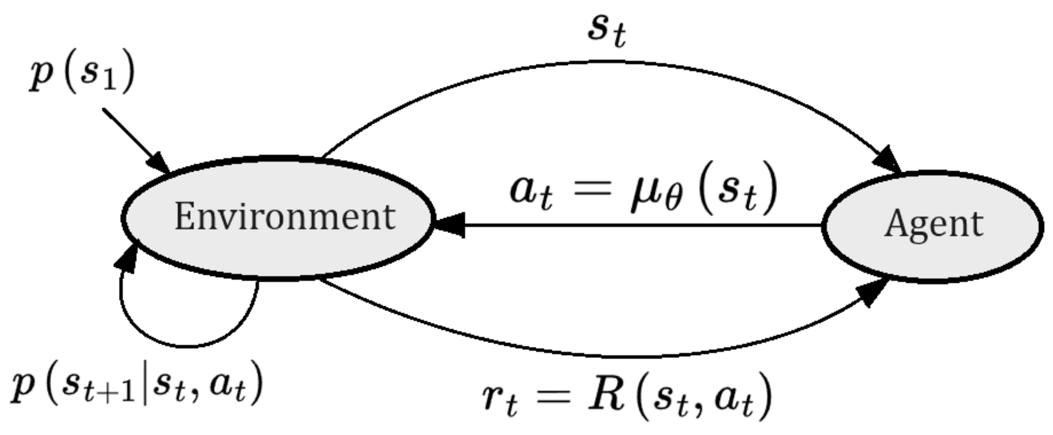


Figure 2.2: Environment-agent signal exchange in a Markov decision process.

with discount factor $\gamma \in [0, 1]$. Having $\gamma < 1$ ensures bounded cumulative reward for infinitely long episodes. $\gamma \ll 1$ puts more weight on short-term rewards and therefore is easier to estimate, but might lead to agent short-sightedness. Agents trained on $\gamma$ values closer to one are more inclined to sacrifice short-term rewards for more favourable long-term behaviour, but could easily lead to training stability problems due to the increased complexity of $G_t$.

Since cumulative rewards can wildly vary for different trajectories, it is customary to take the expected value of it to arrive at a powerful indicator of the agent's quality

$$J(\theta) = \mathbb{E}[G_1|\mu_\theta], \tag{2.6}$$

where the conditionality on $\mu_\theta$ indicates that trajectories are generated using the actual $\theta$ parametrisation of the agent.

The expected cumulative reward - although a useful benchmark - is extremely difficult to estimate and does not really lend itself to agent optimisation in its current form. For this reason, several other value functions have been devised which relax the generality of Equation (2.6). We will start by discussing the state-value function

$$V_\theta(s_1) = \mathbb{E}[G_1|s_1, \mu_\theta], \tag{2.7}$$

which narrows the potential trajectories to those starting from a single starting state. The dependency on $\theta$ is still indicated, albeit as a lower index. By definition,

$$J(\theta) = \mathbb{E}_{p(s_1)}[V_\theta(s_1)] \tag{2.8}$$

holds, where the expected value is performed on the initial state density. Also note that the definition is time shift invariant; in other words, identical irrespective of the starting index of the cumulative reward and initial state.

The state-value function is still closely tied to the current deterministic policy $\mu_\theta$, and does not allow for deviating from it to explore potentially better actions. The action-value function aims to fix this problem by relieving the first (and only the first) action from following the actual policy and replacing it with an arbitrary action:

$$Q_\theta\left(s_1, a_1\right) = \mathbb{E}\left[G_1 | s_1, a_1, \mu_\theta\right]. \tag{2.9}$$

This new value function is more difficult to estimate due to the additional input, but lets us evaluate off-policy actions. Clearly,

$$Q_\theta\left(s_1, \mu_\theta\left(s_1\right)\right) = V_\theta\left(s_1\right). \tag{2.10}$$

Occasionally, there is a third value function created by subtracting the state-value from the action-value function

$$A_\theta\left(s_1, a_1\right) = Q_\theta\left(s_1, a_1\right) - V_\theta\left(s_1\right). \tag{2.11}$$

This is called the advantage function, and shows the benefit of taking a specific first action independent from the baseline policy.

Due to the long time horizon of the underlying cumulative reward in Equation (2.5), we would need to wait out a full episode of environment interaction data to estimate the value functions. To avoid prolonging the training this way, we can use their recursive variant. For the cumulative discounted reward, it is quite straightforward:

$$G_t = r_t + \gamma\, G_{t+1}. \tag{2.12}$$

The recursive formula for the state-value function is as follows:

$$\begin{aligned}
V_\theta(s_t) &= \mathbb{E}\left[G_t|s_t,\mu_\theta\right]\\
&= \mathbb{E}\left[r_t+\gamma G_{t+1}|s_t,\mu_\theta\right]\\
&= R\left(s_t,\mu_\theta\left(s_t\right)\right)+\gamma\int_{\mathcal{S}^\infty} p\left(\tau_{t+1:\infty}|s_t,\mu_\theta\right)G_{t+1}\mathrm{d}\tau_{t+1:\infty}\\
&= R\left(s_t,\mu_\theta\left(s_t\right)\right)\\
&\quad+\gamma\int_{\mathcal{S}} p\left(s_{t+1}|s_t,a_t=\mu_\theta\left(s_t\right)\right)\int_{\mathcal{S}^\infty} p\left(\tau_{t+2:\infty}|s_{t+1},\mu_\theta\right)G_{t+1}\mathrm{d}\tau_{t+2:\infty}\mathrm{d}s_{t+1}\\
&= R\left(s_t,\mu_\theta\left(s_t\right)\right)+\gamma\mathbb{E}_{p(s_{t+1}|s_t,a_t=\mu_\theta(s_t))}\left[\mathbb{E}\left[G_{t+1}|s_{t+1},\mu_\theta\right]\right]\\
&= R\left(s_t,\mu_\theta\left(s_t\right)\right)+\gamma\mathbb{E}_{p(s_{t+1}|s_t,a_t=\mu_\theta(s_t))}\left[V_\theta\left(s_{t+1}\right)\right].
\end{aligned}\tag{2.13}$$

This result is called the Bellman equation for the deterministic action and reward special case. Similarly, for the action-value function:

$$Q_\theta\left(s_t,a_t\right)=R\left(s_t,a_t\right)+\gamma\mathbb{E}_{p(s_{t+1}|s_t,a_t)}\left[Q_\theta\left(s_{t+1},\mu_\theta\left(s_{t+1}\right)\right)\right].\tag{2.14}$$

These recursive formulations have two main advantages. First, we can evaluate the left- and right-hand side for a given transition sample $\{s_t,a_t,r_t,s_{t+1}\}$, and use their difference (temporal difference, Bellman residual) in a function iteration scheme

$$V_\theta\left(s_t\right)\leftarrow V_\theta\left(s_t\right)+\alpha\left(r_t+\gamma V_\theta\left(s_{t+1}\right)-V_\theta\left(s_t\right)\right)\tag{2.15}$$

and

$$Q_\theta\left(s_t,a_t\right)\leftarrow Q_\theta\left(s_t,a_t\right)+\alpha\left(r_t+\gamma Q_\theta\left(s_{t+1},\mu_\theta\left(s_{t+1}\right)\right)-Q_\theta\left(s_t,a_t\right)\right),\tag{2.16}$$

which provably converge under strict conditions [5] to the correct value function of the policy. Equation (2.15) is used in Temporal Difference Learning [6], while Equation (2.16) is employed by the SARSA algorithm [7].

The recursive formulations can also be used for policy generation. Let us denote the optimal policy $\mu_*$, whose value function has the property

$$V_{\mu_*}(s)\geq V_\mu(s)\tag{2.17}$$

for all $s\in\mathcal{S}$ and $\mu$ policy [2]. A key result in reinforcement learning is Bellman's optimality criterion, which is essentially the application of Equations (2.13) and

[2]We are temporarily dropping the policy parameter $\theta$, because these types of policies are typically derived from the value functions themselves.

(2.14) to the optimal policy:

$$V_{\mu_*}\left(s_t\right) = \max_{a_t \in \mathcal{A}} \left[R\left(s_t, a_t\right) + \gamma \mathbb{E}_{p\left(s_{t+1} \mid s_t, a_t\right)}\left[V_{\mu_*}\left(s_{t+1}\right)\right]\right] \tag{2.18}$$

$$Q_{\mu_*}\left(s_t, a_t\right) = R\left(s_t, a_t\right) + \gamma \mathbb{E}_{p\left(s_{t+1} \mid s_t, a_t\right)}\left[\max_{a_{t+1} \in \mathcal{A}} Q_{\mu_*}\left(s_{t+1}, a_{t+1}\right)\right]. \tag{2.19}$$

The first Equation (2.18) is used as a fixed-point iteration (Bellman backup) in the Value Iteration algorithm [8]. The second Equation (2.19) forms the basis of the Q-learning method [9]. Once the optimal value function has been approximated to a sufficient degree, the optimal policy can be obtained by performing some sort of an optimisation or exhaustive search on them:

$$\mu_*\left(s_t\right) = \arg\max_{a_t \in \mathcal{A}} \left[R\left(s_t, a_t\right) + \gamma \mathbb{E}_{p\left(s_{t+1} \mid s_t, a_t\right)}\left[V_{\mu_*}\left(s_{t+1}\right)\right]\right] \tag{2.20}$$

$$\mu_*\left(s_t\right) = \arg\max_{a_t \in \mathcal{A}} Q_{\mu_*}\left(s_t, a_t\right). \tag{2.21}$$

These types of algorithms comprise the value-based reinforcement learning methods (Figure 2.1).

## 2.3 Deterministic Gradient Policy Theorem

The greatest disadvantage of the value-based approaches discussed in the previous section is that they are difficult to apply to continuous action spaces due to the necessary global maximisation at every time step [1] [4]. In case of the so-called policy-based methods, we optimise a separate function estimator to directly yield us the state-action relation.

