Chair of Space Mobility and Propulsion
Group 4
Technical University of Munich

Full Throttle - Reinforcement Learning - Report Part 2

Practical Course: Control and Simulation of Rocket-Hopper Demonstrator

Anibal Guerrero Hernandez, Hannes Kiefer, and Sven Julius Steinert

October 29, 2024

1 Introduction

The following report presents an overview of the results of the practical course ”Control and Simulation of a Rocket-Hopper-Demonstrator”.

The course tasks include creating and validating a fluid simulation of the propulsion system, developing a control algorithm for the main valve, and verifying models on the hardware demonstrator. The Hopper is comprised of a platform attached to a vertical track with locked horizontal and rotational degrees of freedom but can freely change its vertical position. It houses a cold gas nitrogen propulsion system supplied by an external tank connected via a flexible hose. Control is achieved by adjusting the entry pressure of the Laval nozzle via a proportional valve with an internal PID controller. The course emphasizes selecting and creating an appropriate controller, where we decide for an experimental algorithm, to use a reinforcement learning (RL) model like DDPG and TD3.

2 Literature

2.1 Reinforcement Learning Model

2.1.1 Model-Free Algorithms

In reinforcement learning (*RL*), *Model-Free RL Algorithms* don’t model the environment’s dynamics but focus on learning directly from interactions. They optimize policies or value functions through trial and error, adapting strategies based on observed outcomes. This approach is useful for complex environments where accurate modeling is challenging.

Model-Free RL Algorithms concentrate on learning directly from interactions, optimizing policies or value functions without explicitly modeling the environment’s dynamics. They rely on trial and error, adapting strategies based on observed outcomes, making them suitable for complex environments. Examples include *REINFORCE*, *TRPO*, *Q-Learning*, and *SARSA*, offering flexibility and adaptability in various real-world applications.

2.1.2 Algorithm Comparison

Two algorithms, *DDPG* and *TD3*, are chosen for comparison. *DDPG* (Deep Deterministic Policy Gradient) is tailored for continuous action spaces, employing an actor-critic architecture that blends policy-based and value-based methods. On the other hand, *TD3* (Twin Delayed DDPG) extends *DDPG*, enhancing stability with twin critics and delayed policy updates. Despite initially selecting *DDPG* for the *Rocket Hopper Demonstrator* due to its compatibility with continuous action spaces, proven track record in precision control, and simplicity for initial exploration, *TD3* was eventually implemented. As anticipated, *TD3* yielded superior results, confirming its effectiveness in a more stable model, and an enhanced performance. This successful implementation underscores the potential for adopting more advanced algorithms to further refine the model’s performance.

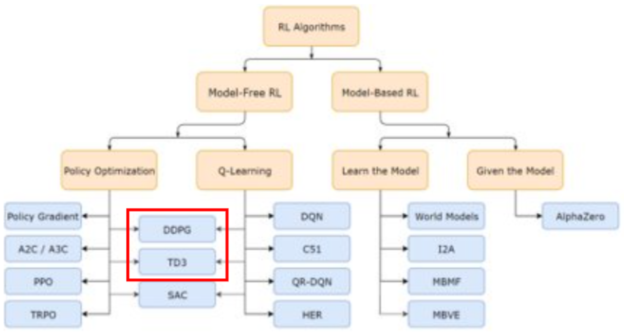


Figure 1 Classification of Reinforcement Learning Algorithms, boxing DDPG & TD3 as algorithms of interest.

2.1.3 DDPG

DDPG, an extension of Deterministic Policy Gradient (*DPG*) algorithms, integrates policy-based and value-based methods using two deep neural networks (*DNNs*) - actor and critic [1][2]. *DDPG* algorithms fall into the category of model-free, deterministic strategy algorithms. Originally developed by *Lillicrap et al.* [3], it has found applications in various research scenarios, though often with modified versions to address issues like falling into local optima and error fluctuations [4]. Strategies like *TD3* aims to enhance *DDPG*’s accuracy and stability, though their implementation complexity may limit adoption in some studies.

2.1.4 TD3

TD3 (Twin Delayed Deep Deterministic Policy Gradient) enhances the *DDPG* algorithm by introducing twin critics to mitigate overestimation bias in *Q-value* estimates. This approach involves training two separate critic networks and selecting the minimum Q-value estimate, reducing variance and stabilizing training. Additionally, TD3 implements a delayed policy update mechanism, updating the policy (actor) network less frequently than the critic networks. This decouples learning rates, preventing noisy critic estimates from affecting policy updates and enhancing stability [5].

Furthermore, TD3 employs target policy smoothing, adding noise to target actions during training to smooth Q-value estimates and reduce variance in policy updates. This regularization technique improves robustness and prevents policy exploitation of inaccuracies in Q-values. Compared to DDPG, TD3 demonstrates improved sample efficiency, stability, and robustness, justifying its computational investment, implementation complexity, and training time [5].

3 Method

3.1 Physical Model

3.1.1 General Description

As illustrated in Figure 2, the *Rocket Hopper Demonstrator* is equipped with a high-pressure N_2 gas tank pressurized at 300 bars serving as the propellant source. To control and optimize the flow, a pressure regulator is implemented to regulate the gas pressure to 11 bars. The regulated gas then flows through a flexible hose, treated as a series of straight segments for simplicity, leading to the fitting section, which is modeled as a resistance. The pressure regulated by the valve is part of the control strategy. The valve is placed before the nozzle and controls the mass flow that goes into the nozzle, generating the thrust required for the *Rocket Hopper Demonstrator* to lift off.

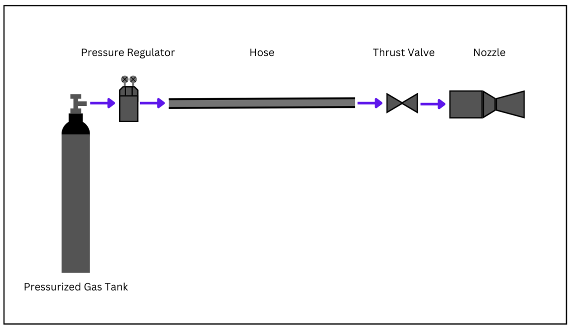


Figure 2 General physical model of the *Rocket Hopper Demonstrator*.

3.1.2 Assumptions of the Model

The elongated hose is divided into smaller segments, which are assumed to be perfectly straight. This allows the assumption that the pressure losses, that incur from the curvature of the original hose are negligible.

The inclusion of safety and drain valves is necessary as part of the safety measures. However, the pressure losses for these components are assumed to be negligible. Furthermore, due to the high flow speeds in the system and the short amount of time the nitrogen spends in the hose, the heat exchange with the environment is not considered. Additionally, isentropic conditions are assumed and no pressure losses through imperfect connections between the components occur.

3.2 Mathematical Model

The following section describes how the physical model presented in the previous paragraph can be adapted into a mathematical model. It discusses the formulas that are later converted into code and explains the reasons why they can be deemed applicable to describe the system's components.

3.2.1 Fluid Simulation

Boundary Conditions Before the model for the system itself can be created, the boundary conditions need to be defined. As mentioned in the task description, the regulator sets the pressure at the beginning of the hose to 11 bar. On the opposing side of the system, the nozzle exit, the boundary condition is defined by the atmospheric conditions at the test location. Here, the pressure is assumed to be 1 bar at a temperature of 293K. Due to the large volume of the tank and the hangar the tests are conducted in, it is assumed, that these conditions are constant and not affected by the launch of the hopper.

Hose To achieve accurate control over the resolution of the calculations for the hose, this component is split into a finite number of individual sections. The number of sections can be set to the desired amount with the help of a variable in the Python code. Each section of the hose is represented by two separate equations in the mathematical model. First, the time derivative of the pressure in each pipe section is obtained by modeling the pipe sections as a capacity. For element i this leads to the following differential equation:

$$\frac{\dot{p}_i}{dt} = \frac{V\rho_i}{K} \cdot (\dot{m}_{i-1} - \dot{m}_i) \qquad (1)$$

In the next step, the time derivative of the massflow rate is calculated with the help of the inductor equation for each section:

$$\frac{\dot{m}_i}{dt} = (p_i - p_{i+1} - \frac{\lambda}{2\rho_i A^2}\dot{m}_i^2 \cdot sign(\dot{m})) \cdot \frac{A}{L} \qquad (2)$$

Both differential equations are later iteratively solved with the help of the Runge-Kutta 4 integrator, which is explained in more detail in section 3.3.3. The required densities ρ_i in each pipe section are calculated by applying the isentropic gas relations to the current pressure, the current density, and the new pressure in each section [6]. For each pipe section, the friction coefficient λ is equal to 3, the length L is equal to $8m$ divided by the number of segments and the cross-sectional area of the pipe is $111.22mm^2$.

Fitting and Main Valve Both, the fitting before the main valve, and the main valve itself are modeled with the same approach. While the derivative of the pressure in the valve and the pipe is calculated analogously to the ones of the hose sections with equation (1), the massflow rate is directly calculated by modeling the resistance that the flow encounters while passing through these components. This value is described by the following correlation:

$$\dot{m}_1 = \sqrt{(p_1 - p_2) \cdot \frac{2\rho_1 A^2}{\lambda \cdot sign(\dot{m}_1)}} \qquad (3)$$

Analogously to the provided example, the λ of both components is assumed to be 100 times as large as the one of the hose sections. Furthermore, the dimensions of the fitting were taken from the task description where they are listed as 49.9mm length and 19.3mm diameter. Since the task description does not provide any information about the inner dimensions of the valve, they were assumed to be equal to the ones of the fitting.

