A Deep Deterministic Policy Gradient approach for Stabilisation of the Double Inverted Pendulum on a Cart

Casper Luiten

***Abstract*—This work presents the design, implementation and results of a Deep Deterministic Policy Gradient (DDPG) agent that is used for the stabilisation of a Double Inverted Pendulum on a Cart (DIPC). The design choices for this agent are explained and important parameters for its training are tuned. Additionally, the performance of the agent is demonstrated and analysed. It is ultimately concluded that the DDPG agent can be a suitable choice for the defined stabilisation task.**

I. Introduction

As application fields of dynamical systems become more complex, the derivation of accurate models for these systems becomes more tedious. As a result, the implementation of state of the art control strategies such as Model Predictive Control (MPC) [1] is infeasible for applications where no model is available. To design controllers for such applications one has to investigate non-model-based strategies. This is where the field of Reinforcement Learning (RL) provides a possible solution. In this work the design, implementation and results of a Deep Deterministic Policy Gradient (DDPG) agent for the stabilisation of a Double Inverted Pendulum on a Cart (DIPC) are presented. The goal of this agent is to keep the DIPC system in the upright centered position, where the cart is located in the origin and the double pendulum is perpendicular to the horizontal plane. From this the main research question of this work arises: Is the DDPG agent a suitable candidate for the stabilisation task of the DIPC system?

The development of this agent starts by creating a learning environment. The dynamics of the DIPC system will be derived and the necessary environment properties will be defined. Next, the DDPG design steps will be presented. The algorithm will be shown and the structure of the agent will be explained. Subsequently, this agent will be trained and important parameters in the training process will be tuned. Finally, the agent will be validated and the control task will be performed and examined.

II. Learning Environment

In this section, the learning environment for the DDPG agent will be constructed. First, the dynamics of the DIPC system will be described. Subsequently, the environment will be initialized and the internal parameters of the environment such as the observation space, the action space, the reward function and the thresholds for termination of the episode will be defined.

Casper Luiten is a master student at TU Delft, The Netherlands. E-mail address: {C.J.Luiten @student.tudelft.nl}

A. Nonlinear DIPC dynamics

In figure 1, a graphical depiction of the DIPC is provided. The system states, which are given by θ in the figure, are defined as follows:

- $\theta_0 = x$: Horizontal position cart
- θ_1: Angle rod 1 pendulum
- θ_2: Angle rod 2 pendulum
- $\dot{x}$: Horizontal velocity cart
- $\dot{\theta}_1$: Angular velocity rod 1 pendulum
- $\dot{\theta}_2$: Angular velocity rod 2 pendulum

Various other system parameters can be seen in figure 1. The used parameter values are as follows:

- $m_0 = 1.5$;
- $m_1 = 0.75$
- $m_2 = 0.5$
- $L_1 = 0.75$
- $L_2 = 0.75$
- $g = 9.81$

The horizontal force u on the cart is the only input in the system.

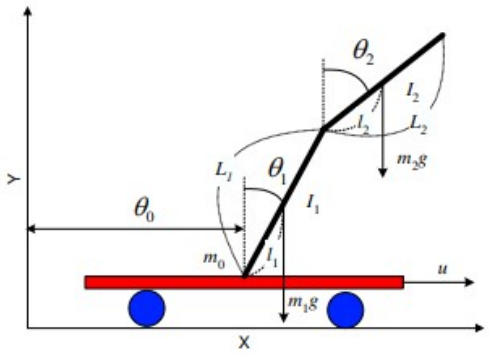

Fig. 1. A graphical description of a DIPC [2]

To find the dynamics of the system Langrangian mechanics can be used. In this method, the potential and kinetic energy in a system are used to formulate the dynamics of a system based on the stationary action principle [3]. The Lagrange equation is defined as follows:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \hat{\theta}}\right) = \frac{\partial L}{\partial \theta} = Q \tag{1}$$

Where $L = T - P$ is a Lagrangian, T is the kinetic energy, and P the potential energy. To derive the equations of motion, one has to derive the kinetic and potential energy of all the components of the system. In [2] the full derivation of the dynamics can be found. Below, a compact notation of the system dynamics is provided:

$$\mathbf{D}(\theta)\ddot{\theta} + \mathbf{C}(\theta, \dot{\theta})\dot{\theta} + \mathbf{G}(\theta) = \mathbf{H}u \tag{2}$$