The function we expect our policy to maximise is the expected cumulative reward in Equation (2.6), so we start by discussing its gradient [4]

$$\begin{aligned} \nabla_\theta J\left(\theta\right) &= \nabla_\theta \mathbb{E}_{p\left(s_1\right)}\left[V_\theta\left(s_1\right)\right] \\ &= \int_{\mathcal{S}} p\left(s_1\right) \nabla_\theta V_\theta\left(s_1\right) \mathrm{d}s_1 \end{aligned} \tag{2.22}$$

Continuing on the gradient of the state-value function (Equation (2.7)):

$$\begin{aligned}
\nabla_\theta V_\theta (s_1) &= \nabla_\theta \left( R(s_1, \mu_\theta(s_1)) + \gamma \int_{\mathcal{S}} p(s_2|s_1, a_1 = \mu_\theta(s_1)) V_\theta(s_2) \mathrm{d}s_2 \right) \\
&= \nabla_\theta \mu_\theta(s_1) \nabla_a R(s_1, a)|_{a=\mu_\theta(s_1)} + \gamma \nabla_\theta \int_{\mathcal{S}} p(s_2|s_1, a_1 = \mu_\theta(s_1)) V_\theta(s_2) \mathrm{d}s_2 \\
&= \nabla_\theta \mu_\theta(s_1) \nabla_a \left( R(s_1, a) + \gamma \int_{\mathcal{S}} p(s_2|s_1, a_1 = a) V_\theta(s_2) \mathrm{d}s_2 \right) |_{a=\mu_\theta(s_1)} \\
&\quad + \gamma \int_{\mathcal{S}} p(s_2|s_1, a_1 = \mu_\theta(s_1)) \nabla_\theta V_\theta(s_2) \mathrm{d}s_2 \\
&= \nabla_\theta \mu_\theta(s_1) \nabla_a Q_\theta(s_1, a)|_{a=\mu_\theta(s_1)} + \gamma \int_{\mathcal{S}} p(s_1 \to s_2, \mu_\theta) \nabla_\theta V_\theta(s_2) \mathrm{d}s_2
\end{aligned} \tag{2.23}$$

Expanding this recursion further yields

$$\begin{aligned}
\nabla_\theta V_\theta(s_1) &= \nabla_\theta \mu_\theta(s_1) \nabla_a Q_\theta(s_1, a)|_{a=\mu_\theta(s_1)} \\
&\quad + \gamma \int_{\mathcal{S}} p(s_1 \to s_2, \mu_\theta) \nabla_\theta \mu_\theta(s_2) \nabla_a Q_\theta(s_2, a)|_{a=\mu_\theta(s_2)} \mathrm{d}s_2 \\
&\quad + \gamma^2 \int_{\mathcal{S}} p(s_1 \to s_3, \mu_\theta) \nabla_\theta V_\theta(s_3) \mathrm{d}s_3 \\
&\ \ \vdots \\
&= \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p(s_1 \to s_t = s, \mu_\theta) \nabla_\theta \mu_\theta(s) \nabla_a Q_\theta(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s,
\end{aligned} \tag{2.24}$$

which then together with Equation (2.22) leads to

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_{\mathcal{S}} p(s_1) \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p(s_1 \to s_t = s, \mu_\theta) \nabla_\theta \mu_\theta(s) \nabla_a Q_\theta(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s \mathrm{d}s_1 \\
&= \int_{\mathcal{S}} \left( \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p(s_1) p(s_1 \to s_t = s, \mu_\theta) \mathrm{d}s_1 \right) \nabla_\theta \mu_\theta(s) \nabla_a Q_\theta(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s \\
&= \int_{\mathcal{S}} \rho_\theta(s) \nabla_\theta \mu_\theta(s) \nabla_a Q_\theta(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s \\
&= \mathbb{E}_{\rho_\theta(s)} \left[ \nabla_\theta \mu_\theta(s) \nabla_a Q_\theta(s, a)|_{a=\mu_\theta(s)} \right],
\end{aligned} \tag{2.25}$$

with $\rho_\theta(s)$ denoting the discounted state distribution. Equation (2.25) is the famous Deterministic Gradient Policy Theorem[3], which allowed for the optimisation of deterministic policies with continuous action spaces.

One surprising fact of Equation (2.25) is that despite the state distribution $\rho_\theta$

[3]To fulfil the conditions of Leibniz's integral rule and Fubini's theorem used during the derivation, we need to impose strict regularity conditions on the individual terms $\rho_\theta$, $\mu_\theta$ and $Q_\theta$.

being dependent on the policy $\mu_\theta$, its gradient does not appear in the final result. This simplifies the gradient of the performance function to the expected value of known Jacobian matrices, which gave rise to the DDPG algorithm.

## 2.4 DDPG algorithm

We are now in a position to outline the Deep Deterministic Policy Gradient algorithm [4] [10] [3]. DDPG is an actor-critic reinforcement learning method, where the Q-function and the policy each have their dedicated neural network function approximator. In a sense, DDPG learning is quite reminiscent of the Generative Adverserial Network (GAN), where two neural networks are competing during the learning process to get better and better in their own respective tasks.

The actor dictates the policy $\mu_\theta$ and is updated according to Equation (2.25). To take advantage of hardware acceleration, we collect enough samples from interactions with the environment and calculate the mean of their gradient evaluations

$$\nabla_\theta J\left(\theta\right) \approx \frac{1}{N}\sum_{k=1}^{N} \nabla_\theta \mu_\theta\left(s_k\right) \nabla_a Q_\theta\left(s_k, a\right)|_{a=\mu_\theta(s_k)}, \tag{2.26}$$

where $N$ is the mini-batch size. Note, how the discounted state distribution $\rho_\theta$ is replaced with the distribution of how we are encountering states.

If we were supplying subsequent states to the policy update in Equation (2.26), the neighbouring correlated samples would distort and destabilise the learning. To ensure sample independence, we employ a replay buffer of length $L$ storing transition tuples $(s_t, a_t, r_t, s_{t+1})$. We first fill up the buffer with an initial interaction sequence, and then during each update, we are sampling it uniformly. Once the buffer is full, the oldest samples are always discarded. Lower buffer length has smaller memory footprint (especially for high-dimensional observation space) and is quicker to converge. A larger buffer ensures higher level of sample efficiency and decorrelatedness, therefore leading to a more economic and stable training.

A critic is appraising the current parametrisation of the policy, and provides estimates for evaluating $\nabla_a Q_\theta\left(s, a\right)$ in Equation (2.26). As in case of the policy $\mu_\theta\left(s\right)$, we are also using a neural network for the Q-function $Q_\omega\left(s, a\right)$, where we are dropping the $\theta$ parameter index in favour of $\omega$, its own direct parametrisation[4]. Equation (2.14) has shown that any true Q-function assessor of the actor must balance both sides of the Bellman equation. We can utilise this fact to update critic parameters $\omega$ by minimising the mean-squared error of the Bellman residuals evaluated on the mini-batch samples:

$$L\left(\omega\right)=\frac{1}{2N}\sum_{k=1}^{N}\left[r_k+Q_\omega\left(s_{k+1},\mu_\theta\left(s_{k+1}\right)\right)-Q_\omega\left(s_k,a_k\right)\right]^2. \tag{2.27}$$

Directly applying Equation (2.27) to neural network-based critics displayed extensive level of instability in practice. It has been demonstrated in [10] that evaluating the right-hand side of Equation (2.14) on a separate set of actor and critic neural networks, whose parameters $\overline{\theta}$ and $\overline{\omega}$ are slowly converging to their counterpart's, has immense stabilising effect. The loss function thus becomes

$$L\left(\omega\right)=\frac{1}{2N}\sum_{k=1}^{N}\left[y_k-Q_\omega\left(s_k,a_k\right)\right]^2, \tag{2.28}$$

where $y_k = r_k + Q_{\overline{\omega}}\left(s_{k+1},\mu_{\overline{\theta}}\left(s_{k+1}\right)\right)$ is the optimisation target evaluated on the target neural networks[5]. The negative loss gradient then takes the form

$$-\nabla_\omega L\left(\omega\right)=\frac{1}{N}\sum_{k=1}^{N}\left(y_k-Q_\omega\left(s_k,a_k\right)\right)\nabla_\omega Q_\omega\left(s_k,a_k\right), \tag{2.29}$$

and can serve as a basis for any available optimisation algorithm (Adam, stochastic gradient descent with momentum, RMSProp [3]). Target network parameters then follow the simple update rule

$$\begin{aligned}\overline{\theta}&\leftarrow\tau\theta+\left(1-\tau\right)\overline{\theta}\\ \overline{\omega}&\leftarrow\tau\omega+\left(1-\tau\right)\overline{\omega},\end{aligned} \tag{2.30}$$

with $\tau$ denoting the smoothing factor.

From a learning stability perspective, it is useful to let the environment emit episode stop signals when the state trajectories diverge to an extent, which is unrecoverable (collision, out of workspace, leaving chaotic attractor/repellor boundaries,

[4]The Q-function is still inherently tied to the policy parametrisation $\theta$, but since this dependency is not highlighted in any of the equations, we are dropping it for convenience.

[5]The dependency of $y_k$ on $\omega$ is typically ignored.

etc.). In these cases, this leads to a slight modification in the calculation of the critic target

$$y_k = r_k, \tag{2.31}$$

to contain the observation and accumulated reward values (neural networks are very sensitive to input scaling). The stop flag is also stored in the replay buffer alongside other transition signals.

Action space exploration is a critical aspect of every reinforcement learning algorithm. The estimated Q-function allows us to learn off-policy, meaning that even if we slightly perturb actions at every simulation step using an arbitrary noise model

$$a_t = \mu_\theta\left(s_t\right) + \mathrm{N}_t, \tag{2.32}$$

the critic will keep being an efficient judge of actor performance. One popular choice is the Ornstein-Uhlenbeck noise model originating from statistical physics [11]. Its discrete time variant is described by the following equations

$$\begin{aligned} n_{t+1} &= n_t + \alpha\left(\overline{n} - n_t\right) T_s + \sigma_t \sqrt{T_s} \mathcal{N}\left(0, 1\right) \\ \sigma_{t+1} &= \max\left(\left(1 - \beta\right)\sigma_t, \sigma_{min}\right), \end{aligned} \tag{2.33}$$

where $\overline{n}$ is the noise process mean the process repeatedly returns to (mean-reverting), $\alpha$ the mean attraction constant, $T_s$ the discrete sample time, $\beta$ the standard deviation decay rate and $\sigma_{min}$ the standard deviation lower bound. It is usually advantageous if the standard deviation is high at the start of the learning and keeps decaying with the number of acquired samples, as the actor and critic are settling on their optimal parameters. With this model, the standard deviation halves after collecting

$$\sigma_{1/2} = \frac{\log 0.5}{\log 1 - \beta} \tag{2.34}$$

number of samples.

Finally, Algorithm 1 contains the summary of the DDPG algorithm [10].

**Algorithm 1** DDPG algorithm

```
Initialise actor $\mu_\theta$ and critic $Q_\omega$ networks
Initialise target actor $\mu_{\overline{\theta}}$ and critic $Q_{\overline{\omega}}$ networks with parameters $\overline{\theta} \leftarrow \theta$ and $\overline{\omega} \leftarrow \omega$
Initialise replay buffer and random noise process N
for each episode do
    Receive initial observation $s_1$
    for each time instance $t$ within episode duration do
        Get action $a_t = \mu_\theta(s_t) + \mathrm{N}_t$ based on current policy and exploration noise
        Execute action $a_t$ and receive new state $s_{t+1}$, reward $r_t$ and stop flag $f_t$
        Store transition $(s_t, a_t, r_t, s_{t+1}, f_t)$ in the replay buffer
        Sample a random minibatch of $N$ samples from the buffer
        for each minibatch sample with index $k$ do
            Set target $y_k = \begin{cases} r_k, \text{if sample } k \text{ is terminating} \\ r_k + Q_{\overline{\omega}}\left(s_{k+1}, \mu_{\overline{\theta}}\left(s_{k+1}\right)\right), \text{otherwise} \end{cases}$
        end for
        Update critic by minimising loss $L(\omega) = \frac{1}{2N}\sum_{k=1}^{N}\left[y_k - Q_\omega(s_k, a_k)\right]^2$
        Update actor using the sampled policy gradient:
                $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{k=1}^{N} \nabla_\theta \mu_\theta(s_k) \nabla_a Q_\omega(s_k, a)|_{a=\mu_\theta(s_k)}$

        Update target networks:        $\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}$
                                       $\overline{\omega} \leftarrow \tau\omega + (1-\tau)\overline{\omega}$
    end for
end for
```

# Chapter 3

# Cart pole environment

After establishing the theoretical setting for reinforcement learning-based controllers in the last chapter, we now turn our attention to applying it to our selected problem. The cart pole system[1] is a popular benchmark problem in control theory due to its relatively low state dimension, chaotic but still tractable dynamics and easy assembly, lending itself to marvellous animations or recordings.

