NTNU
Kunnskap for en bedre verden

Portfolio Assignment 2

Inverted Pendulum Control using Reinforcement Learning trained in a Simulation

AIS4002 - Intelligent Machines

Group 1: Modestas Sukarevicius, Jon Urcelay San Roman

7th May 2025

Contents

List of Figures

List of Tables

Acronyms

RL Reinforcement Learning.

EOM Equations Of Motion.

COG Center Of Gravity.

DOF Degree Of Freedom.

PID Proportional Integral Derivative.

LQR Linear Quadratic Regulator.

PPO Proximal Policy Optimization.

SAC Soft Actor Critic.

CPU Central Processing Unit.

GPU Graphical Processing Unit.

PBT Population Based Training.

NN Neural Network.

1 Objective

The objective of this project is to use RL to control the QUBE-Servo 2 rotating pendulum and stabilize it with the pendulum arm in the vertical position, as shown in Figure 1.

Figure 1: QUBE-Servo 2 with pendulum arm in vertical position.

The pendulum has two encoders, one for each angle, and a voltage input to control the rotation of the motor. The RL algorithm will use these angles to know the current state of the pendulum and will output the voltage that the motor needs to receive to achieve the wanted vertical position for the pendulum arm.

To achieve the objective of the project, the RL algorithm will first be trained in a virtual simulation, to avoid any harm to the physical pendulum. Once the algorithm works in the virtual environment, the same trained model will be used in the real pendulum. Finally, if this doesn't work, further modifications will be made to make it work in the real pendulum.

Consequently, the proposed methodology for this project is the following:

1. Obtain a virtual simulation of the pendulum.
2. Train the RL algorithm to make it work in the virtual simulation.
3. Make the necessary changes to the RL algorithm training process to make it work for the real pendulum.

The remaining chapters of the report will follow the content described in these steps.

2 Virtual simulation

To obtain a virtual simulation model of the pendulum, the EOM must be derived. For this, there are two main classical mechanical approaches which provide equivalent equations: Newton's laws (force-based equations) or Lagrangian mechanics (energy-based equations). As the Lagrange approach is less complicated in terms of formulas, this is the one that will be used.

In this chapter, the equations for the model will be derived using Lagrangian mechanics. In addition, simulation results will be displayed for basic validation, and the model will be tested with an LQR controller to make sure that its possible to achieve the control objective with the obtained model.

2.1 Lagrange's Equations of Motion

In this chapter, an initial model of the pendulum is developed based on the models obtained in AntonioCruz et al., 2015 and Le et al., 2020.

The necessary parameters and variables that define the pendulum's EOM are showed in Tables 1 and 2 and Figures 2 and 3. The value of the parameters have been obtained from Le et al., 2020 and modified by comparing results with the real pendulum.

Parameter	Symbology	Value
Pendulum mass	m_1	0.053 kg
Arm length	L_0	0.086 m
Pendulum length	L_1	0.128 m
Location of the COG of the pendulum	l_1	$\frac{0.128}{2}$ m
Moment of inertia of the arm	I_0	0.0001172 $kg \cdot m^2$
Moment of inertia of the pendulum	I_1	0.0000235 $kg \cdot m^2$
Damping coefficient of the arm	D_0	0.0004 $N \cdot m \cdot s/rad$
Damping coefficient of the pendulum	D_1	0.000003 $N \cdot m \cdot s/rad$
Internal resistance of the motor	R_m	8.94 Ω
Motor back-emf constant	K_m	0.0431

Table 1: Rotational pendulum parameters.

Variable	Symbology	Value range
Angular position of the arm	θ_0	$(-\pi, +\pi]rad$
Angular position of the pendulum	θ_1	$(-\pi, +\pi]rad$
Input voltage to the motor	V_m	$[-24, +24]V$
Motor torque	τ	Depends on voltage $N \cdot m$

Table 2: Rotational pendulum variables.