Nozzle The fluid behavior in the nozzle itself is modeled in two different sections. First, the chamber area before the throat is modeled as a capacity. Similarly to the previous sections, equation (1) is used to calculate the time derivative of the pressure in the chamber. The volume of the chamber was provided as $V_{chamber} = 2ml$. The massflow which leaves the chamber on the other hand can be calculated with the help of the Laval nozzle equation (7), which will be explained further in section 3.2.4.

Results The pressures and massflows in and through each of the components after one second simulation time are shown in figure 3. As the graph

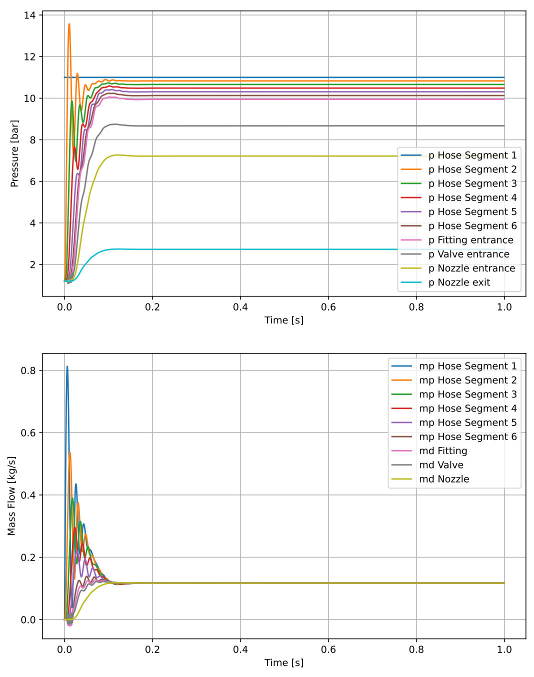


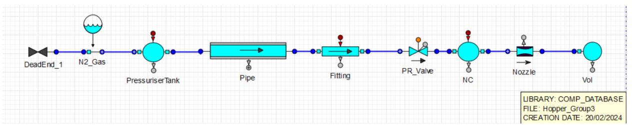
Figure 3 Pressure curve and mass flow from the fluid simulation

shows, the system reaches a near-static state after approximately 0.15 sec. At this point, the pressure in the chamber of the nozzle reaches approximately 7.458 bar and the mass flows stabilize at $0.196kg/s$. Figure 3 displays the system for six simulated hose segments. In this figure, strong oscillations in the pressure and mass flow values can be seen before they stabilize around the aforementioned values. Appendix A.1 shows the same plot for five, ten, and twenty segments. Even though the readability of the plots with high segment values is reduced, it is still obvious, that such plots display considerably smoother pressure and mass flow curves. One likely explanation for this phenomenon is the fact that the values for the density are considered to be constant in each section of the hose - a circumstance that does not depict reality accurately. It leads to sharp pressure changes across the segment borders and can negatively influence the behavior of the differential equation system.

3.2.2 EcosimPro Validation

To validate the fluid simulation tool, EcosimPro was utilized. EcosimPro is a first-class simulation tool for modeling continuous-discrete systems. It is the official tool of the European Space Agency in multiple fields.

System Architecture Ecosim employs a robust system architecture that allows for comprehensive fluid dynamics simulations.



LIBRARY: COMP_DATABASE
FILE: Hopper_Group3
CREATION DATE: 20/02/2024

Figure 4 System architecture of the modeled Rocket Hopper Demonstrator.

The real Nitrogen gas is introduced with the *N2_Gas* and is stored in the *Pressuriser Tank* at $11bar$. Then, the resistive *Pipe* acts as the flexible hose which allows the Nitrogen to be transported from the tanks to the *Rocket Hopper Demonstrator*. The length is set to $8m$ and the inner diameter to $19.3mm$. As an interface between the pressure regulator valve (*PR_Valve*) and the flexible hose, a *Fitting* is introduced. This component has a length of $49mm$ and a diameter of $11.9mm$. The *PR_Valve* is set to open always. Then, the fluid enters the nozzle chamber, *NC*, preceding the *Nozzle* where the thrust is generated. The throat area of the nozzle is calculated from the $9mm$ diameter condition. The final *Vol* is used to simulate the environment, with a volume of 100000.00, the maximum allowed by Ecosim.

Results and Comparison with Fluid Simulation Analogous to the findings obtained in Figure 3 from the fluid simulation, Ecosim yields similar results. These are represented in Figure 14 and Figure 13.

The pressure evolution in Figure 13 indicates slightly lower pressures at the nozzle entrance in Ecosim compared to the fluid simulation. At the nozzle entrance, the fluid simulation indicates pressures of around $7bar$, while Ecosim shows slightly lower pressures, averaging below $7bar$. Although the values are very close, it's essential to consider Ecosim as the ground truth due to the instabilities observed in the integration process, represented as spikes in Figure 3. This is likely caused by the use of the Runge-Kutta 4 integrator. However, these errors do not significantly affect the overall trend..

In terms of mass flow evolution (see Figure 14), Ecosim reports a smoother transient regime compared to the fluid simulation, with a mass flow of approximately $0.92kg/s$ versus over $0.1kg/s$ predicted by the fluid simulation.

Furthermore, a notable difference lies in the duration of the transient regime until convergence. The fluid simulation typically takes around 0.1 seconds to converge, while Ecosim converges in a shorter transient of only 0.03 seconds. This discrepancy may stem from the improved numerical methods and integration error handling in Ecosim, as well as a reduced time step size, contributing to faster convergence.

3.2.3 Main Control Valve Behavior

Since the remote-controlled main valve requires a small amount of time to open and close, an appropriate mathematical model is necessary to replicate this latency. As a first starting point, the transfer function of the valve was supplied to describe its behavior:

$$\frac{p_{actual}}{p_{set}} = \frac{1}{1 + 0.2 \cdot s} \tag{4}$$

When examining this formula, it becomes evident that this behavior is equal to that of a PT1-bit with a time constant of $T = 0.2$. To replicate the conditions in the valve, the control input p_{set} is interpreted as a σ step function with p_{set} as its scaling factor K. In the frequency domain, a PT1-bit with such an activation function can be simulated with the following equation:

$$Y(s) = \frac{1}{Ts+1} \cdot \frac{K}{s} + \frac{Ty(0)}{Ts+1} \tag{5}$$

To transition back to the time domain, an inverse Laplace transformation is conducted, and the previously defined model resulted in the following explicit equation:

$$p_{actual}(t) = p_{set} \cdot (1 - e^{-\frac{t}{0.2}}) + p_{current} \cdot e^{-\frac{t}{0.2}} \tag{6}$$

Equation 6 is then used to directly describe the relation between the set pressure, the current absolute pressure, and the absolute pressure at the time of the next control input. Since the input frequency is defined as 60Hz, the time t which has passed since the last control input is set to a constant $\frac{1}{60}s$. The outcome of the application of these equations to a real signal can be seen in figure 5. There, a random, noisy input signal

p_{set} is sent to the valve controller. p_{actual} shows the resulting absolute pressure, which is consequently sent to the nozzle.

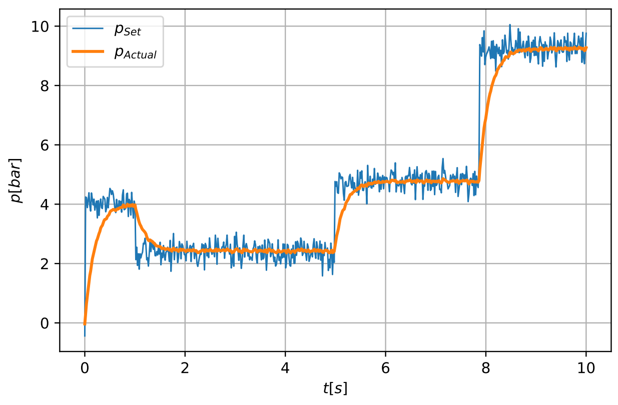


Figure 5 Valve for an exemplary noisy signal

Meanwhile, the problem could also be solved by utilizing the existing numeric integrator. A PT1-bit can also be modeled by an ODE, which could then be approximated with the help of the already implemented Runge-Kutta 4 method. However, since the analytical solution is available and could deliver more accurate results while being more efficient when it comes to computational power, it is decided to use the aforementioned strategy.