Unlike supervised learning, we saw that in case of reinforcement learning no offline measurement of the system is necessary, the agents are able to learn on the go. This however has severe practical consequences, if we think of how much damage a freshly initialised actor neural network can do while interacting with the physical system (Figure 3.1). Therefore a naive reinforcement learning implementation needs to be augmented with many safety mechanisms to keep the system within operating range.

Another approach is the so-called *Sim-to-Real* knowledge transfer of neural networks [13] [14], where a representative simulation environment is used to cost-effectively precondition agents before they are deployed in reality. Depending on the fidelity of the simulation, the agents can undergo further training to fine-tune them to the peculiarities of the real environment.

In this chapter, we will only be focusing on the simulation environment details, which can later serve as a stepping stone for real-world application.

[1] Also known under names, like inverted pendulum, single pendulum, pendulum on a pulley.

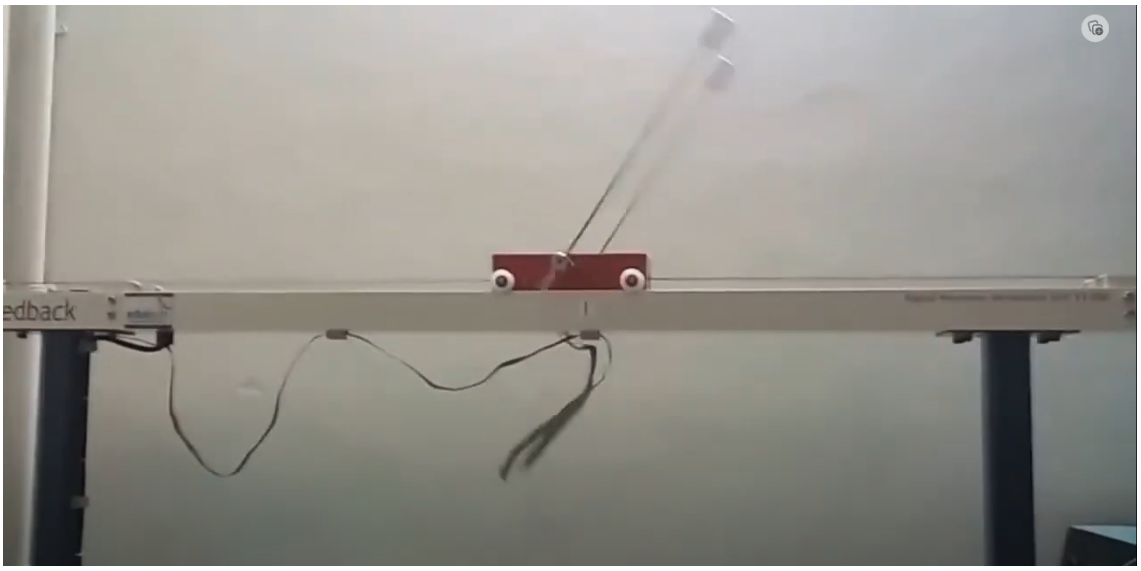


Figure 3.1: Experimental cart pole setup from [12] during a swing-up manoeuvre. The cart is rolling on a horizontal console while being drawn on a belt drive.

## 3.1 Dynamical model

We choose to model the cart pole system through the lens of its simplest mechanical model (Figure 3.2). The cart with mass $M_0$ is constrained to move on a horizontal line and can be affected by a horizontal force $F$. It is connected with a concentrated mass point of mass $M_1$ with a rigid, massless rod of length $L_1$. They are situated in a uniform gravitational field with constant $g$ in the downward direction.

To stabilise the simulation, linear dampings $b_0$ and $b_1$ are introduced at the cart-surface and cart-pole connections, respectively. The actuator has a force limit of $F_{max}$ and a sample time of $T_s$, during which it is holding the actual sampled force constant (zero-order hold discretisation). The model parameters are summarised in Table 3.1.

Table 3.1: Cart pole dynamical system parameters.

| Description | Symbol | Value |
|---|---|---|
| Mass of the cart | $M_0$ | 0.1 kg |
| Concentrated mass of the pole | $M_1$ | 0.1 kg |
| Length of the pole | $L_1$ | 2 m |
| Gravitational constant | $g$ | 9.81 ms$^{-2}$ |
| Cart damping | $b_0$ | 0.1 Nsm$^{-1}$ |
| Pole damping | $b_0$ | 0.01 Nms |
| Actuator force limit | $F_{max}$ | 5 N |
| Actuator sampling time | $T_s$ | 0.05 s |

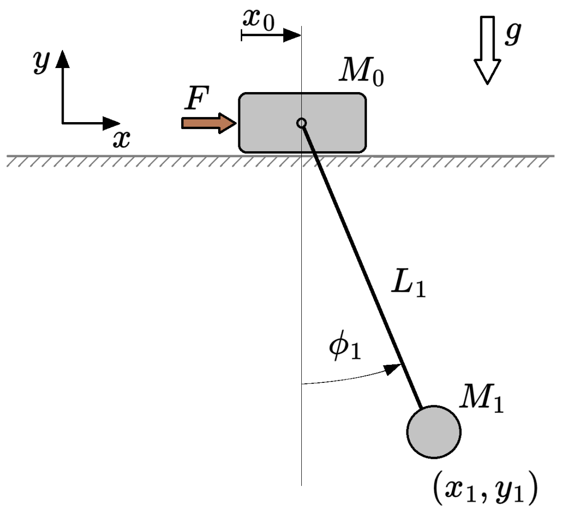


Figure 3.2: Mechanical model of the cart pole system.

The mechanical model is holonomic[2], therefore it is best studied via its Euler-Lagrange equations [15]. Assuming a planar setting, the generalised coordinates are the position of the cart and the swaying angle of the pole compared to the lower stable position

$$q(t) = \begin{bmatrix} x_0(t) \\ \phi_1(t) \end{bmatrix}, \tag{3.1}$$

making our model to have two degrees of freedom. The position vectors of the two mass points are then

$$p_0(t) = \begin{bmatrix} x_0(t) \\ 0 \end{bmatrix} \qquad p_1(t) = \begin{bmatrix} x_0(t) + L_1 \sin(\phi_1(t)) \\ -L_1 \cos(\phi_1(t)) \end{bmatrix}. \tag{3.2}$$

The kinetic energy of the system is

$$T = \frac{1}{2} M_0 |\dot{p}_0|^2 + \frac{1}{2} M_1 |\dot{p}_1|^2, \tag{3.3}$$

while the potential function of the only conservative force is

$$U = -M_1 g L_1 \cos\phi_1. \tag{3.4}$$

Assuming dissipative forces linearly depending on the velocity difference at frictional

[2]Mechanical systems are holonomic, if the constraints concerning the constellation of its mass points are of purely geometrical nature, i.e. depend on their positions only and not their velocities (or higher time-derivatives). In this case, the system has a minimal number of general spatial parameters by which the positions of each of its particle can be expressed. Examples for non-holonomic systems are the rolling coin, Segway hoverboard and ice skating.

surfaces, their dissipative potential can be formulated as

$$D = \frac{1}{2} b_0 \dot{x}_0^2 + \frac{1}{2} b_1 \dot{\phi}_1^2. \tag{3.5}$$

Our only non-conservative force is the actuator effect, whose power is expressed as $P = F\dot{x}_0$, leading to a generalised force vector as follows:

$$Q = \begin{bmatrix} F \\ 0 \end{bmatrix} \tag{3.6}$$

Based on these quantities, the equations of motion are directly obtained from the Euler-Lagrange equations

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial D}{\partial \dot{q}_i} + \frac{\partial U}{\partial q_i} = Q_i \qquad i = 1, 2. \tag{3.7}$$

After solving the resulting system of ordinary differential equations for the second time derivatives, we arrive at the system equations

$$\begin{aligned} \ddot{x}_0 =& \frac{L_1 F + b_1 \dot{\phi}_1 \cos\phi_1 - L_1 b_0 \dot{x}_0 + {L_1}^2 M_1 \sin\phi_1 \dot{\phi}_1^2 + L_1 M_1 g \cos\phi_1 \sin\phi_1}{L_1\left(-M_1\cos^2\phi_1 + M_0 + M_1\right)} \\ \ddot{\phi}_1 =& -\frac{M_0 b_1 \dot{\phi}_1 + M_1 b_1 \dot{\phi}_1 + L_1 {M_1}^2 g \sin\phi_1 + L_1 M_1 F \cos\phi_1}{{L_1}^2 M_1\left(-M_1\cos^2\phi_1 + M_0 + M_1\right)} \\ &- \frac{L_1 M_0 M_1 g \sin\phi_1 - L_1 M_1 b_0 \dot{x}_0 \cos\phi_1 + {L_1}^2 {M_1}^2 \dot{\phi}_1^2 \cos\phi_1 \sin\phi_1}{{L_1}^2 M_1\left(-M_1\cos^2\phi_1 + M_0 + M_1\right)}. \end{aligned} \tag{3.8}$$

The environment implementation divides the time scale into sampling periods, during which the cart force $F$ is held constant, and integrates Equations (3.8) with an explicit, adaptive step-size solver (*ode45*) with the default tolerances. The system is non-linear, a stability analysis would show two fixed points, one stable at $\phi_1 = 0$ and another unstable at $\phi_1 = \pi$. Accordingly, in the following sections, we will be referring to the half-plane below the cart level as stable region, while the other one will be called unstable region.