Where,

$$\mathbf{D}(\theta) = \begin{pmatrix} d_1 & d_2\cos\theta_1 & d_3\cos\theta_2 \\ d_2\cos\theta_1 & d_4 & d_5\cos(\theta_1-\theta_2) \\ d_3\cos\theta_2 & d_5\cos(\theta_1-\theta_2) & d_6 \end{pmatrix} \tag{3}$$

$$\mathbf{C}(\theta, \dot{\theta}) = \begin{pmatrix} 0 & -d_2\sin(\theta_1)\dot{\theta}_1 & -d_3\sin(\theta_2)\dot{\theta}_2 \\ 0 & 0 & d_5\sin(\theta_1-\theta_2)\dot{\theta}_2 \\ 0 & -d_5\sin(\theta_1-\theta_2)\dot{\theta}_1 & 0 \end{pmatrix} \tag{4}$$

$$\mathbf{G}(\theta) = \begin{pmatrix} 0 \\ -f_1\sin\theta_1 \\ -f_2\sin\theta_2 \end{pmatrix} \tag{5}$$

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \tag{6}$$

It is assumed that the center of mass of the rods coincides with the geometrical center, thus the following expressions for d_i and f_i can be found:

$$\begin{aligned} d_1 &= m_0 + m_1 + m_2 \\ d_2 &= m_1 l_1 + m_2 L1 = \left(\frac{1}{2}m_1 + m_2\right) L_1 \\ d_3 &= m_2 l_2 = \frac{1}{2} m_2 L_2 \\ d_4 &= m_1 l_1^2 + m_2 L_1^2 + I_1 = \left(\frac{1}{3}m_1 + m_2\right) L_1^2 \\ d_5 &= m_2 L_1 l_2 = \frac{1}{2} m_2 L_1 L_2 \\ d_6 &= m_2 l_2^2 + I_2 = \frac{1}{3} m_2 L_2^2 \\ f_1 &= (m_1 l_1 + m_2 L_1)\, g = \left(\frac{1}{2}m_1 + m_2\right) L_1 g \\ f_2 &= m_2 l_2 g = \frac{1}{2} m_2 L_2 g \end{aligned} \tag{7}$$

Following this step we can now construct the state vector $\mathbf{x} = \begin{pmatrix} x & \theta_1 & \theta_2 & \dot{\theta}_0 & \dot{\theta}_1 & \dot{\theta}_2 \end{pmatrix}^T$ which can be used to reformulate the above system to an ordinary 6-th order nonlinear differential equation:

$$\dot{\mathbf{x}} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & -\mathbf{D}^{-1}\mathbf{C} \end{pmatrix} \mathbf{x} + \begin{pmatrix} \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{G} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{H} \end{pmatrix} u \tag{8}$$

Where $\mathbf{0}$ denote the zero matrices, each of sufficient size.

B. Environment construction

By definition, the environment is the world in which the agent exists. The agent can perform actions and observe a reaction in the environment, however the agent can not change the rules and dynamics of the environment. To create the environment for the DIPC system, the following features have to be defined [4]:

1) Observation space: the information from the environment that can be extracted by the agent. It is assumed that the agent has access to all the states of the DIPC. Therefore the observation space $\mathcal{O}$ is defined as:

$$\mathcal{O} \in \mathbb{R}^6 \tag{9}$$

2) Action space: the allowable actions from the agent that can alter the states in the environment. According to the DIPC model, the horizontal force on the cart is the only input to the system. This results in the following action space $\mathcal{U}$:

$$\mathcal{U} \in \mathbb{R} \tag{10}$$

3) Step function: a function that describes the dynamics of the environment for one time step. Given a certain current state and action, the differential equation in equation 8 is used to compute the next state. This is done by Forward Euler integration with time step $\mathrm{d}t = 0.01$ sec.

4) Reward function: a function that returns a reward based on the state of the environment. Using this reward, the agent can be told which actions are preferable over others. Since the goal is to stabilize the DIPC in the upward position, the reward function is an adjusted version of the Linear Quadratic Regulator (LQR) cost [5]:

$$r(\mathbf{x}, u) = C_r \exp\left(-\frac{1}{2}\left(\mathbf{x}^T W_x \mathbf{x} + W_u u^2 + W_{\Delta u}(\Delta u)^2\right)\right) \tag{11}$$

Where the weights W_x, W_u and $W_{\Delta u}$ are used to stress relative importance of errors in the state, input and input difference respectively. The constant C_r can be used for scaling. The reward resembles a bell curve where the maximum reward of C_r can be achieved when the state, input and input difference are all zero. For this environment, using the negative LQR cost directly as reward is not feasible, since the agent has a cheat option. By shortening the episode length, the accumulation of rewards will contain less negative elements, resulting in a higher total reward.