3.2.4 Nozzle

To determine the thrust, classic nozzle flow equations are used [7]. Besides the geometric dimensions, the exit pressure and the temperature from the previous section are known. The first value that needs to be determined is the massflow $\dot{m}$ through the nozzle. For this purpose, the following formula is utilized:

$$\dot{m} = \frac{A_t \cdot p_t}{\sqrt{\frac{T_t \cdot R}{\gamma}}} \cdot \left(\frac{\gamma+1}{2}\right)^{\frac{-\gamma-1}{2(\gamma-1)}} \tag{7}$$

Equation 7 creates a relation between the throat area A_t, the specific heat capacity of nitrogen γ, the specific gas constant of nitrogen R, as well as the total pressure p_t and total temperature T_t. Since the conditions in the nozzle are assumed to be a chocked flow, the Mach number is equal to 1. Furthermore, the remaining values except the total temperature, are known from the previous sections or the task description. In the next step, the exit Mach number M_e is calculated. The calculation is dependent on the geometric dimensions of the nozzle, as well as on the specific heat capacity of the nitrogen. Formula 8 is used to simulate the connection between the diameter ratios of the throat and exit areas, and the Mach number [7]:

$$\frac{A_e}{A_t} = \left(\frac{\gamma+1}{2}\right)^{\frac{-\gamma-1}{2(\gamma-1)}} \frac{\left(1+\frac{\gamma-1}{2}\cdot M_e^2\right)^{\frac{\gamma+1}{2(\gamma-1)}}}{M_e} \tag{8}$$

While the formula describes the connection accurately, it cannot be solved for M_e directly. Thus, a Newton algorithm with the following format is implemented to iteratively determine the value.

$$M_{e,i+1} = M_{e,i} - \frac{f(M_{e,i})}{f'(M_{e,i})} \tag{9}$$

Depending on the initial value $M_{e,0}$, this algorithm delivers two different Mach numbers. The lower result is $M_E = 0.582$, leading to a subsonic flow. Since supersonic behavior was assumed, the higher result, $M_e = 1.548$ is chosen as the more accurate one for the process. With the exit Mach number calculated, the following formulas are used to find the exit pressure p_e and the exit temperature T_e:

$$p_e = p_t \cdot \left(1 + \frac{\gamma-1}{2} \cdot M_e^2\right)^{\frac{-\gamma}{\gamma-1}} \tag{10}$$

$$T_e = T_t \cdot \left(1 + \frac{\gamma-1}{2} \cdot M_e^2\right)^{-1} \tag{11}$$

Both formulas use a combination of the exit Mach number and the specific heat capacity to calculate the ratio between the total value and the exit value. While the total pressure, p_t, is assumed to be the absolute pressure that was set by the main valve, the total temperature, T_t, is still unknown. However, once all variables are combined in Equation 12 and Equation 13, it becomes evident that T_t is canceled out when the mass flow is multiplied by the exit velocity. Thus, this parameter does not affect the thrust generation and does not need to be determined. As the last variable in the process, the flow velocity in the exit area needed to be determined. For this purpose, the exit Mach number was multiplied by the speed of sound, represented by the root of the product of the specific gas constant and the specific heat capacity of nitrogen, as well as the exit temperature:

$$v_e = M_e\sqrt{\gamma R T_e} \tag{12}$$

Finally, the calculated values are combined with the ambient pressure p_0 to calculate the generated thrust F_T:

$$F_T = \dot{m}V_e + (p_e - p_0)A_e \tag{13}$$

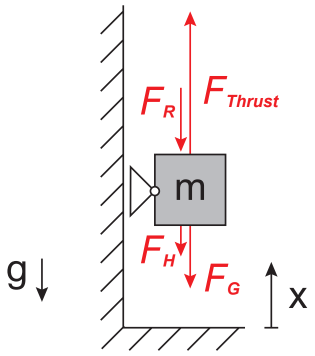


Figure 6 Physical model of the Rocket Hopper

3.2.5 Kinematics

The motion of the *Hopper* is described by the Newtonian motion of $F = m \cdot a$ which expands to all present forces on the *Hopper*, being the thrust from the nozzle, F_T, the gravitational force, F_G, the force of the hose, F_H, and the resistance, F_R.

$$\ddot{x} = \frac{1}{m} \cdot (F_T - F_G - F_H + F_R) \tag{14}$$

Gravitational Force Modelling F_G The mass of the *Hopper* and the hose are combined into a constant gravitational force. The gravitational acceleration is set to standard gravity $g_0 = 9.80665 m/s^2$ as the experiment is carried out on Earth's surface level.

$$F_G = m \cdot g = (3.5 + 1)\ kg \cdot g_0 = 44.13\ N \tag{15}$$

Hose Force Modelling F_H As the *Hopper* rises in the vertical position, the load increases from the additional hose mass. This force is modeled through a spring behavior with constant $k_{Hose} = 6N/m$.

$$F_H(x) = k_{Hose} \cdot x \tag{16}$$

Resistance Force Modelling F_R The resistance is generally modeled by a constant force against the velocity of the *Hopper*. However, to avoid high-frequency alternating forces around $\dot{x} = 0$, a transitioning section is defined where the force is changed linearly, which removes the step function at zero.

$$F_R(\dot{x}) = \begin{cases} sign(\dot{x}) \cdot 10N, & \text{for } |\dot{x}| > 0.01\frac{m}{s} \\ \frac{|\dot{x}|}{0.01\frac{m}{s}} \cdot sign(\dot{x}) \cdot 10N, & \text{for } |\dot{x}| \le 0.01\frac{m}{s} \end{cases} \tag{17}$$

The threshold of velocity to switch between those two sections is set to $|\dot{x}| \le 0.01\frac{m}{s}$, which aligns with the measuring resolution on the acceleration of $0.01\frac{m}{s^2}$, when seen per second. This piecewise-defined function is plotted in Figure 15.

Thrust Force Modelling F_T Connecting to Equation 13, the thrust is first derived from the nozzle flow equations. To increase simulation speed for Reinforcement Learning, a fitting is performed on the transfer function from p_{valve} in *bar* to F_T in Newton, which requires no loss in accuracy due to the linear characteristics of this problem. To avoid negative values for F_T, a maximum comparison is applied.

$$\begin{gathered} F_T(p_{valve}[bar]) = \max\{0, m \cdot p_{valve} + n\}\ [N] \\ \text{with} \quad m = 10.600789,\ n = -9.503318 \end{gathered} \tag{18}$$

A plot of the resulting thrust modeling can be found in Appendix 16.

3.3 Numerical Model

3.3.1 ODE

The differential equation of motion is defined by the ODE state vector y:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{19}$$

and its time derivative $\dot{y}$:

$$\dot{y} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} y_2 \\ \frac{1}{m} \cdot (F_T - F_G - F_H(y_1) - F_R(y_2)) \end{bmatrix} \tag{20}$$

Which together define position x, velocity $\dot{x}$, and acceleration $\ddot{x}$ of the *Hopper*.

3.3.2 Restrictions

To prevent the *Hopper* from falling into a negative position during the beginning of the simulation when the weight is supported by the ground, a restriction was made to the acceleration $\ddot{x}$ which sets it to zero if the position is at zero or below and the acceleration from the ODE would be negative.

$$\ddot{x} = \begin{cases} 0\ \frac{m}{s^2}, & \text{if } x \le 0\ m \text{ and } \ddot{x}_{ode} < 0\ \frac{m}{s^2} \\ \ddot{x}_{ode}, & \text{else} \end{cases} \tag{21}$$

This restriction prevents initial dropping into the negative realm as the thrust is built up, but not if velocity has already built up downwards. This scenario would then correspond with a crash, which means the end of a meaningful simulation anyway.

3.3.3 Integrator

To solve the differential equation of motion which describes the system's behavior, a numerical integration method is utilized. The Runge-Kutta method is selected for this purpose, as it offers greater performance for the ODE system than explicit Euler through its higher order of degree. In particular, explicit Runge-Kutta of 4^{th} order is used.

The selection of the integration stepsize h is influenced by the dynamics of the resistance force modeling, as h needs to be small enough to land on the linear slope around zero velocity, otherwise alternating forces and instability are the result. Using a constant input of $p_{set} = 6.5\ bar$ where the *Hopper* will reach a steady position, the accelerations $\ddot{x}$ are plotted for multiple integration stepsizes h in Figure 7.

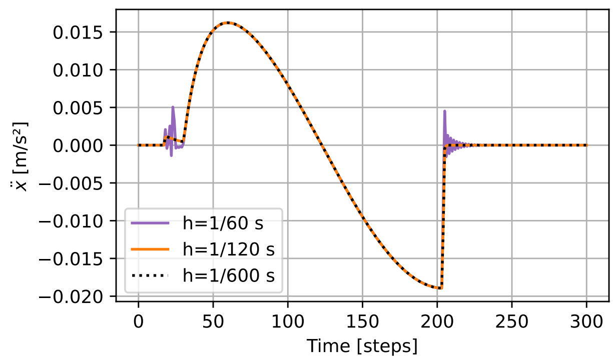


Figure 7 Acceleration over multiple integration stepsizes

Instabilities are visible for $h = \frac{1}{60}\ s$. Otherwise, the values are not changing from a reduced integration stepsize from $\frac{1}{120}\ s$ to $\frac{1}{600}\ s$ which proves convergence and the integration stepsize of $h = \frac{1}{120}\ s$ is selected.

3.3.4 Simulation

The interaction strategy between the simulation and the controller is driven by the update frequency of the sensors or the *Rocket Hopper*. Hereby all sensors $(x, \ddot{x}, p)$ share a sampling rate of 60 Hz, as well as the actor p_{set} with 60 Hz as the maximum update frequency. This leads to the decision to also base the interaction of the controller on 60 Hz, which has the benefit of processing the absolute newest available data for the controller input.