## 3.2 Trajectory

We control the system states $[x_0, \phi_1]$ indirectly by prescribing a reference trajectory in terms of the endpoint of the pole $[x_1, y_1]$. The reference trajectory can be thought of as the ideal behaviour of the system, which can only be followed

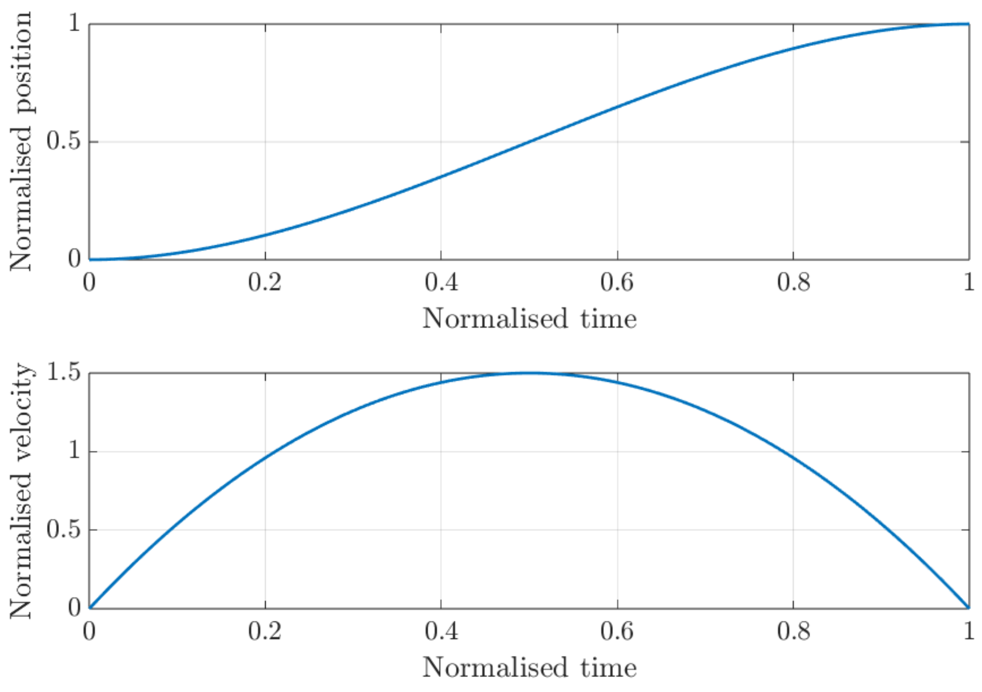


Figure 3.3: Akima spline used for normalised reference trajectory.

approximately with some tracking error due to actuator power and reaction time limitations. We can significantly improve on the tracking error by designing feasible reference signals, in line with the system dynamics.

The most important manoeuvre we expect the system to perform is the transition of the endpoint from standstill position to another position, with no residual swaying at the end, both in the stable and unstable regions. For this reason, we are using Akima spline [16] with the interpolation points

$$\begin{bmatrix}-2\\0\end{bmatrix}, \begin{bmatrix}-1\\0\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}3\\1\end{bmatrix} \tag{3.9}$$

resulting in a smooth zero to one transition depicted in Figure 3.3. This normalised curve can then be scaled and offset to arbitrary starting and ending positions, maximal velocity, as well as manoeuvre durations. These four parameters however cannot be chosen independently, as they are related in the following way

$$v = \frac{m\,\Delta x}{\Delta t}, \tag{3.10}$$

where $m = 1.5$ is the maximum velocity of the normalised transition.

We decided that the most natural parametrisation is to define allowed ranges for

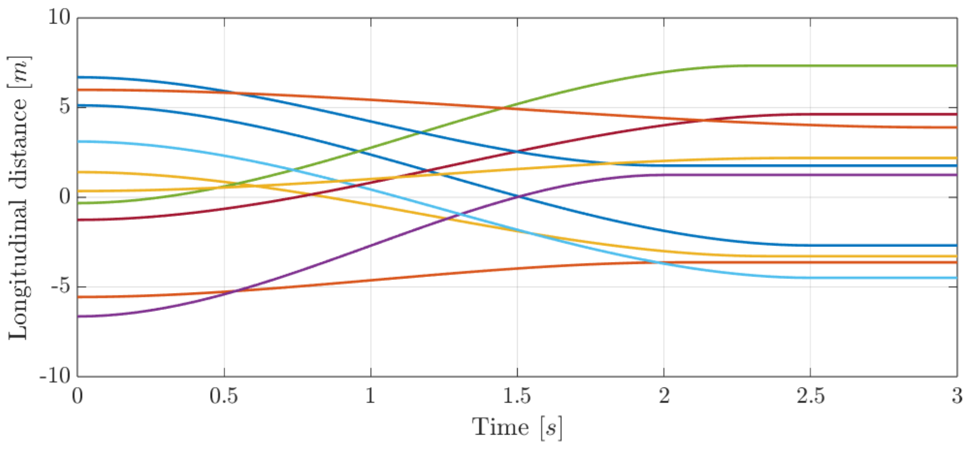


Figure 3.4: Different realisations of reference trajectories for the training of the stable agent. If the transition duration is shorter than the episode length, the destination position is extended.

the transition duration $\Delta t \in [\Delta t_{min}, \Delta t_{max}]$ and maximal velocity $v \in [v_{min}, v_{max}]$, which restricts the transition lengths as a result[3]

$$\Delta x \in \left[ \frac{v_{min}\, \Delta t_{max}}{m}, \frac{v_{max}\, \Delta t_{min}}{m} \right]. \tag{3.11}$$

Randomly selecting transition duration and length from these uniform distributions will ensure that the maximal velocity will always fall within the allowed range. After the normalised trajectory has been scaled up with the selected length and duration values, it is then randomly offset within the middle 80% of the horizontal workspace. Finally, its direction is flipped with 50% probability. Figure 3.4 is showing a few example trajectories.

Another very important manoeuvres are the stable-unstable and unstable-stable transitions. Since the two regions have their own set of actor-critic networks, we don't generate a reference transition[4] but will leave it up to the stable actor-critic to find the optimal swaying elimination strategy, and to the unstable networks to settle on the optimal swing-up path. When we start a new episode, we either generate a horizontal or vertical transition, or a simple standstill as a possible third option with probabilities varying through the learning stages. We also provide many uniform distribution width parameters to select the initial angle around the actual starting

[3]This of course requires that $v_{min}\, \Delta t_{max} \leq v_{max}\, \Delta t_{min}$. We also check for $\Delta x_{max}$ not being greater than the allowed horizontal workspace.

[4]The truth is that this would be a very difficult task, because the ideal trajectory is a factor of the physical and actuator parameters.

position, as it had tremendous effect on the learning process and actor performance.

Table 3.2: Reference trajectory parameters in the stable region.

| Description | Symbol | Value |
|---|---|---|
| Transition duration | $\Delta t$ | [2, 3] s |
| Maximal transition velocity | $v$ | [0.5, 6] $\mathrm{ms}^{-1}$ |
| Starting angle distribution width | $\Delta\phi_1$ | 1 rad |
| Probability of transition manoeuvre | | $0.8\ldots1$ |
| Probability of unstable starting position | | $0\ldots0.2$ |
| Probability of standstill starting position | | $0\ldots0.1$ |

Table 3.3: Reference trajectory parameters in the unstable region.

| Description | Symbol | Value |
|---|---|---|
| Transition duration | $\Delta t$ | [3, 5] s |
| Maximal transition velocity | $v$ | [0.2, 1] $\mathrm{ms}^{-1}$ |
| Unstable starting angle distribution width | $\Delta\phi_1$ | 0.5 rad |
| Stable starting angle distribution width | $\Delta\phi_1$ | $0.5 \to \pi$ rad |
| Probability of transition manoeuvre | | $0 \to 0.2$ |
| Probability of stable starting position | | $0 \to 0.7$ |
| Probability of standstill starting position | | $1 \to 0.1$ |

## 3.3 Environment signals

The choice of the actor-critic input interface has far-reaching consequences. A naive solution would suggest to measure the full state of the system, but we are often limited in terms of physical accessibility or sensor cost considerations. Available sensors therefore often produce a mapping of the state, many times with lower dimensionality than the state itself. The question of whether the original state can be reconstructed in theses cases is studied in control theory, and is stated in various observability conditions. Higher-dimensional observations also see applications in redundant systems, or for example in camera-based control algorithms working from raw video pixel stream data.

We opted for using state-feedback for simplicity but there are still nuances to its implementation. Directly feeding back the cyclical coordinate $\phi_1$ would cause scalability problems for neural networks, as a continuous angle can grow indefinitely. We could theoretically limit angles within $(-\pi, \pi]$ but then the network would have to smoothen out the discontinuities of angle wraps. Typically, the solution to this problem is to split angle information into $\sin\phi_1$ and $\cos\phi_1$. We have opted to a similar route by including direct kinematics information (Equation (3.2)) $x_0$, $x_1$ and $y_1$ in the observations sent to the agent, encoding angle data indirectly. Thus, the observation vector initially encompassed the following signals

$$\left[x_0,\, \dot{x}_0,\, x_1,\, y_1,\, \dot{\phi}_1,\, x_{1,ref},\, \dot{x}_{1,ref}\right], \tag{3.12}$$

with the last two elements explained in the previous section on trajectory generation.

The agents have been trained using Equation (3.12) with relative success, but its problems became quickly apparent. The intuition of a control engineer would immediately tell that the actor network would have to use one dense layer just to produce the error signals $x_{1,ref} - x_1$ and $\dot{x}_{1,ref} - \dot{x}_1$, with $\dot{x}_1$ alone being really challenging to recreate. Moreover, having absolute positions in the observation vector made the agent performance inconsistent across the workspace to the point of showing instabilities towards the edge of the workspace, where training samples were sparse. To rectify these issues, we modified the agent interface with more preprocessed signals, even reducing the dimension by one

$$\left[x_{1,ref} - x_0,\, \dot{x}_{1,ref} - \dot{x}_0,\, x_{1,ref} - x_1,\, \dot{x}_{1,ref} - \dot{x}_1,\, y_1,\, \dot{y}_1\right], \tag{3.13}$$

which showed much better performance. In practice, these signals could be measured by either a camera detecting the objects in every frame, or two angle sensors at the belt drive and pole bearing.

## 3.4 Reward

Similarly to trajectory generation, we are also handling the reward structure of the stable and unstable agents differently. What they had in common was the following function

$$\begin{aligned} r =& R_b - P_{x_1}\left(x_{1,ref} - x_1\right)^2 - P_{y_1}\left(y_{1,ref} - y_1\right)^2 + \\ & - P_{\dot{x}_0}\left(\dot{x}_{1,ref} - \dot{x}_0\right)^2 - P_{\dot{x}_1}\left(\dot{x}_{1,ref} - \dot{x}_1\right)^2 - P_F F^2, \end{aligned} \tag{3.14}$$

where $P_{x_1}$, $P_{y_1}$ are the Euclidean distance-based penalty gains with $y_{1,ref}$ defined as constant $-L$ for stable and $+L$ for unstable agents. The penalty terms $P_{\dot{x}_0}$ and $P_{\dot{x}_1}$ help with trajectory tracking, and together with $P_F$, they act as controller stabilisers, preventing oversensitivity to positional errors.

The penalty terms in Equation (3.14) have been initially selected to normalise the squared errors to fall within roughly the same order of magnitude. Then, they stayed the same for hundreds of episodes, filling up the sample buffer with rewards corresponding to an identical reward function. After each of these longer training cycles, the reward parameters were revised based on a manual inspection of the agent performance. After a long trial and error of hyperparameter tuning, the resulting reward gains are summarised in Table 3.4 and 3.5.