5) Thresholds: the boundary conditions that specify when an episode in the environment must be terminated. Terminating the episode once a certain threshold has been reached greatly increases the efficiency of the agent training. Reaching this threshold can occur in both positive and negative direction, i.e. the upper limit of the state is the positive threshold and the lower limit is the negative threshold. For the DIPC system, the following thresholds have been defined:

$$\begin{aligned} T_\theta &= 10^\circ \\ T_x &= 2 \text{ m} \end{aligned} \tag{12}$$

Where T_θ and T_x respectively denote the threshold for both angles θ_1, θ_2 and the threshold for the displacement of the cart x. In addition to these thresholds, bounds have been introduced to limit the search space of the agent. These bounds are defined by:

$$u_{\min} = -u_{\max} = -5 \text{ N} \tag{13}$$

6) Reset function: a function that, after termination, describes how to prepare the environment for the next episode. Once terminated, the state vector of the DIPC system is set equal to a random initial condition. All states are set to zero except for θ_2:

$$\begin{aligned} &\theta_2(0) = (2\alpha - 1)\delta_\theta \\ \implies &\mathbf{x}(0) = \begin{pmatrix} 0 & 0 & \theta_2(0) & 0 & 0 & 0 \end{pmatrix}^T \end{aligned} \tag{14}$$

Where $\alpha \in [0, 1]$ is a uniformly distributed random variable and $\delta_\theta > 0$ is set to a small angle. This random initiation is used to increase the adaptivity of the agent during training.

III. DDPG Design

In this section, the design of the DDPG agent will be discussed. First, the choice of implementing the DDPG method will be motivated. Next, the basic DDPG algorithm will be explained and a pseudocode will be presented. Subsequently, the actor and critic structures are defined. Finally, the implementation of the algorithm for the DIPC system will be elaborated on.

A. DDPG motivation

Since the environment of the DIPC system has a continuous observation and action space, a Q-learning based method such as "Deep Q Network" (DQN) [6] is not feasible [7]. In theory it is possible to discretize the action space, however in practice this results in very high dimensions for the discrete action space. This effect is especially problematic for the DIPC system, since the chaotic nature of the double pendulum requires a very fine discretization for successful control [8]. As such, the DDPG algorithm is a much more suitable candidate, since it can learn policies in continuous action spaces, eliminating the need for discretization [7]. In addition, the DDPG algorithm can handle complex high-dimensional nonlinear systems, of which the DIPC system is an example.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Fig. 2. Pseudocode for DDPG algorithm as described by Lillicrap et al. [7]

B. Algorithm

The description of the DDPG algorithm is well-documented in the original paper by Lillicrap et al. [7]. Based on this work, a brief summary of the algorithm will be provided here. In it's core, DDPG is an actor/critic based algorithm. Both the actor and critic are function approximators that use Neural Networks (NN) to find the policy and Q-value function respectively. In order to improve the stability of the method and reduce overfitting, the DDPG algorithm uses a replay buffer where past transitions are saved. A mini-batch containing a random set of transitions from the replay buffer is used to update the actor and critic. Additionally, the algorithm makes use of an offline "target" actor and critic to train the main online actor and critic networks. These target networks are slowly updated, further improving the stability of the DDPG method. To enhance the exploration ability of the agent, a noise is added to the actor policy. This noise is based on an Ornstein-Uhlenbeck process [9]. The above mentioned features can be recognized in figure 2.

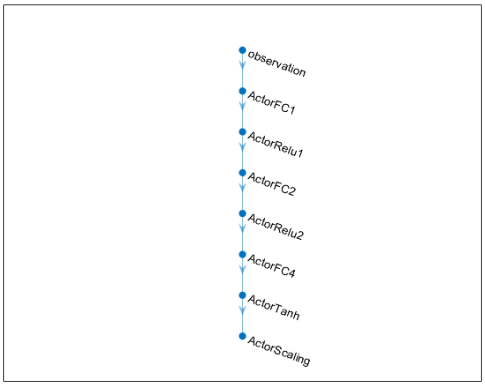

Fig. 3. Neural Network structure for the actor.

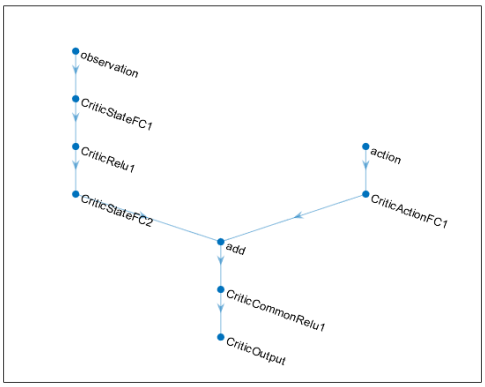

Fig. 4. Neural Network structure for the critic.

C. Actor/critic structure

As stated in section III-B the actor and critic use Neural Networks for the function approximations. For the DIPC system, the structure of these NNs was kept equal to the NN structures in the original DDPG work [7]. These networks each consist of one or two input layers, two hidden layers and one output layer.

1) Actor: the single input layer contains 6 neurons, one for each state. The output layer contains one neuron and uses a `tanh` activation function to bound the actions.