Starting the controller with a state and delivering an action in the next update implies, that in the real world, the controller has roughly 16 ms time to compute the action.

3.4 Controller

3.4.1 General Description of Architecture

The TD3 actor-network is composed of two linear (dense) hidden layers of 16 neurons whereas the critic network shares the same structure but with 27 neurons. The reward function f_r is a Gaussian distribution depending on the control error Δx as stated in Equation 22 with $\sigma = 0.5$.

$$f_r(\Delta x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{1}{2}\left[\frac{\Delta x}{\sigma}\right]^2\right) \tag{22}$$

Exploration vs Exploitation Achieving high rewards requires a balance between exploration and exploitation. The agent must exploit learned knowledge from the environment while exploring new actions for future strategies [8]. Various strategies, such as those compared by Plappert [9] and Lilicrap [3], exist for continuous action spaces. The *Ornstein-Uhlenbeck Process* (OUP) noise is preferred for its smoothness in exploration, particularly suitable for continuous control tasks [3].

To incorporate exploration, OUP adds noise smoothly, as described by the stochastic differential equation:

$$dx_t = -\theta x_t dt + \sigma dW_t \tag{23}$$

where $\theta > 0$ and $\sigma > 0$ are parameters and W_t denotes the Wiener process.

Hyperparameter Settings Hyperparameters not only affect the reinforcement learning algorithm but also the exploration strategy and the environment. DDPG generally provides good results in continuous control problems [10] but is more sensitive to numerous hyperparameters than other algorithms [11].

These are grouped into DDPG, Exploration, and Environmental Hyperparameters.

3.4.2 Sensitivity Analysis

Performing sensitivity analysis in hyperparameter tuning involves systematically exploring how changes in these values impact the model's performance metrics. It provides valuable insights into the robustness of the model and helps identify optimal configurations for achieving the best results.

Hyperparameter Optimization Methods The suitable hyperparameters are problem-specific and the optimal hyperparameter combination is often non-intuitive. The widespread manual choice of hyperparameters therefore requires experience and is time-consuming [12].

Several strategies are presented that can be followed depending on the approach. There are Model-free Approaches and Model-based Approaches, but the focus remains on the former.

- Grid Search - discrete, grid-shaped subset instead of the entire parameter space.
- Random Search - selects hyperparameters from the equally distributed search space.
- Optuna - employs advanced optimization techniques, such as Bayesian optimization, to adaptively explore and intensify the search in areas of the parameter space with high potential.

Bergstra and Bengio proves that the random choice of hyperparameter combinations is more efficient than searching a grid subset [13] Optuna however is better than a Random Search, which is an improvement from the initial DDPG approach and hyperparameter tuning. These approaches prevent the convergence into local optima but are extremely time-consuming due to the curse of dimensionality in large parameter spaces.

Random Search To conduct the sensitivity analysis, a random sample from a log-uniform distribution was generated for key hyperparameters, namely $learning\_rate\_actor$, $learning\_rate\_critic$, and $exploration\_noise$. The log-uniform distribution is chosen for its ability to explore different orders of magnitude efficiently. The range used to perform the random search is provided in Table 1.

Table 1 Ranges employed for Random Search

Hyperparameter	Range
Learning Rate (Actor)	$[0.0001, 0.0005]$
Learning Rate (Critic)	$[0.001, 0.05]$
Exploration Noise	$[0.1, 3]$

Random Search Plot Two graphical representations are provided to illustrate the outcomes:

Figure 8 displays multiple line plots, each depicting the evolution of episode rewards over time for specific hyperparameter settings. The x-axis represents the episode number, while the y-axis shows the corresponding reward achieved during training.

Observations from Figure 8 reveal three general behaviors. Line plots with steep rewards indicate highly effective configurations leading to rapid learning and substantial rewards. Line plots approaching zero suggest a bimodal distribution in the effectiveness of set configurations. Additionally, most line plots plateau after approximately 100 episodes, indicating diminishing returns from further training. Some configurations exhibit slower learning but lack significant reward improvement.

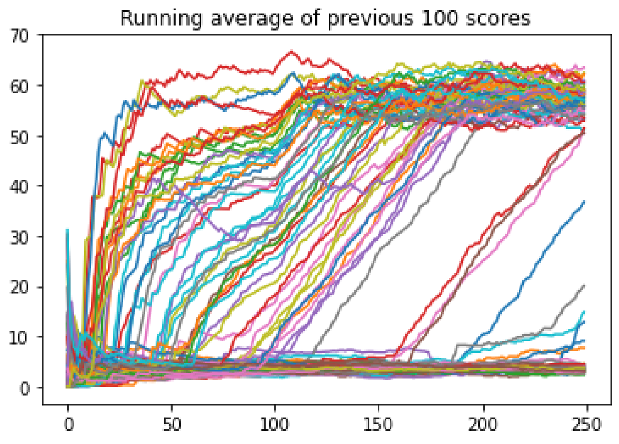


Figure 8 Random Search Results with 100 random log-uniform distributed samples.

Scatter Plot of Rewards The observed pattern in the scatter plot in Figure 9, where almost all configurations either obtain the maximum reward or almost zero, suggests a distinctive bimodal distribution in the effectiveness of the hyperparameter configuration. This proves the hypersensitivity of DDPG models to hyperparameter tuning. The bimodal behavior emphasizes the existence of configurations that lead to highly successful, near-optimal performance, and sub-optimal or negligible learning configurations.

Upon analyzing Figure 9, it becomes evident that certain hyperparameter configurations stand out as highly effective. A distinct cluster of configurations is observed and included in Table 2.

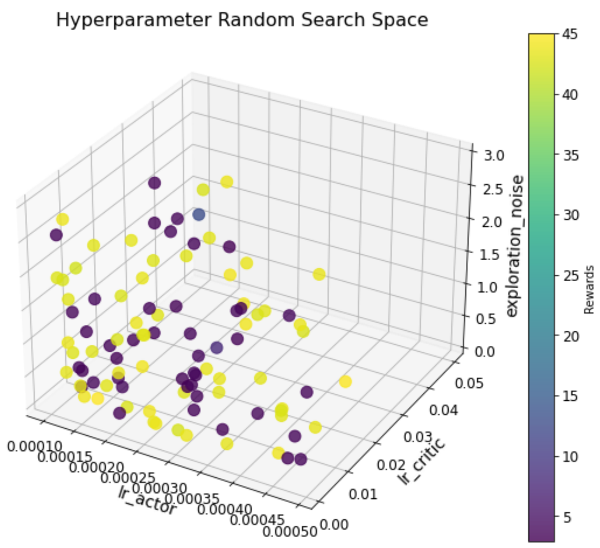


Figure 9 Scatter Plot of Rewards obtained for the 100 random log-uniform distributed samples.

Table 2 Hyperparameter Settings and Highest Reward with DDPG Random Search

Hyperparameter	**Value**
Learning Rate (Actor)	0.00024188
Learning Rate (Critic)	0.0001
Exploration Noise	0.28878
Highest Reward	**43.86**

3.4.3 TD3 Optimization with Optuna

After determining the optimal hyperparameter settings for DDPG with the random search, the settings were then transferred to TD3 and further hyperparameter tuning was performed with the Optuna framework. Optuna allows for a bigger number of hyperparameter tuning, and pruning, allowing for better computational resource allocation. Figure 19 demonstrates how different hyperparameters spread to find the highest total reward, while Figure 19 shows the importance of each hyperparameter. Certain hyperparameters show a local accumulation, i.e. gamma, whereas similar optima are found for diversified values in other hyperparameters, i.e. learning rates.

By adopting these settings for TD3, a baseline for its performance was established, evaluating its effectiveness compared to DDPG, and enabling a systematic comparison of the two algorithms in reinforcement learning tasks.

A final comparison between the different controllers implemented is collected in Table 4. Even though TD3

Table 3 Hyperparameter Settings and Highest Reward with TD3 Optuna

Hyperparameter		**Value**
Learning Rate	Actor	0.00086479
	Critic	0.03483899
Layer Size	Actor	16
	Critic	27
Tau		0.0066
Gamma		0.9867
Batch Size		42
Highest Reward		**141.35**

proves as an exponential improvement upon DDPG, it is still falling behind a PID controller.

Table 4 Reinforcement Learning Controller Comparison with PID

Reward	**Value**
DDPG	43.86
TD3	141.35
PID (Ground Truth)	157.6

3.4.4 Robustness

Robustness is crucial for evaluating control algorithms and deep learning models. According to Moos et al. [14], robustness in reinforcement learning (RL) refers to the ability to handle variations or uncertainties in the environment's dynamics. Training RL models with noise, such as Ornstein-Uhlenbeck Process (OUP), enhances robustness by allowing networks to adapt to noisy signals [14]. Moreover, proper selection of hyperparameters significantly impacts model robustness, with methods like random search and grid search proving effective [14]. Additionally, quick and clear training convergence serves as an indicator of robust systems, as observed in the DDPG network [14]. To further validate robustness, methods like the Monte-Carlo method can be employed [15].