Table 3.4: Final reward parameters in the stable region.

| Description | Symbol | Value |
|---|---|---|
| Baseline reward | $R_b$ | 10 |
| Horizontal position error penalty factor | $P_{x_1}$ | 3 |
| Vertical position error penalty factor | $P_{y_1}$ | 1 |
| Cart velocity error penalty factor | $P_{\dot{x}_0}$ | 0.2 |
| Pole velocity error penalty factor | $P_{\dot{x}_1}$ | 0.2 |
| Actuator force penalty factor | $P_F$ | 0.1 |

Table 3.5: Final reward parameters in the unstable region.

| Description | Symbol | Value |
|---|---|---|
| Baseline reward | $R_b$ | 10 |
| Horizontal position error penalty factor | $P_{x_1}$ | 3 |
| Vertical position error penalty factor | $P_{y_1}$ | 3 |
| Cart velocity error penalty factor | $P_{\dot{x}_0}$ | 1.5 |
| Pole velocity error penalty factor | $P_{\dot{x}_1}$ | 0.6 |
| Actuator force penalty factor | $P_F$ | 0.1 |

The training is stopped and episode stop signal is emitted once the cart has left the workspace (or the horizontal position error grew too much). $R_b$ is the baseline reward and in the earlier iterations it included a large negative value in this case as a form of negative reinforcement. Its problems quickly came to surface when the freshly initialised agent was unable to learn anything, because it was incentivised to end the episode as quickly as possible by crashing the cart into the workspace boundaries in order to keep the accumulated negative reward low (Figure 3.5). The only way to stabilise the training was to apply positive reinforcement at every time instance, when the agent has managed to operate within boundaries. This positive constant $R_b$ started high to offset the large tracking errors initially, then was gradually reduced to make the small differences in the penalty terms more highlighted. Another strong

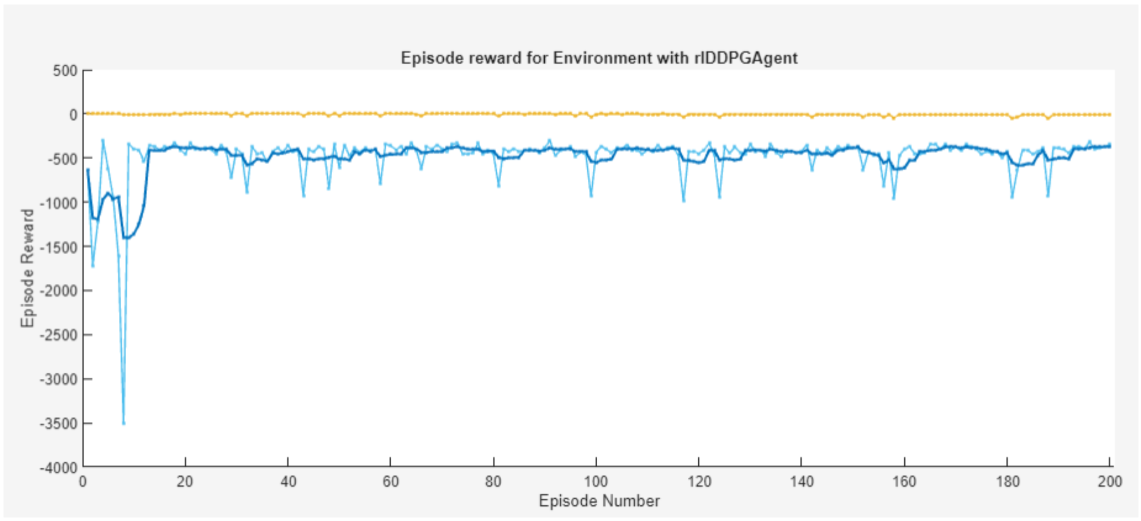


Figure 3.5: Reward hacking phenomenon, where a freshly initialised agent was unable to overcome the local optimum of ending the episode as quick as possible. Plotted with default Matlab *rlTrainingOptions* settings, where the light blue line shows the cumulative discounted reward obtained by the agent during subsequent episodes of the training process.

argument against having large discontinuities in the reward function is that it makes it more difficult for the critic network to estimate the cumulative reward.

Generally, it is arguable, where to end an episode. Wrapping it up too early might stifle exploration, too late however poses numerical problems of intractable cumulative rewards (penalties). In this specific problem, concluding early turned out to be the better approach, especially during the training of the unstable agent. First, it was trained to balance the pole starting close to the upwards position $\phi_1 = \pi$. As soon as it deviated from the unstable fixed point by more than 45°, the episode was finalised as we could no longer realistically expect the agent to recover. This also sped up the training because we could quickly try out a new starting position instead of dragging out a failed episode. A carefully tuned positive reinforcement term $R_b$ was instrumental in guiding the unstable agent to prolong the episode further and further, until it successfully learned to balance the pole (Figure 3.6). To extend this to the swing-up case (where the pole is starting in the stable position), we set a flag for the environment if the pole has entered the unstable range, and only terminate the episode if it is outside again with an active flag.

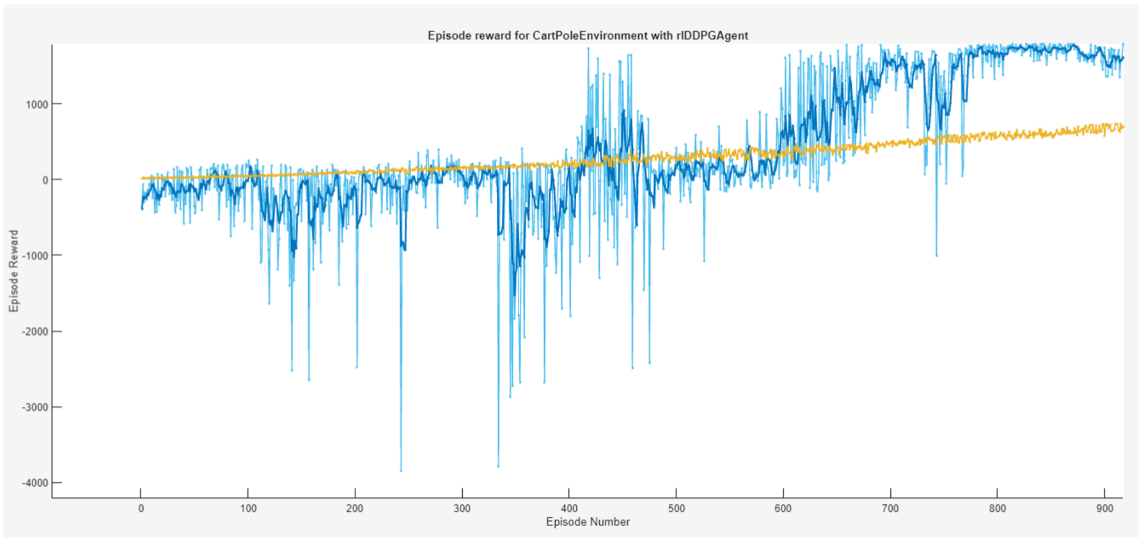


Figure 3.6: After multiple episodes of exploration, the unstable agent is slowly increasing the episode duration and expected reward, until it finally achieves a breakthrough in balancing the pole. Plotted with default Matlab *rlTrainingOptions*, where the light blue line shows the cumulative discounted reward obtained by the agent during subsequent episodes of the training process.

# Chapter 4

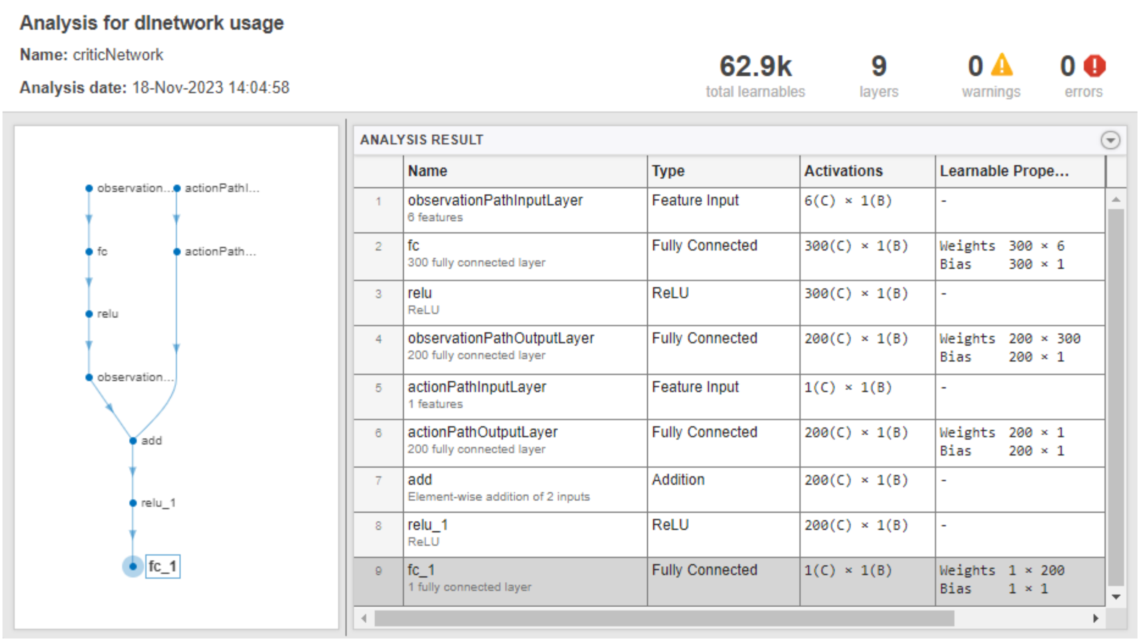
# Agents

After the discussion on the environment, we now turn our attention to the agents. We are trying to support two distinct operating modes, namely controlling the horizontal position of the endpoint with minimal swaying (stable operating range, typical use case in material movement), and pole balancing (unstable operating range, frequently occurring in robotics tasks). We could have integrated these two mutually exclusive use cases into one single agent with an additional switch input. Doing so would have led to training problems however, as the neural network would have been trying to decide which of its connections is responsible for which task. The easier and less computationally intensive solution was to dedicate two separate agents to each operating mode and switch between them with a supervisor logic. For the sake of brevity, we will be referring to them as "stable" and "unstable" agents, indicating their operating range (and not their dynamical properties).

## 4.1 Agent architecture

The construction of the agent networks follows the example set forth in [4]. Both the actor and critic networks entail two hidden layers following the observation input in Equation (3.13). The actor network is closed off with a *tanh* and a scaling layer to make its output adherent to the actuator limitations. The critic network action input is only included in the second layer.