2) Critic: the two input layers contain 6 and 1 neurons respectively. The first input layer represents the states and the second layer represents the action. The first input layer is connected to the first hidden layer. This layer is connected to the second hidden layer, where the second input layer is connected as well. The second hidden layer is then connected to the output layer, which contains one neuron. This single neuron represents the Q-value computed by the critic.

For both the actor and critic, the two hidden layers contain 400 and 300 neurons respectively and for all hidden layers Rectified Linear Unit (ReLU) activation functions are used. Schematic representations of the actor and critic NN structures are shown in figures 3 and 4.

D. MATLAB Implementation

To create the DDPG agent, the Reinforcement Learning Toolbox [10] and Deep Learning Toolbox [11] are used in MATLAB. These toolboxes allow for an Object Oriented Programming approach, where instances of the environment and the agent can be created. The agent instance can then be passed to a training function that updates the NNs in the actor and critic structure of the agent. New observations can easily be obtained by updating the properties of the environment instance using its step function method. Once the performance of the agent is considered sufficient, the trained agent instance can be saved for later use in, for example, simulations. The code used for this work can be found in the following open source repository [12].

IV. Training

In this section, the DDPG agent as designed in section III will be trained. This is done by using an in-build training function in the Reinforcement Learning Toolbox in MATLAB [13]. This training function uses the "Adam" optimizer [14] to find the optimal update direction and step size for the actor and critic. In the following section, the initialization of the parameters that influence the learning process, often called "hyperparameters" is explained. Subsequently, the effects of tuning several of these hyperparameters are examined.

A. Initializing Hyperparameters

The DDPG algorithm has a high number of hyperparameters that can be difficult to tune [15]. To investigate the training behaviour of the agent, the hyperparameters are initialized according to the default settings of the DDPG agent object of the RL toolbox in MATLAB [16]. Some of the hyperparameters known to have a high influence are listed below:

- α_a: The learning rate of the actor. This value determines how big the steps in the network optimization of the actor are. Its default value is $\alpha_a = 0.01$.
- α_c: The learning rate of the critic. This value determines how big the steps in the network optimization of the critic are. Its default value is $\alpha_c = 0.01$.
- $\sigma_{\mathcal{N}}$: The standard deviation of the action exploration noise. When this value is set high, the agent is more likely to explore. Its default value is $\sigma_{\mathcal{N}} = 0.3$;
- τ: The target smooth factor. This value determines how fast the target actor and critic networks respond to the online networks. Its default value is $\tau = 0.001$

B. Tuning

In addition to changing the hyperparameters, tuning the reward function as defined in equation 11 greatly affects the training results as well. Through trial and error the following values have been found to be suitable:

$$\begin{aligned} C_r &= 2 \\ W_x &= \text{diag}(1,\ 10,\ 10,\ 2,\ 20,\ 20) \\ W_u &= 0.1 \\ W_{\Delta u} &= 1 \end{aligned} \tag{15}$$

Using this reward function and the default values as specified in section IV-A, the training diagram shown in figure 5 can be generated. In this plot the light blue line shows the episode rewards and the dark blue lines show the average reward of the last 5 episodes. It can be seen that as the episode number grows, the reward for each episode gradually increases, with sparse reward peaks of approximately 2200 near the end of the training session.

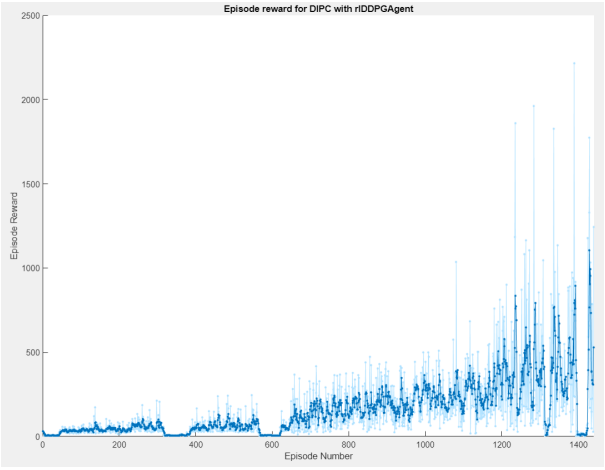

Fig. 5. Training session with default hyperparameters.

It is important to note that though the policy of the DDPG method is deterministic, the results of the training process are not. The training process is stochastic due to the exploration noise of the agent and the random variable in the reset function of the environment. This means that the set of initial conditions

is random for each training session and even for equal starting parameters the outcome of a training session can be different, increasing the difficulty of finding suitable hyperparameters. To allow for more successful training sessions where higher rewards can be achieved, the previously mentioned hyperparameters have been tuned to examine their effect:

1) Learning rate α: In general, it is good practice to keep the learning rate of both the actor and critic low since this improves the convergence to an optimum. However, it was found that for very low values of α_a and α_c the algorithm has too much difficulty escaping local optima, resulting in a slower training process and ultimately lower rewards after an equal number of episodes. An example of a training process with low learning rates is shown in figure 6. It can be seen that the shape of the training curve is similar, however for an equal amount of episodes the episode rewards are significantly lower than those presented in figure 5. Inversely, it was found that if the learning rates are set very high the agent is less likely to converge to a desired optimum, since it is more likely to "fall out" of this optimum once reached.