3.4.5 Interfaces

As the interface to the *Hopper* includes raw integer values, the *uint12* range [0..4095] is mapped to [0..7] *bar* and reverse. Communication goes over pyserial by a baud rate of 115200 which transfers an utf-8 encoded string of a length around 33 (e.g. *"< 74282: 9.73: 1.21: 1980: 0: 0 >"*). This gives an estimation of 33 bytes per transfer which would need 2.3 ms in transfer time via the mentioned baud rate. Additionally, the shifted zero point in position is compensated by subtracting 0.11 m in altitude before it is handed to the controller.

4 Results

4.1 TD3 Controller

4.1.1 PID Baseline

A PID controller was created as a baseline for the example scenario to reach a static target $x_{target} = 2$ from $y = [0, 0]$. The chosen value $u(t)$ is defined by:

$$u(t) = K_\mathrm{p} \cdot e(t) + K_\mathrm{i} \cdot \int_0^t e(\tau)\mathrm{d}\tau + K_\mathrm{d} \cdot \frac{\mathrm{d}e(t)}{\mathrm{d}t} \quad (24)$$

Which is mapped to the action space of $p_{min} = 0$ to $p_{max} = 7$ by a sigmoid function $\sigma(x)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (25)$$

$$p_{set}(t) = \sigma(u(t)) \cdot (p_{max} - p_{min}) + p_{min} \quad (26)$$

Through manual adjustment, the constants $K_\mathrm{p} = 50$, $K_\mathrm{i} = 0.08$ and $K_\mathrm{d} = 1000$ have been determined, which gives a reward score over a simulation time of 5 s with reward evaluation every $\frac{1}{60}$ s. This reward score of 157.6 gives an orientation of what performance can be achieved, which sets the baseline for the TD3 controller in the benchmark scenario.

4.1.2 Results after Hyperparameter Tuning

After extended training with the selected hyperparameter values described in subsubsection 3.4.2, the reward score measured 142 in the benchmark scenario. This score still falls behind the PID baseline of 157.6 but still managed to achieve a decent control under simulation.
In Figure 24 can be seen the simulation of a target height of 2 m together with a descent maneuver. The linear decrease of the target height was implemented due to overshooting.

4.1.3 Real Time Capability

In the deployment stage the actor and critic are loaded from a file and run in forward mode. The time to compute an action is tested over 1 million actions (CPU: Intel i5-6600K) which resulted in a mean execution time of 0.19 ms. Together with the communication time of 2.3 ms to be 2.5 ms, it fits the 16.33 ms time window of a 60 Hz control with plenty of margin.

4.1.4 Test Campaign

During the test campaign, 4 runs are attempted, where different types of errors occurred. In run 1, due to a full throttle command, a comparison between a full throttle simulation can be drawn, which showed considerable differences (Figure 23). In run 2 and 4, the action space of the network was not mapped correctly, yet still some valve behaviour remains unexplained (Figure 25). In run 3, a constant resting on the initial position is recorded, which gives a great condition to measure the noise of the acceleration sensor. The following errors and learnings are summarized:

- Actions have been recreated from the recorded state instead of logged directly.
- Descent maneuver was triggered immediately.
- Wrong mapping of pressures between the integer value of 10 bar to the action space of the network of 7 bar.
- Maximum pressure, thrust, and acceleration have been overestimated in the simulation
- Acceleration noise ($\sigma = 0.67$ m/s^2 measured in run 3) is largely transferred into the action.
- Remaining unexplained lack of response of the valve under some conditions.

5 Conclusion

Only with the final model, stability is obtained in the simulation, however with considerable over- and under-shooting, but no positional oscillation. In the test campaign, stability could not be demonstrated. The concluded reasons for this are the wrong action space mapping and the highly oscillating commands to the valve, which are induced by the sensor noise. One solution would be to train the model without the information of the acceleration sensor, so a steady control set value is achieved.

During the development of the RL model, selecting the appropriate reward function has proven to be vital. As the control requirements have to be converted into feedback to the network, while still featuring a continuous landscape to ensure optimization conditions.

Advances are made by the transition to TD3, while still a hypersensitivity on the hyperparameter settings is present. Only under extended randomized search, a model is obtained that compares with 142 as reward score to the PID controller of 157.6.

Open Access The source code with all processing steps and the Python Notebooks can be found in the Git-Repository.

References

[1] Gabriel Cassimiro. A deep dive into actor-critic methods with the ddpg algorithm. *Geek Culture*, 14.4.2023.

[2] Nesma M. Ashraf, Reham R. Mostafa, Rasha H. Sakr, and M. Z. Rashad. Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm. *PLoS ONE*, 16(6):e0252754, 2021.

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning.

[4] Haifei Zhang, Jian Xu, Jian Zhang, and Quan Liu. Network architecture for optimizing deep deterministic policy gradient algorithms. *Computational Intelligence and Neuroscience*, 2022:1117781, 2022.

[5] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[6] Isentropic flow equations, 30.03.2024.

[7] Nancy Hall. Rocket thrust equations, 2021.

[8] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[9] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[10] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl ˆ2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[11] A Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR, 2018.

[12] Roman Liessner, Jakob Schmitt, Ansgar Dietermann, and Bernard Bäker. Hyperparameter optimization for deep reinforcement learning in vehicle energy management. In *ICAART (2)*, pages 134–144, 2019.

[13] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[14] Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315, 2022.

[15] Kimia Vahdat and Sara Shashaani. Robust output analysis with monte-carlo methodology.

A Appendix

A.1 Fluid Simulation

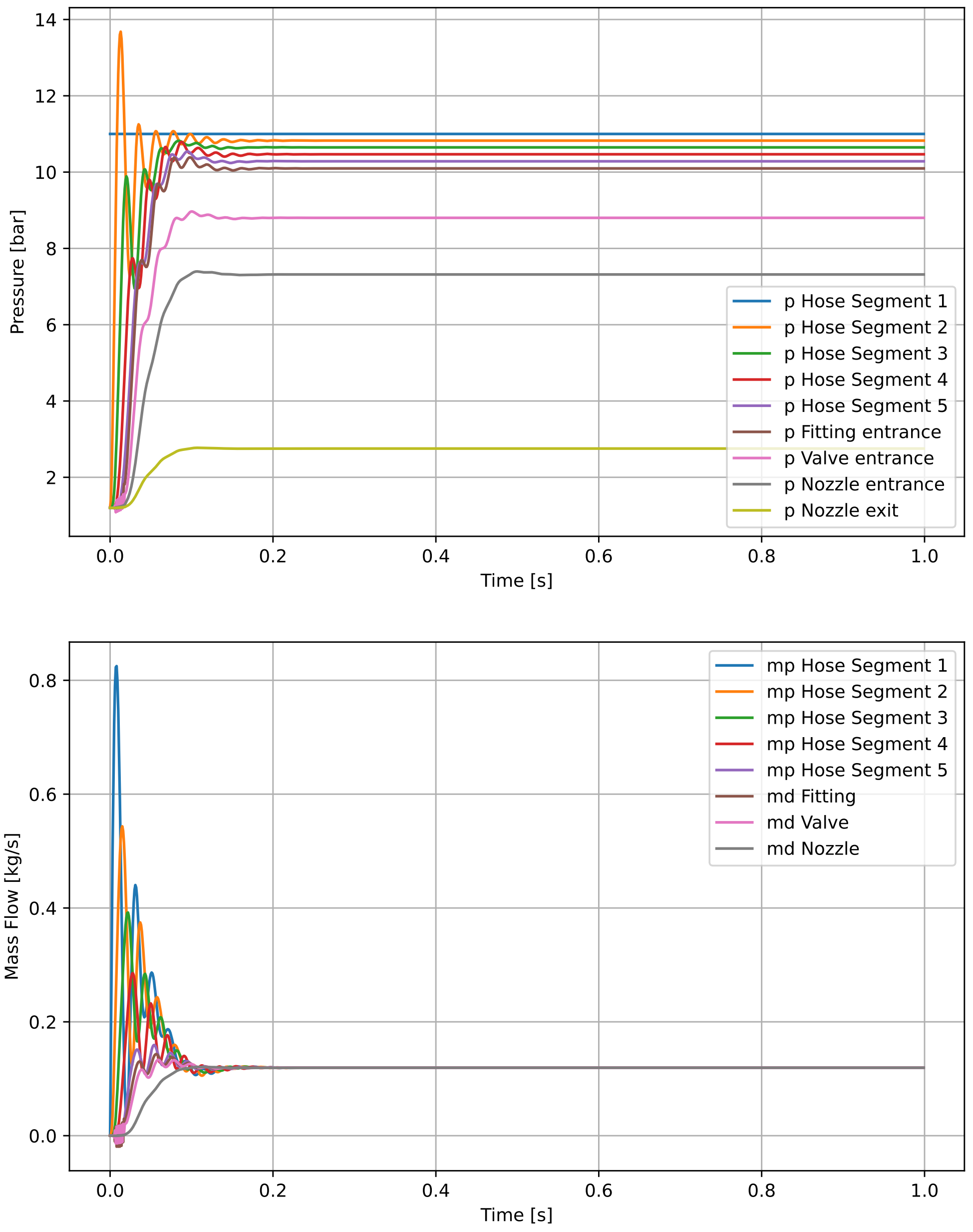


Figure 10 Pressure curve and mass flow from the fluid simulation with five hose segments