Layer hierarchy is shown in Figure 4.1 and 4.2, with the parametrisation reflecting the final state of the stable agent. Network sizes have been gradually increasing with the maturity of the environment, starting from a few dozens of neurons each layer,



**Analysis for dlnetwork usage**

**Name:** criticNetwork

**Analysis date:** 18-Nov-2023 14:04:58

62.9k total learnables | 9 layers | 0 warnings | 0 errors

ANALYSIS RESULT

| | Name | Type | Activations | Learnable Prope... |
|---|---|---|---|---|
| 1 | observationPathInputLayer<br>6 features | Feature Input | 6(C) × 1(B) | - |
| 2 | fc<br>300 fully connected layer | Fully Connected | 300(C) × 1(B) | Weights 300 × 6<br>Bias 300 × 1 |
| 3 | relu<br>ReLU | ReLU | 300(C) × 1(B) | - |
| 4 | observationPathOutputLayer<br>200 fully connected layer | Fully Connected | 200(C) × 1(B) | Weights 200 × 300<br>Bias 200 × 1 |
| 5 | actionPathInputLayer<br>1 features | Feature Input | 1(C) × 1(B) | - |
| 6 | actionPathOutputLayer<br>200 fully connected layer | Fully Connected | 200(C) × 1(B) | Weights 200 × 1<br>Bias 200 × 1 |
| 7 | add<br>Element-wise addition of 2 inputs | Addition | 200(C) × 1(B) | - |
| 8 | relu_1<br>ReLU | ReLU | 200(C) × 1(B) | - |
| 9 | fc_1<br>1 fully connected layer | Fully Connected | 1(C) × 1(B) | Weights 1 × 200<br>Bias 1 × 1 |

Figure 4.1: Critic network structure of the final stable agent. The bias parameters in the action fully connected layer are disabled. Created by *analyzeNetwork* Matlab function.

until the training has stabilised and the performance was deemed satisfactory. Table 4.1 contains the final layer sizes and parameter counts.

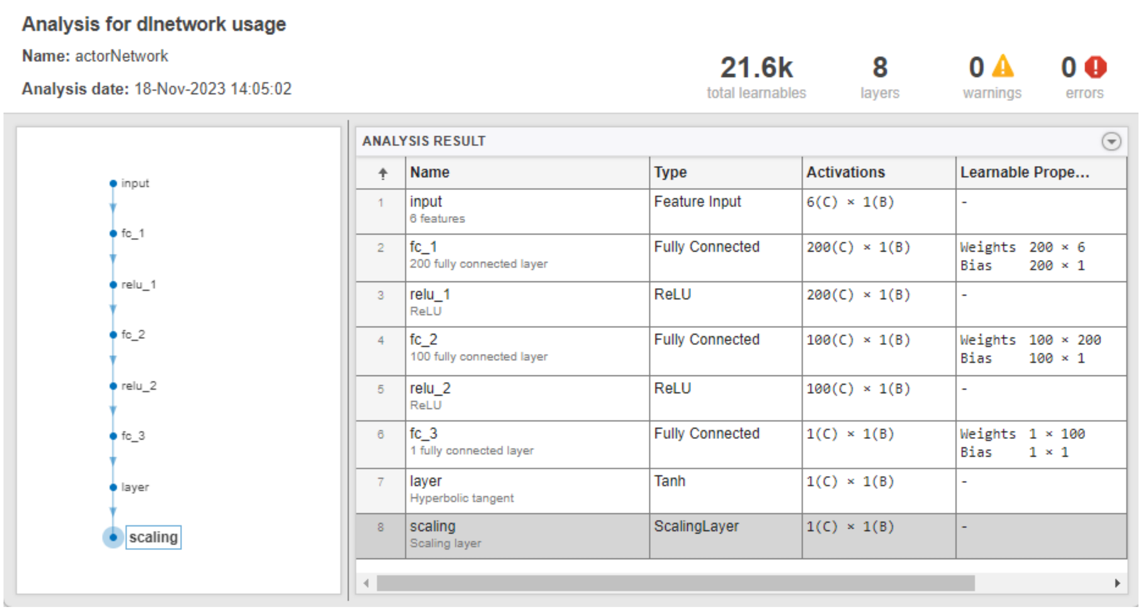
Table 4.1: Final agent network sizes.

| | Hidden layer sizes | Total learnables |
|---|---|---|
| Stable critic | $[300, 200]$ | $\sim 62.9k$ |
| Stable actor | $[200, 100]$ | $\sim 21.6k$ |
| Unstable critic | $[350, 200]$ | $\sim 73.2k$ |
| Unstable actor | $[250, 200]$ | $\sim 52.1k$ |

## 4.2 Training process

Some training details have already been disclosed in Section 3.4, we would like to extend it here. Unless we state a number in parenthesis, it is valid for both the stable and unstable agents, otherwise the number in parenthesis is valid for the unstable agent only.

After a network has been initialised, its experience buffer gets filled at the start of the first training process. The length of the experience buffer was set to 200000 samples, which means that episodes lasting up to 250 (300) samples, we can expect the data in the experience buffer to be fully replaced within 800 (667) full episodes.



**Analysis for dlnetwork usage**

**Name:** actorNetwork

**Analysis date:** 18-Nov-2023 14:05:02

**21.6k** total learnables

**8** layers

**0** warnings

**0** errors

ANALYSIS RESULT

| | Name | Type | Activations | Learnable Prope... |
|---|---|---|---|---|
| 1 | input<br>6 features | Feature Input | 6(C) × 1(B) | - |
| 2 | fc_1<br>200 fully connected layer | Fully Connected | 200(C) × 1(B) | Weights 200 × 6<br>Bias 200 × 1 |
| 3 | relu_1<br>ReLU | ReLU | 200(C) × 1(B) | - |
| 4 | fc_2<br>100 fully connected layer | Fully Connected | 100(C) × 1(B) | Weights 100 × 200<br>Bias 100 × 1 |
| 5 | relu_2<br>ReLU | ReLU | 100(C) × 1(B) | - |
| 6 | fc_3<br>1 fully connected layer | Fully Connected | 1(C) × 1(B) | Weights 1 × 100<br>Bias 1 × 1 |
| 7 | layer<br>Hyperbolic tangent | Tanh | 1(C) × 1(B) | - |
| 8 | scaling<br>Scaling layer | ScalingLayer | 1(C) × 1(B) | - |

Figure 4.2: Actor network structure of the final stable agent. Created by *analyzeNetwork* Matlab function.

These relatively high buffer sizes provided enough decorrelation for a stable learning. We could theoretically use this experience buffer to normalise the observations with parametrising *featureInputLayer*. However, all of the observation variables seemed to fall within the same order of magnitude after an initial check, so we refrained from implementing an outer normalisation loop from training to training[1].

The discount factor was 0.99 (0.993), which means that the "halving time" of future rewards was 69 (99) samples, or 3.45 (4.93) seconds. Having discount factors this high was key in ensuring agent far-sightedness, for example during swing-up manoeuvre or reducing the static error. Mini-batch size has been chosen at 256 and stayed the same throughout the training. Optimisation algorithm was Adam with default parametrisation, except for the gradient threshold set to 1 and the actor-critic learn-rate, which started in the $10^{-3}$ range and was gradually reduced to as low as $10^{-4}$ with agent maturity. Target network smoothing factor $\tau$ was set to 0.001.

Two different noise models (Equation (2.32)) have been used throughout the training. Gaussian random process helped us explore the stable agent action space. The unstable agent balancing and swing-up noise model initially followed the Ornstein-Uhlenbeck process presented in Equation (2.33). Its correlated nature however was very prone to leading to high static horizontal errors, so a final fine-tuning

[1] As the possible observation ranges could very much depend on agent performance and reference trajectory proportions

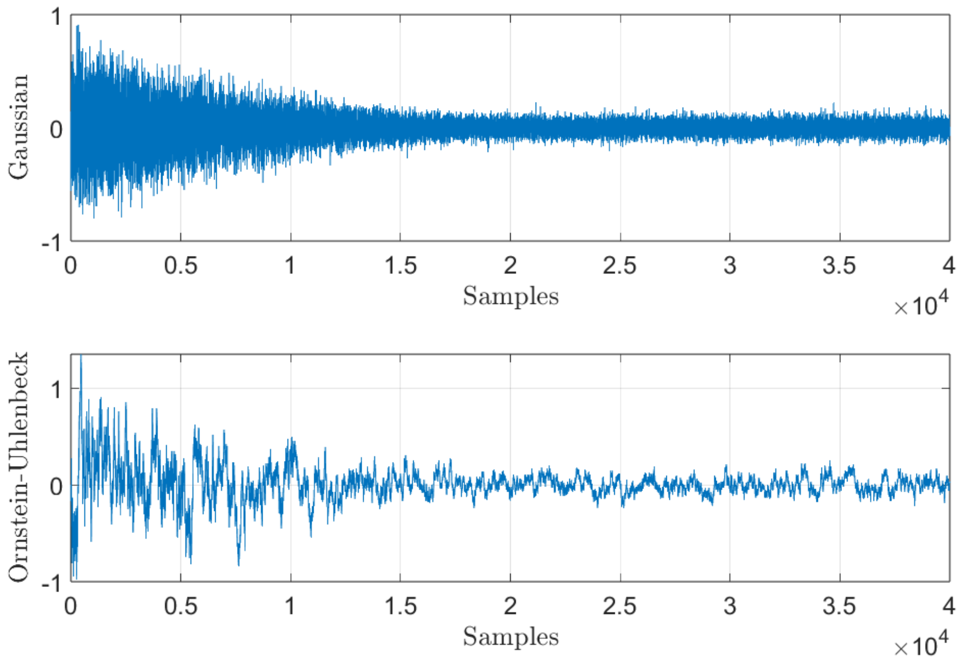


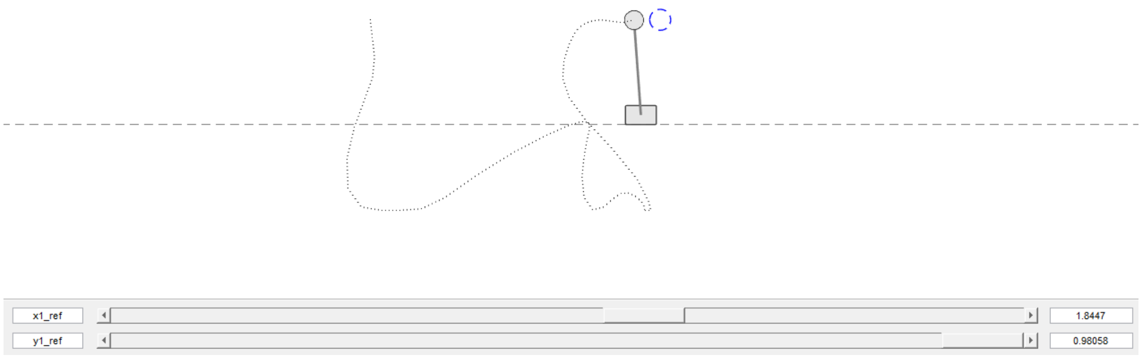
Figure 4.3: Comparison of Gaussian and Ornstein-Uhlenbeck noise processes applied to the actuator force at some point throughout the training, in addition to the agent output. They have identical parameters, with mean $\bar{n} = 0$, starting standard deviation $\sigma_0 = 0.3$, minimum standard deviation $\sigma_{min} = 0.05$ and a standard deviation decay rate of $\beta = 0.0001$. A mean attraction constant of the Ornstein-Uhlenback process is $\alpha = 0.2$.

was carried out with a small-variance Gaussian process. Figure 4.3 gives an impression of the scale and dynamics of the two random processes.

The training process consisted of training cycles lasting 10-15 minutes each. Human supervision took care of the visual performance assessment and potential archivisation of the latest agent, as well as the tuning of hyperparameters in-between cycles.