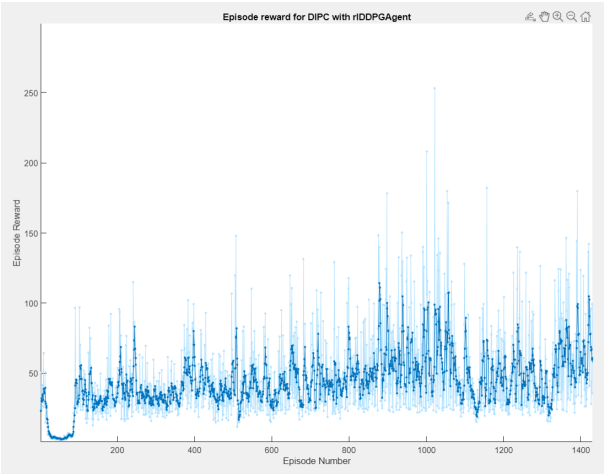

Fig. 6. Training session with default hyperparameters and $\alpha_a = \alpha_c = 1e{-}5$.

2) Exploration noise standard deviation $\sigma_{\mathcal{N}}$: Since the DDPG computes a deterministic policy, it is essential to promote action exploration by using a noise on the action. If the standard deviation of this noise is set to a very small value the agent is not able to escape from local optima and is very likely to not reach the desired behaviour. For an opposite scenario, where the standard deviation is very high, the agent is likely to escape from an optimum, which is preferable for low reward optima but undesirable at higher rewards. Figure 7 shows a training process of such a scenario. It can be seen that the agent is struggling to stay on policy and the episode rewards constantly decrease sharply. This is presumably caused by the agent escaping its local optimum and results in a lot of the previous training development being lost. Due to this, the performance of the agent after 1400 episodes is significantly less when compared to the results shown in figure 5.

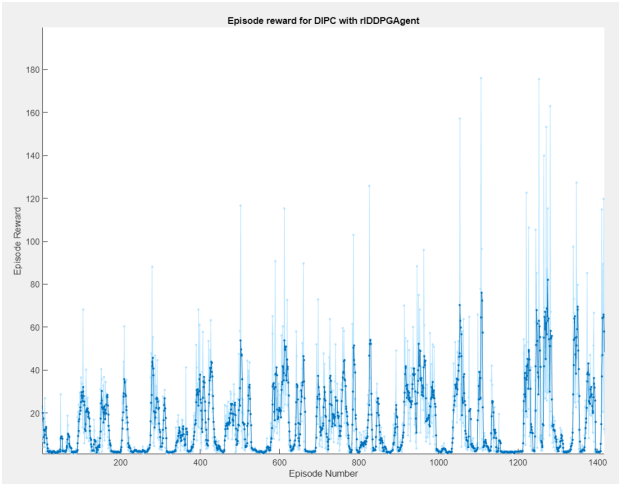

Fig. 7. Training session with default hyperparameters and $\sigma_{\mathcal{N}} = 3$.

3) Target smooth factor τ: Since this parameter controls the delay between the online and target networks, its effect is comparable to that of the learning rates α_a and α_c. If the learning rate of the online networks is high and the target smooth factor is low, the unstable learning behaviour can be smoothed out by the slow target network development. Reversely, if the learning rate is low and the smooth factor high, the target networks and online networks will become more similar. In its maximum (τ = 1), the target networks and online networks behave identically, revealing that the target networks can only develop as fast as their online counterparts. In figure 8 a training session can be seen with default learning rates and a low target smooth factor. As explained it can be seen that the agent learns more slowly, improving the stability of the process. Due to this slow behaviour, the increase in episode reward between 200 and 1400 episodes is barely noticeable.

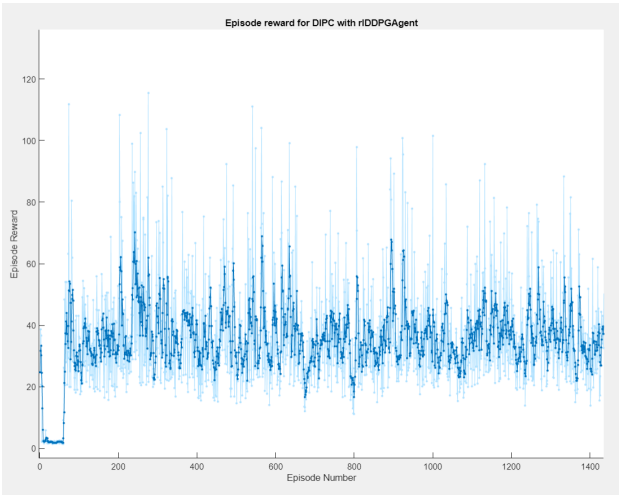

Fig. 8. Training session with default hyperparameters and $\tau = 1e - 5$.