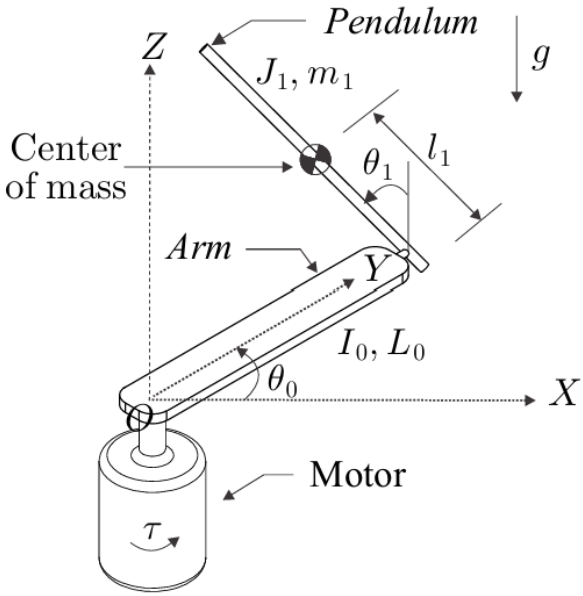


Figure 2: Rotational pendulum 3D representation with variables and parameters.

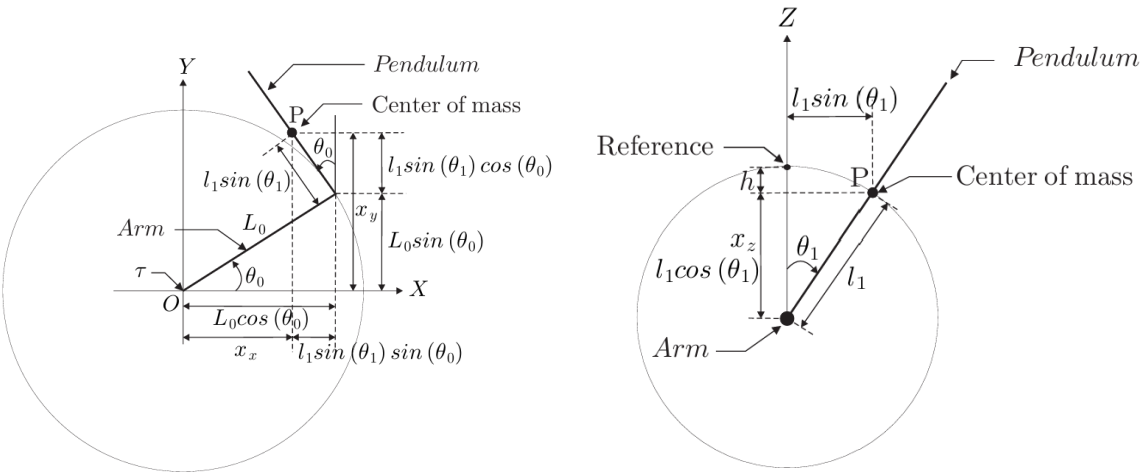


Figure 3: Top (left) and front (right) view of rotational pendulum.

The Euler-Lagrange equations for a n degrees of freedom system are defined as:

$$\frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial \dot{q}_i} - \frac{\partial L(q,\dot{q})}{\partial q_i} = \tau_i \tag{1}$$

As the rotational pendulum is a 2 DOF system, it has 2 Lagrangian equations, which are defined

as:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_0} - \frac{\partial L}{\partial \theta_0} = \tau - D_0\dot{\theta}_0 \tag{2}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_1} - \frac{\partial L}{\partial \theta_1} = -D_1\dot{\theta}_1 \tag{3}$$

The Langragian (L) is defined as:

$$L(q(t), \dot{q}(t)) = K(q(t), \dot{q}(t)) - U(q(t)), \tag{4}$$

where K is the kinetic energy of the system, U is its potential energy and $q(t) = [\theta_0, \theta_1]^T$.