## 4.3 Simulation results

Agent performance can be evaluated using a convenient environment simulation interface programmatically; or manually, using a real-time interactive simulation where the user can interact with the loaded agents with two sliders, as shown in Figure 4.4. The lower slider is used to switch between stable and unstable agents, while the upper one prescribes horizontal reference signal to the currently active agent, after undergoing a smoothing moving average filter.



x1_ref 1.8447

y1_ref 0.98058

Figure 4.4: Interactive simulation window.

In the rest of this section, we summarise many simulation results concerning the different use cases of the agents.

### 4.3.1 Stable agent

The most important task of the stable agent is the movement of the mass with minimal swaying, Figure 4.5 shows the trajectory of the final stable agent. The agent is capable of tracking the reference position with a relatively constant error of 0.2 metres. Unfortunately, there is always a static error of 8 centimetres, the agent structure and the training framework could not guarantee the complete elimination of this steady state error. The root cause seems to be a non-zero actor action in standstill, error-free configuration, which is only cancelled out by slightly pushing the cart off the target. The following have been tried to alleviate this effect:

- Increasing the mass horizontal position error term $P_{x_1}$ to the limit of stability.
- Increasing the discount factor $\gamma$ and episode length.
- Gradually reducing learning rate and noise variance $\sigma_t$.
- Switching from Ornstein-Uhlenbeck to Gaussian noise.
- Rerunning the training multiple times. Sometimes, the resulting static error was down to pure chance.

In a practical setting, the cleanest solution could probably be offsetting the horizontal positions with the static error. It is also remarkable, how the swaying angle has been contained within $\pm 10°$. To achieve this, a quite erratic actuation strategy was

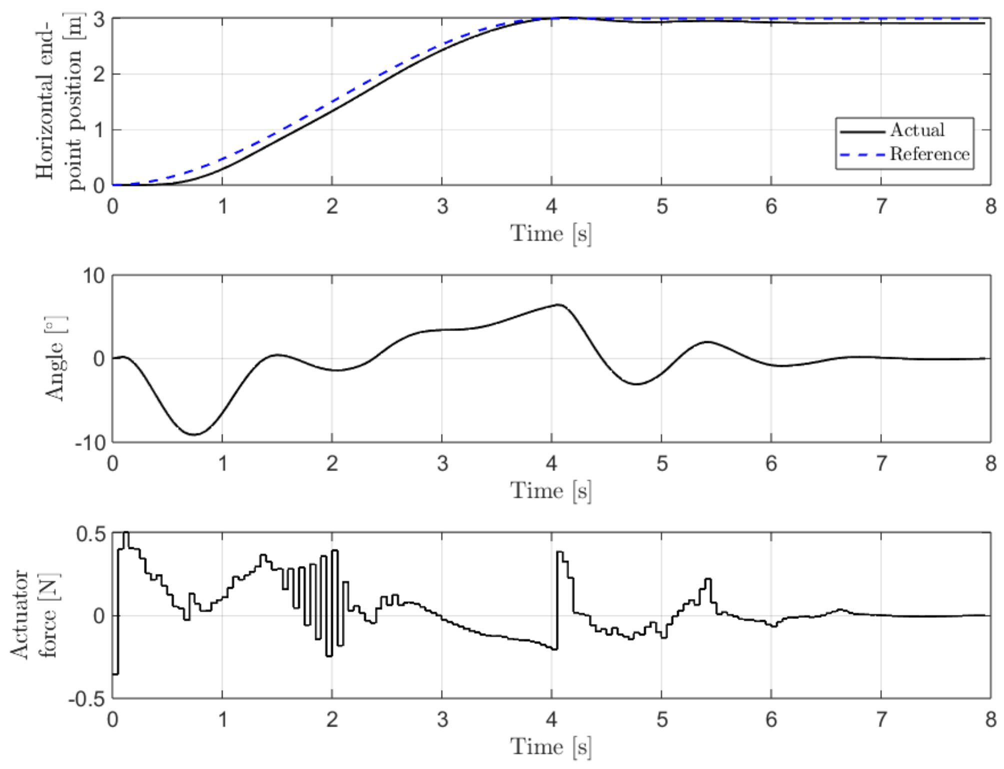


Figure 4.5: Stable agent performance in response to a transition reference trajectory.

picked up by the actor, which displays many discontinuities in response to smooth input change.

As with any control systems, a note on stability is warranted. Unfortunately, it is very complicated to provide strong stability guarantees for non-linear dynamical systems. As an alternative, it is customary to investigate the agent performance amidst perturbed environment, differing from the one it has been trained on. Figure 4.6 shows one such experimental setup, where the agent shows robustness to the perturbation of selected environment physical parameters and initial conditions. Although, the tracking performance and static error are affected, none of the trajectories showed uncontrollable divergence from the nominal trajectory.

A similar experiment is shown in Figure 4.7 with respect to the swaying elimination capabilities during an unstable-stable transition. Angle damping is greatly affected by parameter change, often extending the manoeuvre by multiple seconds, but the trajectories all converge to the stable position eventually.

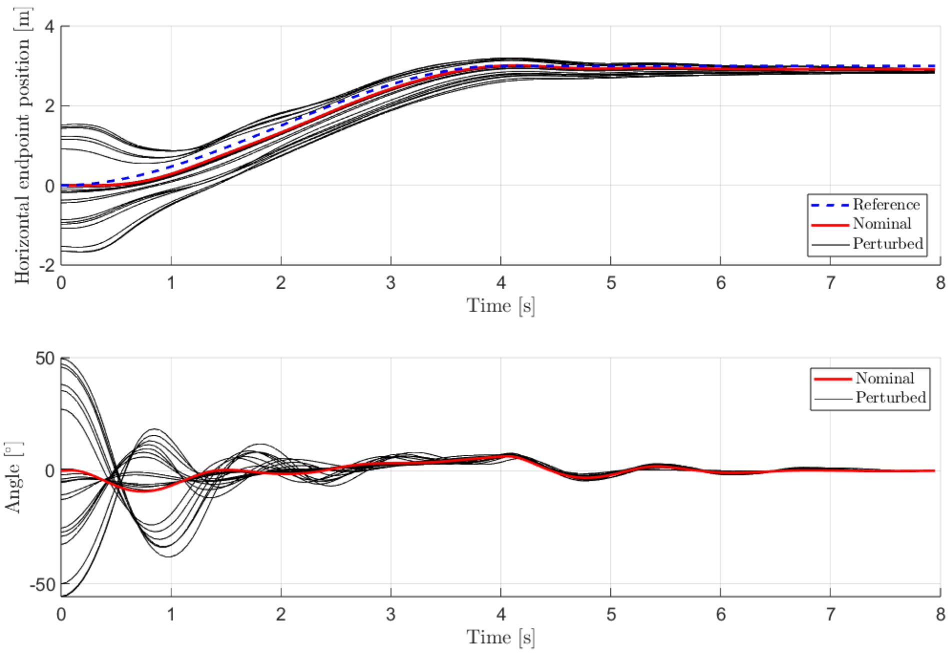


Figure 4.6: Same stable agent interacting with different parametrisations of the environment and initial swaying angles, during a transition manoeuvre. Affected environment parameters ($M_0$, $M_1$, $b_0$, $b_1$) have been perturbed within a range from 40% to 160% of their nominal values. Starting angles have been randomly selected from a 120° wide range around the stable fixed point.

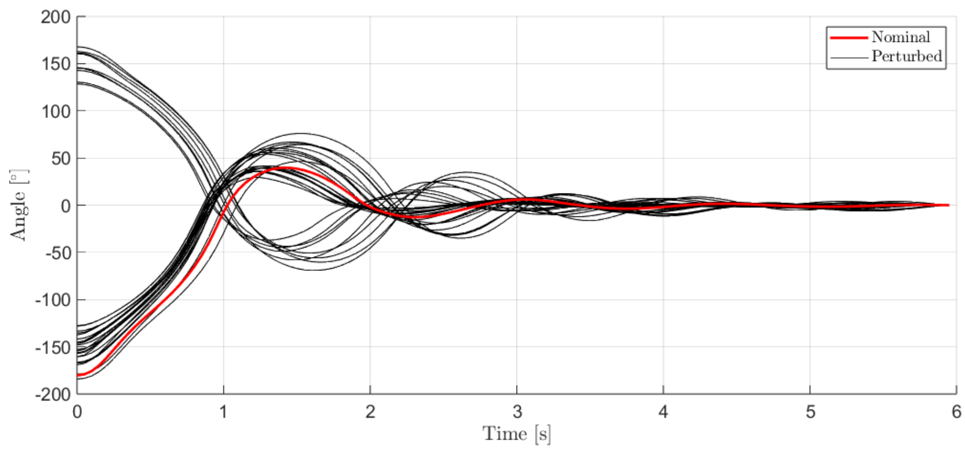


Figure 4.7: Same stable agent interacting with different parametrisations of the environment and initial swaying angles, during a swaying elimination manoeuvre. Affected environment parameters ($M_0$, $M_1$, $b_0$, $b_1$) have been perturbed within a range from 40% to 160% of their nominal values. Starting angles have been randomly selected from a 120° wide range around the unstable fixed point.

### 4.3.2 Unstable agent

Similarly to the stable agent, we summarise the robustness results of the final unstable agent with respect to the transition (Figure 4.8) and swing-up (Figure 4.9) manoeuvres. Understandably, the unstable agent is less resilient to environmental changes, with a much narrower parameter range that the agent can tolerate. The transition trajectories show a spectacularly large overshoot, followed by rapid angular velocity reversals, until the mass finally converges to its final position with a static error of 15 centimetres.

It is remarkable how the agent was able to figure out a way to swing up the mass in a single pull. This was likely made possible by the oversized actuator capabilities compared to the small masses. Despite of this, swing-up via multiple back and forth motion has also been observed at certain phases of the training, so the agent is definitely able to pass up on short-term rewards and look ahead over longer durations. The agent seems to prefer pulling the cart to the left, even when the positive initial angle would suggest the opposite direction. This could be explained by the neural network size restrictions, as there were no more free neurons to isolate this set of initial angles.

Proving the stability and robustness of neural network-based controllers is an extremely difficult task, as we can't easily tell how the network would extrapolate to observations it had never seen earlier during training. Only by exposing the agent to long and tedious training could we shed light on these critical regions, where the training progress indicated plummeting cumulative reward for a few episodes (Figure 4.10). With agent maturity, these phenomena were occurring more and more rarely, just as our confidence in the agent was growing.