C. Final training session

Now that the effect of tuning some highly influential hyperparameters is know, this knowledge can be used to perfect the learning procedure. In figure 9, the final training session can

be seen that produced a sufficiently effective agent. During training, for any episode reward greater than 1500 the agent parameters are saved. After training, these different agents can be evaluated by simulation and compared. The best agent is selected and used for the control goals. These final two steps are further explained in section V. The hyperparameters used in the final training session are given by:

$$\begin{aligned} \alpha_a = \alpha_c &= 1e-3 \\ \sigma_{\mathcal{N}} &= 0.3 \\ \tau &= 1e-3 \end{aligned} \tag{16}$$

It can be concluded that the initial values for the hyperparameters produced a fairly successful training session and only minor changes to the learning rate were necessary to achieve the final result.

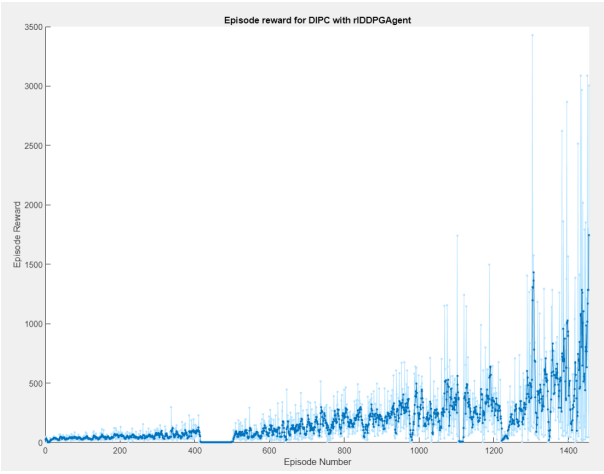

Fig. 9. Final training session.

V. Validation and Controller Results

From the final training session as described in section IV-C, the most suitable agent can be used for control purposes. However, selecting this agent requires a form of validation. In this section, the agent validation and subsequent controller results will be discussed.

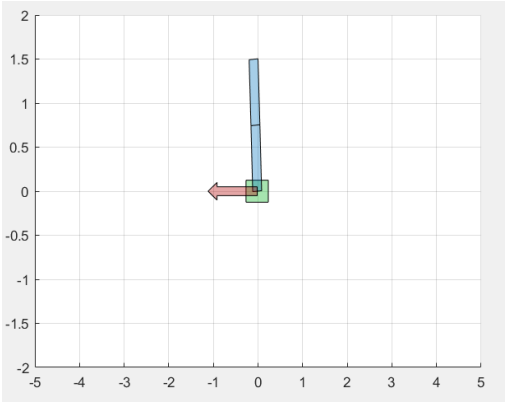

Fig. 10. Frame of the animation of the DDPG controlled DIPC.

A. Validation

To select the "best" agent from a list of candidates, it was decided that a visual comparison of their performance during several simulations is sufficient. In MATLAB, the simulation is constructed using an in-build function that is a method of the environment object [17]. A separate function is defined to plot a figure and create a visualization of the DIPC where the states and input are retrieved from the environment object. Combining these two functions results in an animated simulation of the DDPG agent controlled DIPC system, of which a frame is shown in figure 10. It can be seen that for the pendulum rods blue polygons have been used. The cart is depicted as a green polygon and the input force is shown as a red arrow. Different agents can be used for this animated simulation, which allows for comparison between them and ultimately the best performing agent (i.e. the agent that can keep the DIPC in the central upright position for the longest amount of time with the smallest amount of error) is selected.

B. Controller Results

Using the validation method as explained in section V-A the list of agent candidates resulting from the final training in section IV-C can be reduced to a single agent that is most compatible for the stabilisation task. The performance of the agent will be investigated more thoroughly in the next section. Additionally, the agent is implemented in a reference tracking setting of which the simulation is analyzed as well.

1) Stabilisation: Figure 11 shows a plot of the first three states during a simulation with the DDPG agent. Additionally, the input force u is shown. The initial conditions for the simulation are depicted, where the values for the angles are shown in degrees. It is important to note that these initial conditions are chosen such that they do not resemble the states after a reset of the environment. A video of the animation of this simulation can be found here [12].