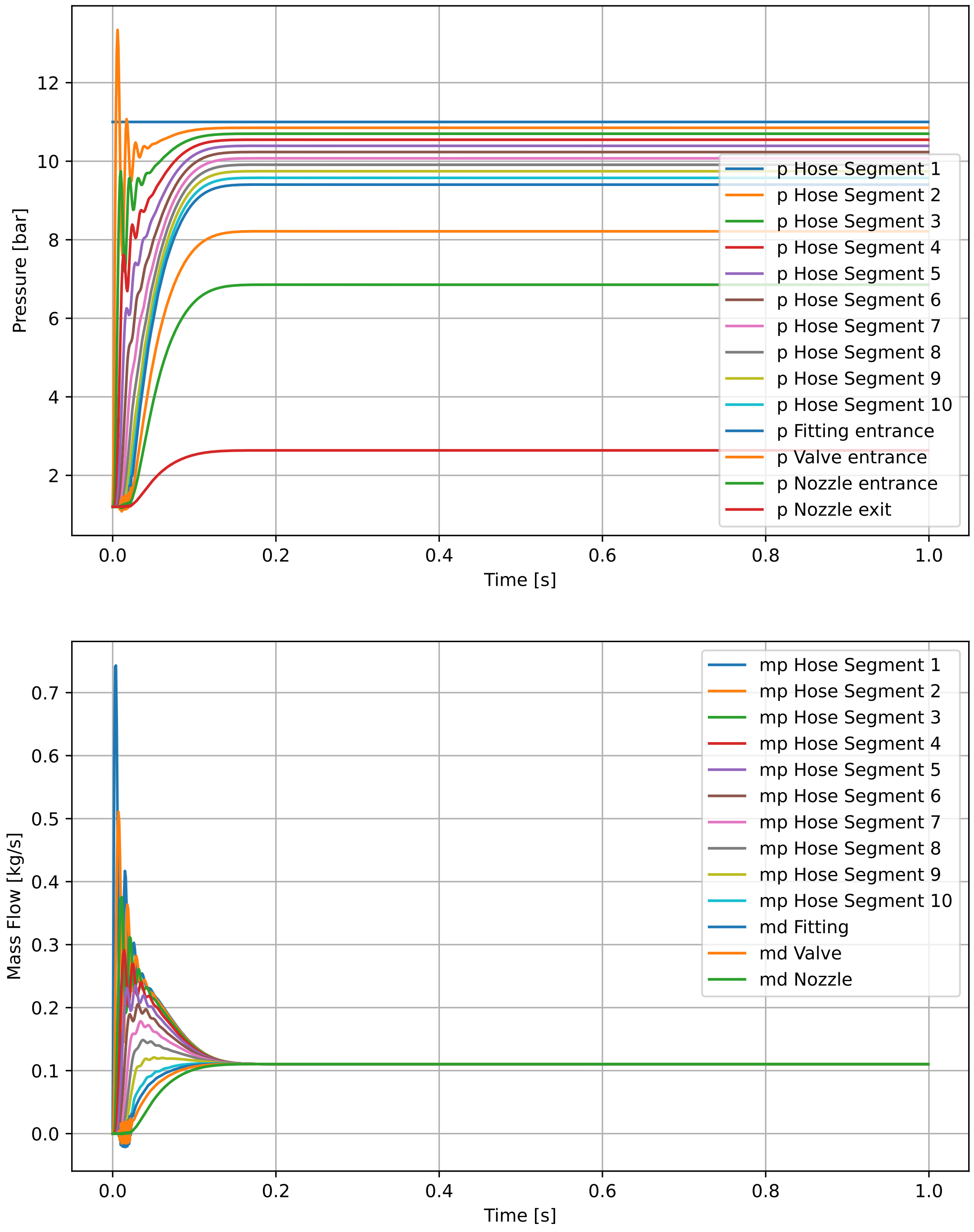
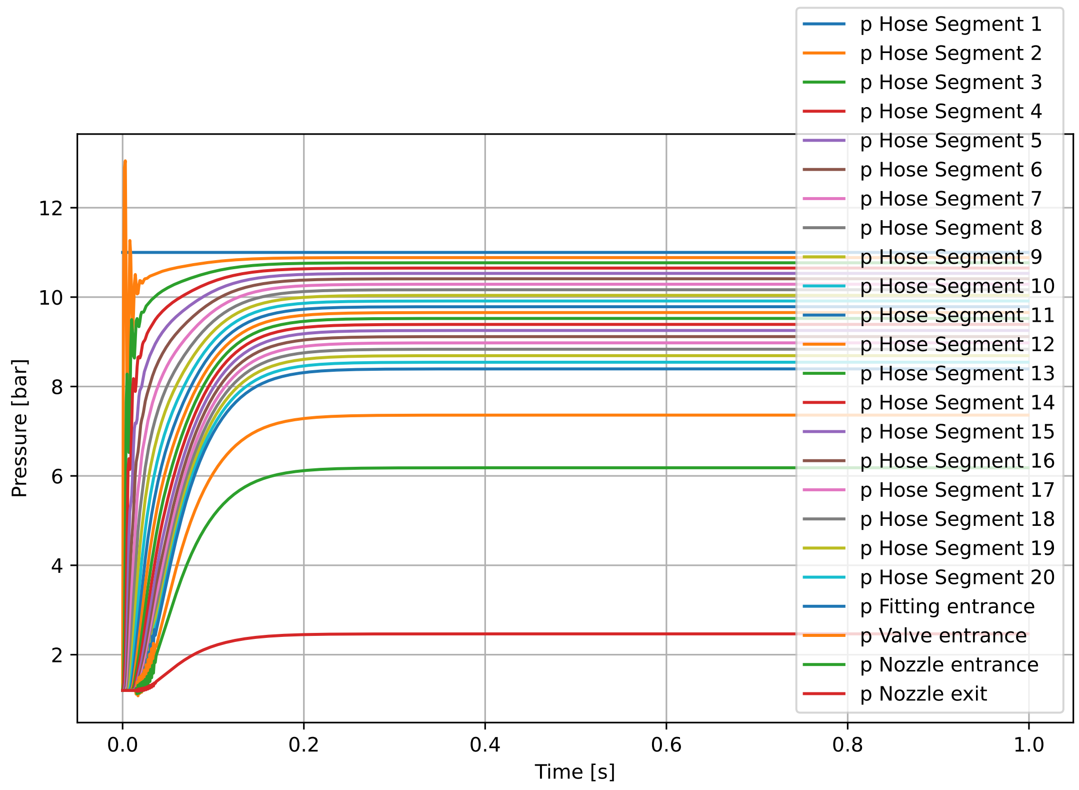


Figure 11 Pressure curve and mass flow from the fluid simulation ten five hose segments



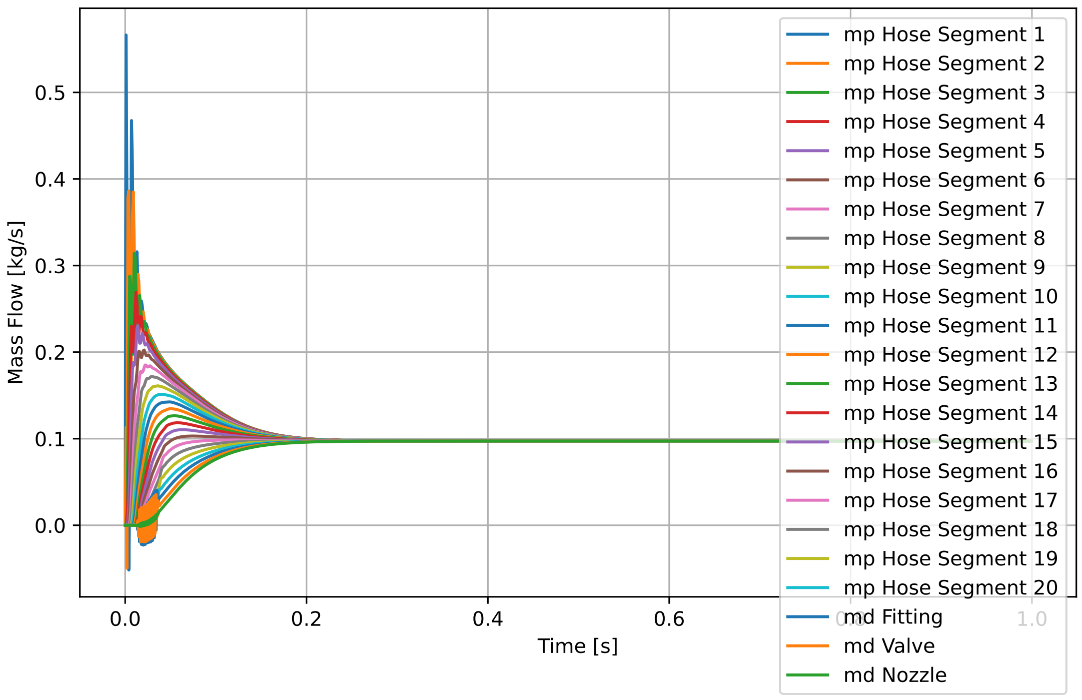


Figure 12 Pressure curve and mass flow from the fluid simulation with twenty hose segments