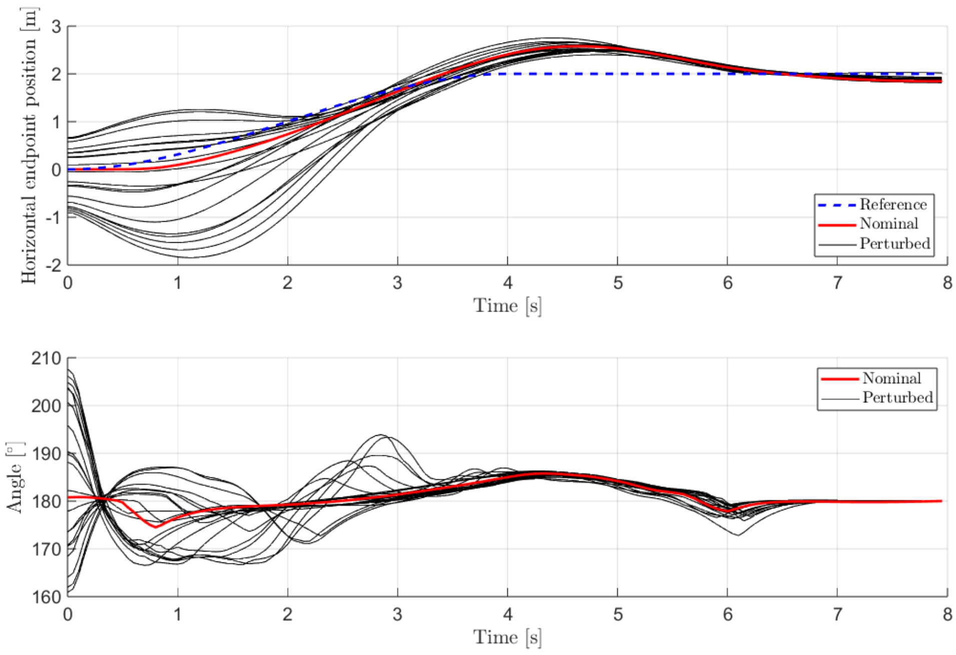


Figure 4.8: Same unstable agent interacting with different parametrisations of the environment and initial swaying angles, during a transition manoeuvre. Affected environment parameters ($M_0$, $M_1$, $b_0$, $b_1$) have been perturbed within a range from 70% to 130% of their nominal values. Starting angles have been randomly selected from a 60° wide range around the unstable fixed point.

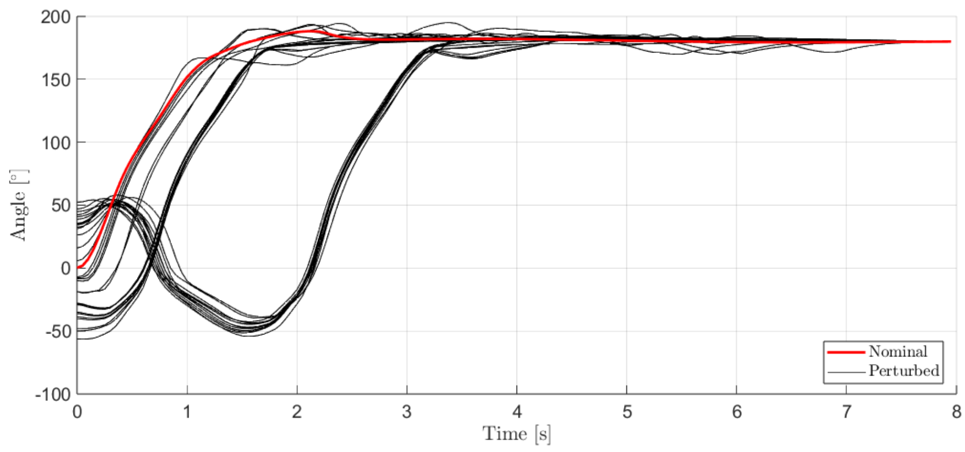


Figure 4.9: Same unstable agent interacting with different parametrisations of the environment and initial swaying angles, during a swing-up manoeuvre. Affected environment parameters ($M_0$, $M_1$, $b_0$, $b_1$) have been perturbed within a range from 70% to 130% of their nominal values. Starting angles have been randomly selected from a 120° wide range around the stable fixed point.

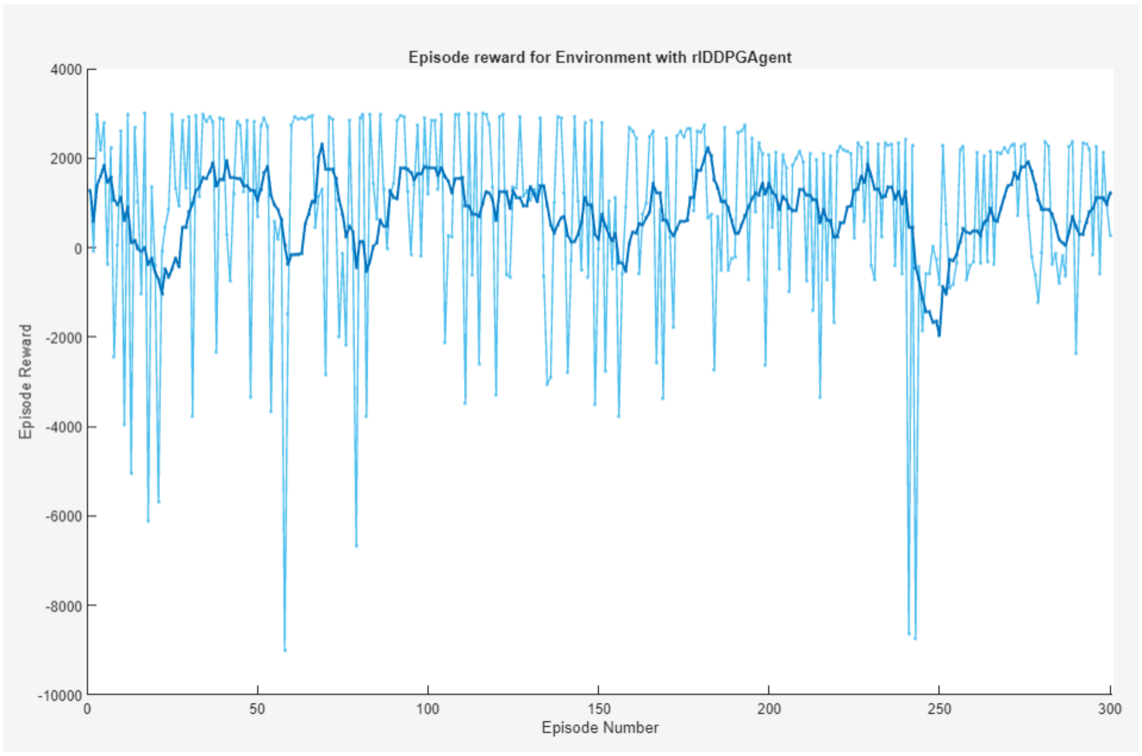


Figure 4.10: Training progress during the final stages of the unstable agent. Episodes can represent different types of trajectories with wildly different expected cumulative reward, hence the large variance. Occasional large negative reward spikes show a critical operating range corresponding to the swing-up manoeuvre that the agent had not yet learned to handle. Plotted with default Matlab *rlTrainingOptions*, where the light blue line shows the cumulative discounted reward obtained by the agent during subsequent episodes of the training process.

# Chapter 5

# Summary

The aim of this thesis was to give an introduction to the Deep Deterministic Policy Gradient algorithm before applying it to the popular cart pole problem. A reasonable compromise has been struck between succinctness and thoroughness, as we outlined the fundamental mathematical theorems underpinning the algorithm. A quick look at the numerous, often quite technical hyperparameters of the DDPG reinforces the view that a deep understanding of its mathematical background is very beneficial in ensuring its effective application.

After this general introduction, we turned our attention to covering every aspects of our simulated cart pole system, starting with the environment. First, we derived the system equations from classical dynamics principles. We used this simulated environment as a proxy for a potential real-world use case, and we briefly touched on the *Sim-to-Real* knowledge transfer technique necessary for deploying these prepared agents. An elaborate trajectory generation and reward module were also of great importance, their implications on the training process was thoroughly investigated.

We then moved on to discussing the various agent and training details. We achieved the swing-up manoeuvre and unstable balancing with a $\sim$52k-parameter actor, which is 60% less than the one used in the original DDPG paper. Although reinforcement learning is said to be the most sample-inefficient machine learning method, we have found that DDPG training was able to produce acceptable results relatively quickly. However, long and tedious training was required to approach the performance of classical controllers in the linear regions (close to the fixed points).

There are several promising directions this project can move in the future. Some authors have shown that a kinematic (position/velocity-based) actuation strategy is preferable over a kinetic (force-based, our current approach) one from a *Sim-to-Real* standpoint. Neural network architecture could also be revisited, as there are several reports of recurrent networks with LSTM layer being successfully used in reinforcement learning control. Besides, it could be worth enhancing the environment with the most typical real-world effects, like non-linear friction, to see if the agent is able to overcome them. Finally, model-based reinforcement algorithms (DDPG is model-free) show potential in improving sample-efficiency, and it would be interesting to see what they have in common with the proven out Model Predictive Control approaches.

# Bibliography

[1] Kevin P. Murphy. “Probabilistic Machine Learning: Advanced Topics”. In: (2023). URL: `http://probml.github.io/book2`.

[2] Pouria Razzaghi et al. “A Survey on Reinforcement Learning in Aviation Applications”. In: (2022). arXiv: `2211.02147 [eess.SY]`.

[3] MathWorks. “Deep Deterministic Policy Gradient (DDPG) Agents”. In: (). URL: `https://www.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html`.

[4] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: ICML’14 (2014), I–387–I–395.

[5] D. Bertsekas. “Reinforcement learning and optimal control”. In: (2019).

[6] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (1988), pp. 9–11. DOI: `10.1007/BF00115009`. URL: `https://doi.org/10.1007/BF00115009`.

[7] Richard S. Sutton. “Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding”. In: *Proceedings of the 8th International Conference on Neural Information Processing Systems*. NIPS’95. Denver, Colorado: MIT Press, 1995, 1038–1044.

[8] R. Bellman. “Dynamic Programming”. In: (1957).

[9] Christopher Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8 (1992), pp. 279–292. URL: `https://api.semanticscholar.org/CorpusID:208910339`.

[10] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: (2019). arXiv: `1509.02971 [cs.LG]`.

[11] G. E. Uhlenbeck and L. S. Ornstein. “On the Theory of the Brownian Motion”. In: *Physical Review* 36.5 (Sept. 1930), pp. 823–841. DOI: 10.1103/PhysRev.36.823.

[12] Bhuvaneswari S. “Reinforcement Learning for Cart Pole system”. In: (2017). URL: https://www.youtube.com/watch?v=qMlcsc43-lg&ab_channel=BhuvaneswariS.

[13] Stephen James et al. “Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks”. In: (2019). arXiv: 1812.07252 [cs.RO].

[14] Joanne Truong et al. “Rethinking Sim2Real: Lower Fidelity Simulation Leads to Higher Sim2Real Transfer in Navigation”. In: (2022). arXiv: 2207.10821 [cs.RO].

[15] Gábor Stépán Gábor Csernák. “Rezgéstan”. Hungarian. In: *Budapesti Műszaki és Gazdaságtudományi Egyetem, Műszaki Mechanikai Tanszék* (2019).

[16] Hiroshi Akima. “A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures”. In: *J. ACM* 17.4 (1970), 589–602. ISSN: 0004-5411. DOI: 10.1145/321607.321609. URL: https://doi.org/10.1145/321607.321609.