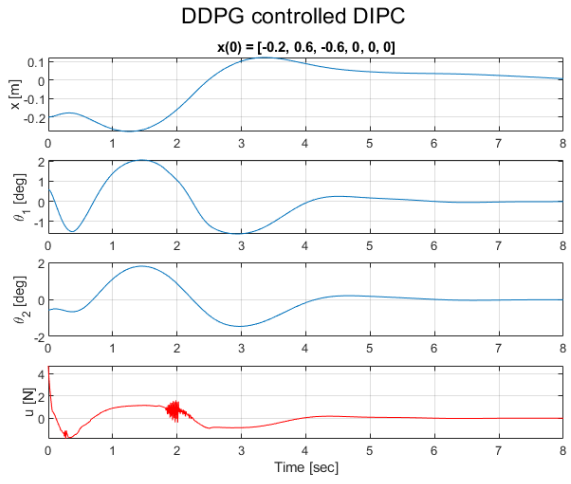

Fig. 11. Simulation of the DDPG controlled DIPC system using the final agent.

It can be seen that the agent is able to successfully stabilize the DIPC system after approximately 8 seconds. The

angles of the pendulum converge to zero faster than the cart displacement, which is likely closely related to the chosen reward weights shown in equation 15. With these weights, the agent has learned a strategy where the angles of the pendulum are stabilized first after which the cart displacement is slowly pushed towards the origin. In addition, the agent has learned to first move in opposite direction in order to change the cart's position. Since the DIPC is an example of a non-minimum-phase system [18], this strategy is imperative for success. In the input plot, it can be seen that for a limited time the agent provides a jittery input signal, where the action is changed very rapidly, at approximately 2 seconds. In a real life application this jittering might be problematic due to limitations of the actuators, however for this simulation it does not pose a problem.

2) Reference tracking: To test the diversity of the agent, a simple reference tracking task is designed. Instead of directly providing observations from the environment to the agent, an offset x_{ref} is applied to the position of the cart:

$$x_{\text{new}} = x - x_{\text{ref}} \tag{17}$$

As a result, the agent believes that the system is not in the origin and tries to correct for this, ultimately moving the cart to the desired position. Figure 12 shows the results of this reference tracking task. The reference signal to track is a scaled Heaviside step function [19] $x_{\text{ref}}(t) = C_S H(t-8)$ with $C_S = 0.295$ that kicks in at $t = 8$ sec. A video of the animation of this simulation is can be found here [12].

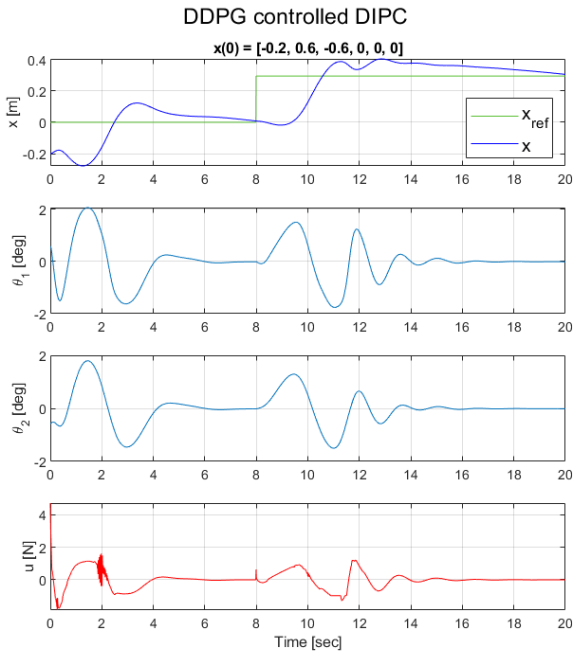

Fig. 12. Simulation of the DDPG agent performing a reference tracking task on the DIPC system.

From the results it can be observed that the agent applies a small negative force immediately after the step in the reference signal to let the pendulum "fall" in the positive x direction. This behaviour shows once more that the agent is capable of dealing with the non-minimum-phase characteristics of the system. It was found that the reference tracking task can not be reliably performed for steps in the reference signal greater than 0.3.

VI. Discussion

Throughout the development of this work, multiple interesting discussion points have been discovered. In this section, the goal is to list and briefly explain these topics. Then, possible areas for future work will be presented.

A. Points of Discussion

1) Sensitivity to reward function: Since the reward function gives the agent feedback on which actions are desired and which are less preferable, changing this function greatly influences the development of the training sessions. As speculated in section V-B1, changing the weights of the reward function might drastically change the strategy of the final agent.

2) Sensitivity to environment reset: During training, it was discovered that the performance of the resulting agent is highly sensitive to the design of the reset function. The agents trained to stabilize the system from a certain initial angle range $\delta\theta_2(0) = 2\theta_2(0)$ showed very poor performance when this range was increased by a small amount. If the range was decreased, the agents did not show any loss of performance. This phenomenon might be closely related to the termination thresholds of the environment and may also be caused by the input bounds introduced in equation 13.