A.2 Ecosim Results

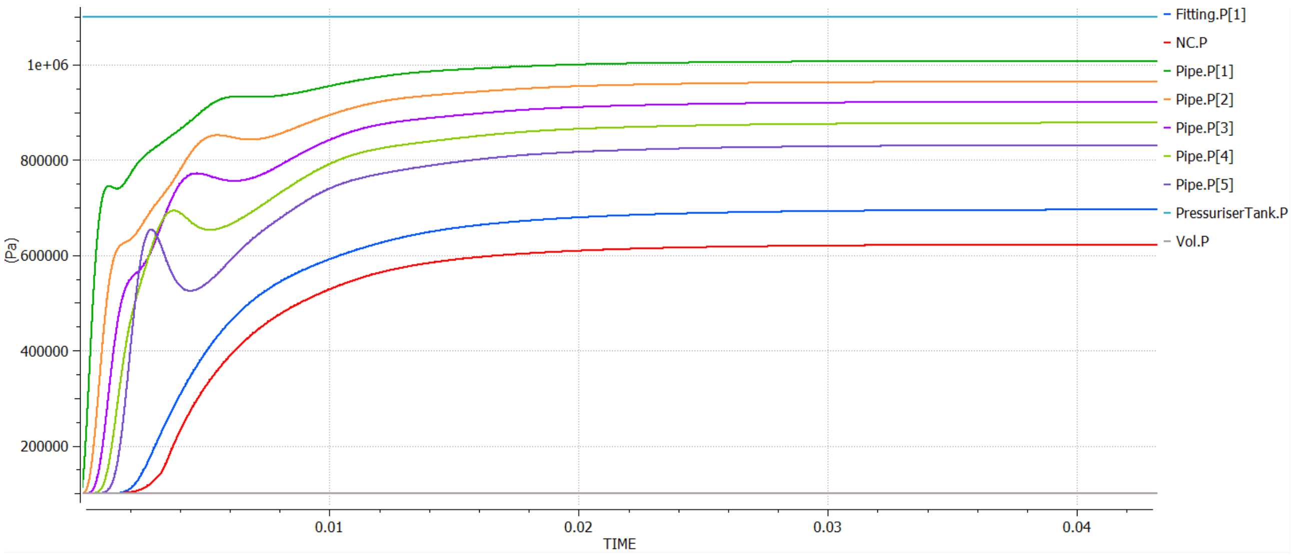


Figure 13 Pressure evolution through all components in the system provided for the Ecosim simulation of the Rocket Hopper Demonstrator.

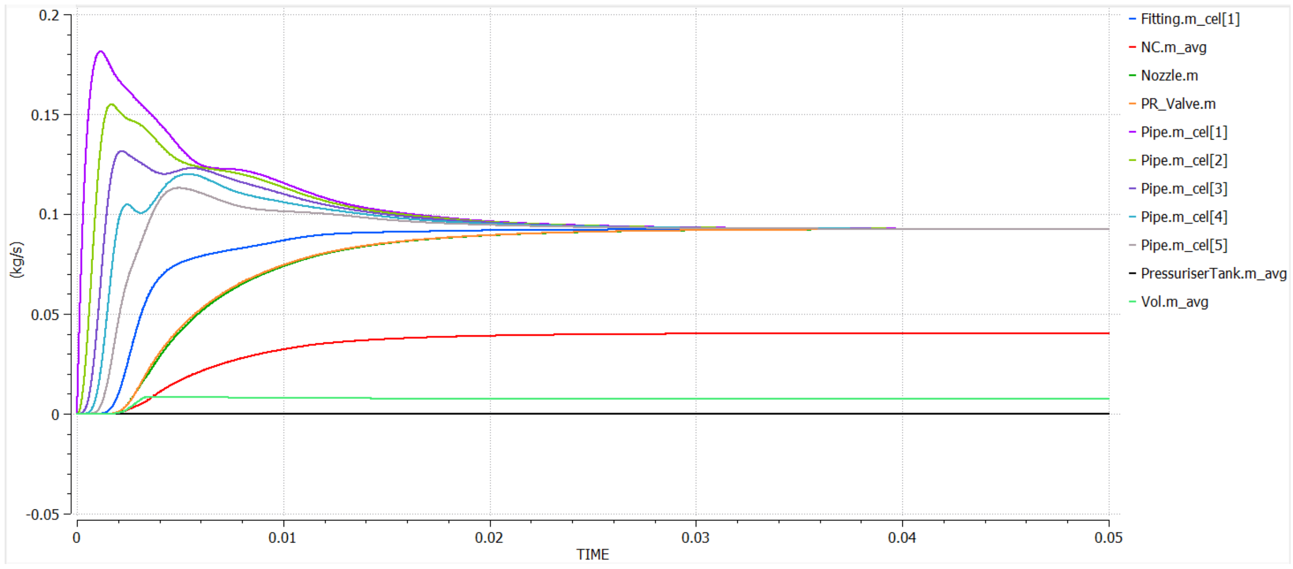


Figure 14 Mass Flow evolution through all components in the system provided for the Ecosim simulation of the Rocket Hopper Demonstrator.

A.3 Modelling

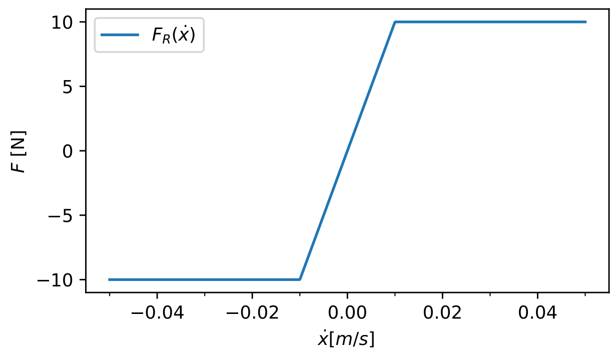


Figure 15 Resistance modelled $F_R(\dot{x})$ with smoothing

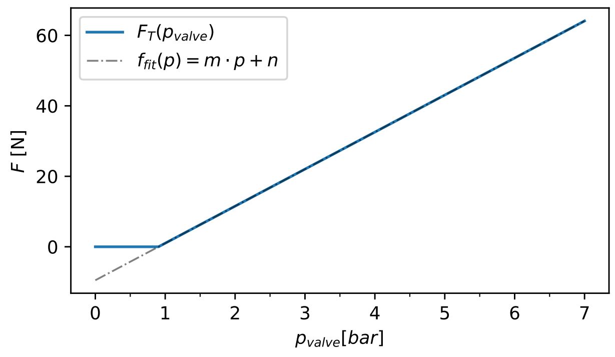


Figure 16 Thrust modeled $F_T(p_{valve})$ with restrictions

A.4 Simulation Interaction

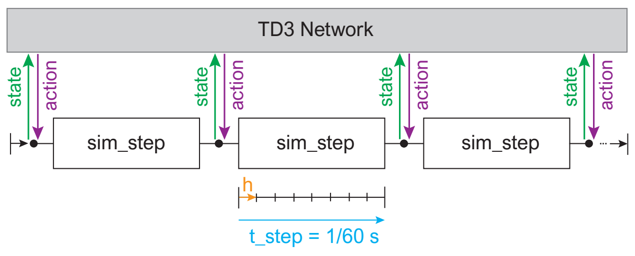


Figure 17 Simulation structure and controller interaction

A.5 Network Structure

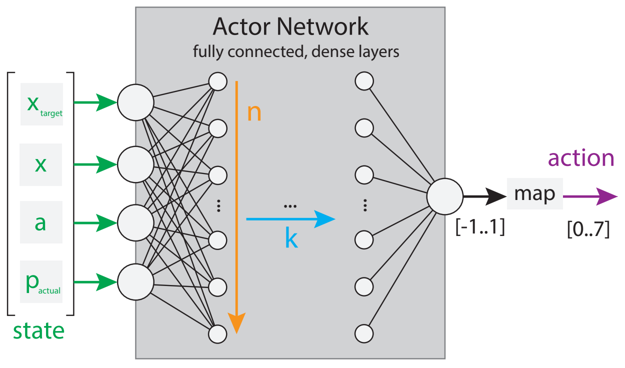


Figure 18 Internal structure of the actor with k hidden layers and n number of neurons. ($k = 2$, $n = 16$)