3) Optimal stopping point for training: As a result of the random variables in both the agent and reset function and due to the high complexity of the DIPC system, determining when to stop a training session proved to be a hard task. Even with the effect of some of the hyperparameters known, it remains difficult to predict the behaviour of the training process and on multiple occasions the training process improved mid-session unexpectedly. The opposite also occurred numerous times.

4) Numerical Validation: Finding the most suitable candidate from a list of agents via a visual performance analysis is most likely not optimal. If a more strict numerical validation method would be implemented, the performance of the DDPG agent might differ from the results shown in this report.

B. Future Work

To improve the performance of the DDPG based controller more research can be done on the influence of the reward weights of the environment and hyperparameters of the agent. This research can be facilitated by a well defined numerical validation method that can rank agents reliably. Additionally, the robustness of the DDPG controller can be improved by training the agent for a greater range of states.

VII. Conclusion

The purpose of this work was to investigate the suitability of the DDPG agent for the task of stabilizing the DIPC system. Through effective tuning of the training process, an agent was

developed that showed promising results during simulation. The agent was able to keep the pendulum in the upright position for a random initial condition. For a simple reference tracking task, the agent was able to move the cart to a desired point whilst keeping the pendulum in the upright position. It can therefore be concluded that the DDPG algorithm can provide an agent that is capable of successfully completing the stabilisation task.

Based on the results found in this report, it is recommended to execute more training sessions with DDPG agents to further investigate their capabilities for more extreme stabilisation scenarios where the initial conditions result in a greater challenge for the agent. Additionally, it is recommended to investigate alternative continuous RL strategies such as Twin Delayed DDPG (TD3) [20], Proximal Policy Optimization (PPO) [21] or Soft Actor Critic (SAC) [22] methods and compare their results with the results found in this report.

REFERENCES

[1] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109889900022

[2] A. Bogdanov, "Optimal control of a double inverted pendulum on a cart," 12 2004, (Accessed on 03/27/2022).

[3] "Lagrangian mechanics," https://en.wikipedia.org/wiki/Lagrangian_mechanics, (Accessed on 03/27/2022).

[4] R. S. Sutton, A. G. Barto *et al.*, "Reinforcement learning," *Journal of Cognitive Neuroscience*, vol. 11, no. 1, pp. 126–134, 1999.

[5] A. J. Shaiju and I. R. Petersen, "Formulas for discrete time lqr, lqg, leqg and minimax lqg optimal control problems," *IFAC Proceedings Volumes*, vol. 41, pp. 8773–8778, 2008.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[8] P. A. Barnes, M., and Bozack, "Chaos in a double pendulum," 2004.

[9] J. Stein, S. R. C. Lopes, and A. V. Medino, "A generalization of the ornstein-uhlenbeck process: Theoretical results, simulations and parameter estimation," 2021. [Online]. Available: https://arxiv.org/abs/2108.06374

[10] MathWorks, "Reinforcement learning toolbox documentation," https://nl.mathworks.com/help/reinforcement-learning/, (Accessed on 08/31/2022).

[11] ——, "Deep learning toolbox documentation," https://nl.mathworks.com/help/deeplearning/, (Accessed on 08/31/2022).

[12] C. Luiten, "Github repository for matlab code," https://github.com/CasperLuiten/DDPG-DIPC, September 2022.

[13] MathWorks, "Train reinforcement learning agents," https://nl.mathworks.com/help/reinforcement-learning/ug/train-reinforcement-learning-agents.html, (Accessed on 08/31/2022).

[14] ——, "Training options for adam optimizer," https://nl.mathworks.com/help/deeplearning/ref/nnet.cnn.trainingoptionsadam.html, (Accessed on 08/31/2022).

[15] N. Ashraf, R. Mostafa, R. Sakr, and M. Rashad, "Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm," *PLOS ONE*, vol. 16, p. e0252754, 06 2021.

[16] MathWorks, "Deep deterministic policy gradient reinforcement learning agent," https://nl.mathworks.com/help/reinforcement-learning/ref/rlddpgagent.html, (Accessed on 08/31/2022).

[17] ——, "Simulate trained reinforcement learning agents within specified environment," https://nl.mathworks.com/help/reinforcement-learning/ref/rl.env.abstractenv.sim.html, (Accessed on 09/01/2022).

[18] L. Qiu and E. Davison, "Performance limitations of non-minimum phase systems in the servomechanism problem," *Automatica*, vol. 29, no. 2, pp. 337–349, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000510989390127F

[19] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals Systems (2nd Ed.)*. USA: Prentice-Hall, Inc., 1996.

[20] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[22] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2018. [Online]. Available: https://arxiv.org/abs/1812.05905