A.6 Optuna Results

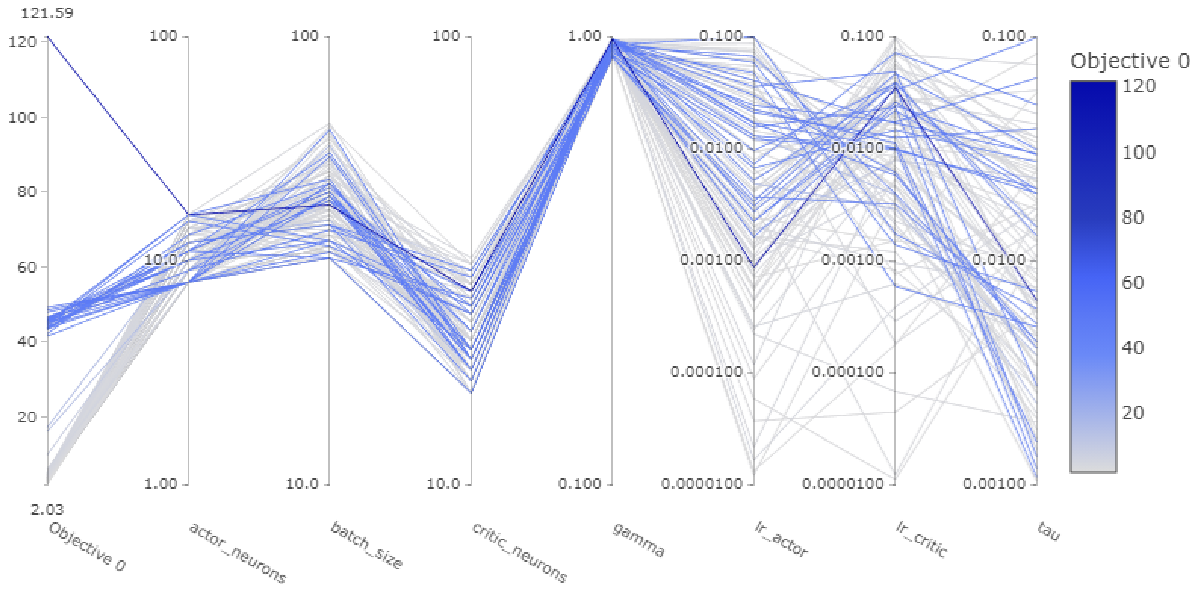


Figure 19 TD3 Hyperparameter optimization with Optuna.

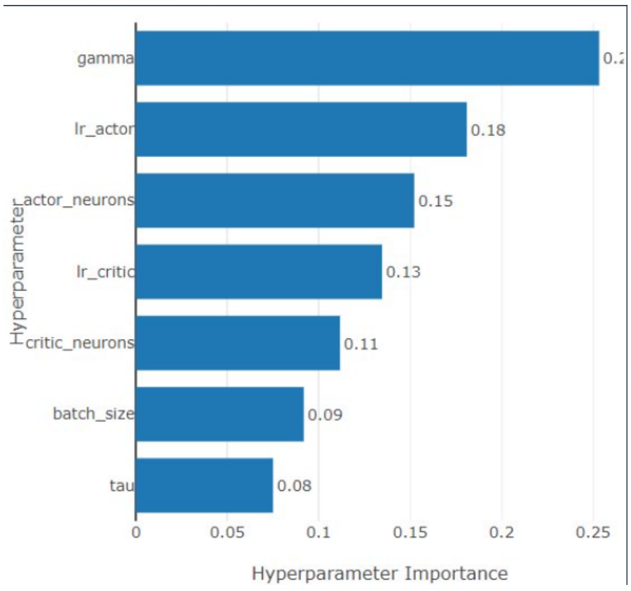


Figure 20 TD3 Hyperparameter importance.

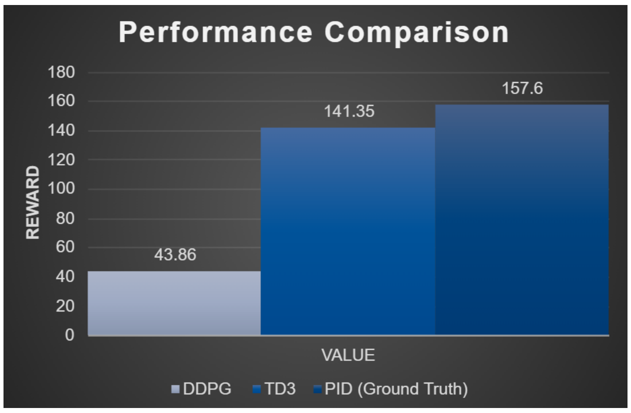


Figure 21 Total Reward Comparison between DDPG, TD3, and PID controller.

A.7 Simulation and Test Campaign Results

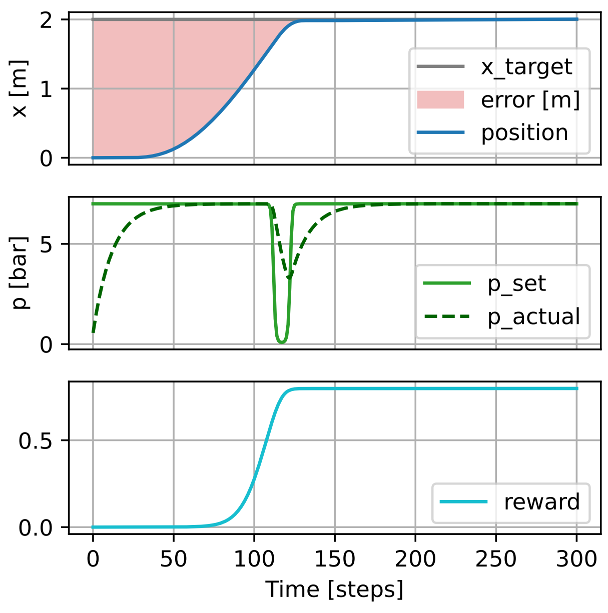


Figure 22 Simulation of the baseline PID controller

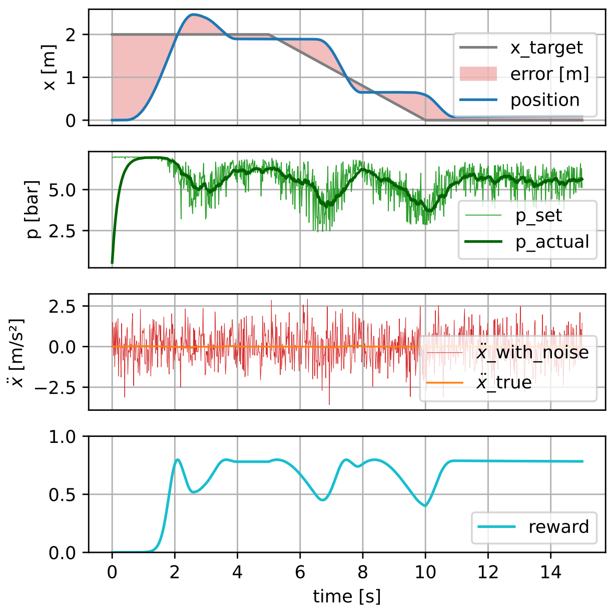


Figure 24 Simulation with descent maneuver of the final trained model

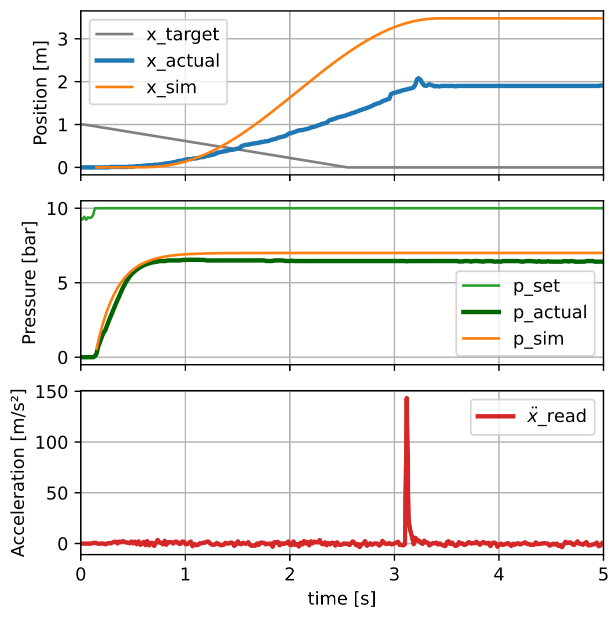


Figure 23 Test Campaign Run #1, with wrong mapping, resulted in full throttle compared with simulation

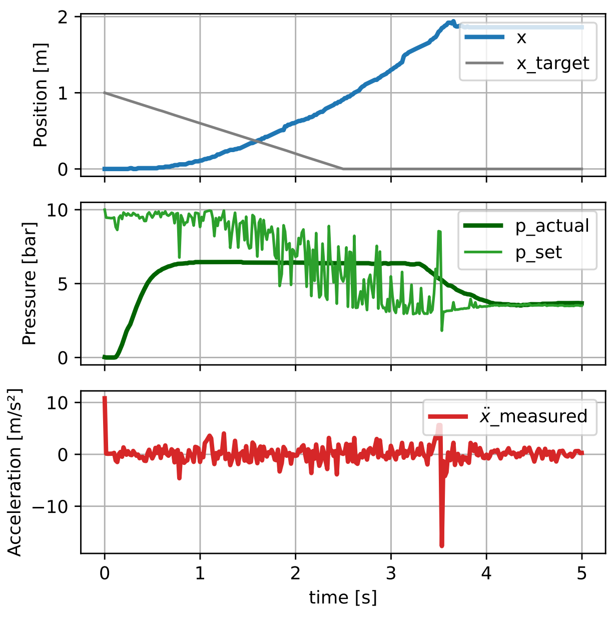


Figure 25 Test Campaign Run #4, with wrong action space mapping and unexplained lack of response