On the one side, K is the sum of the kinetic energy of the arm and of the pendulum, which are defined as:

$$K_0 = \frac{1}{2}I_0\dot{\theta}_0^2 \tag{5}$$

$$K_1 = \frac{1}{2}J_1\dot{\theta}_1^2 + \frac{1}{2}m_1 v_1^T v_1 \tag{6}$$

The linear velocity term is obtained as:

$$v_1^T v_1 = \dot{x}_x^2 + \dot{x}_y^2 + \dot{x}_z^2 \tag{7}$$

where $\dot{x}_x, \dot{x}_y$ and $\dot{x}_z$ are the pendulum's COG's velocities. To the define the velocities, the coordinates need to be defined first:

$$x_x = L_0 cos(\theta_0) - l_1 sin(\theta_1) sin(\theta_0) \tag{8}$$

$$x_y = L_0 sin(\theta_0) + l_1 sin(\theta_1) cos(\theta_0) \tag{9}$$

$$x_z = l_1 cos(\theta_1) \tag{10}$$

The velocity components are obtained by differentiating each position:

$$\dot{x}_x = -\dot{\theta}_0 L_0 sin(\theta_0) - l_1(\dot{\theta}_0 sin(\theta_1)cos(\theta_0) + \dot{\theta}_1 cos(\theta_1)sin(\theta_0)) \tag{11}$$

$$\dot{x}_y = \dot{\theta}_0 L_0 cos(\theta_0) + l_1(\dot{\theta}_1 cos(\theta_1)cos(\theta_0) - \dot{\theta}_0 sin(\theta_1)sin(\theta_0)) \tag{12}$$

$$\dot{x}_z = -\dot{\theta}_1 l_1 sin(\theta_1) \tag{13}$$

Substituting equations 11, 12 and 13 into 7, the next expression is obtained for the linear velocity term:

$$v_1^T v_1 = L_0^2\dot{\theta}_0^2 + l_1^2(\dot{\theta}_1 + \dot{\theta}_0 sin^2(\theta_1)) + 2L_0 L_1 m_1 \dot{\theta}_0 \dot{\theta}_1 cos(\theta_1) \tag{14}$$

On the other hand, V is the sum of the potential energy of the arm and the pendulum. As the arm moves in an horizontal plane, its potential energy is constant and can be considered equal

to 0. So the potential energy V is reduced to the potential energy of the pendulum, which is defined as:

$$V = m_1 g l_1 (cos(\theta_1) - 1) \tag{15}$$

Substituting equations 5, 6, 14 and 15 into equation 4, the Lagrangian (L) can be obtained. By computing the partial derivatives of the Lagrangian and substituting as described in equations 2 and 3, the 2 EOM of the system are defined as:

$$\tau - D_0\dot{\theta}_0 = \alpha\ddot{\theta}_0 + \beta\dot{\theta}_0\dot{\theta}_1 + \gamma\ddot{\theta}_1 - \sigma\dot{\theta}_1^2 \tag{16}$$

$$-D_1\dot{\theta}_1 = \gamma\ddot{\theta}_0 + (m_1 l_1^2 + I_1)\ddot{\theta}_1 - \frac{1}{2}\beta\dot{\theta}_0^2 - m_1 g l_1 sin(\theta_1) \tag{17}$$

where,

$$\alpha = I_0 + m_1 L_0^2 + m_1 l_1^2 sin^2(\theta_1) \tag{18}$$

$$\beta = m_1 l_1^2 sin(2\theta_1) \tag{19}$$

$$\gamma = m_1 L_0 l_1 cos(\theta_1) \tag{20}$$

$$\sigma = m_1 L_0 l_1 sin(\theta_1) \tag{21}$$

Finally, the relation between the control input (voltage of the motor V_m) and the motor torque (τ) is given by:

$$\tau = K_m \frac{V_m - K_m\dot{\theta}_0}{R_m} \tag{22}$$

Including equation 22 into the OEM's, the final model can be described as:

$$K_m \frac{V_m - K_m\dot{\theta}_0}{R_m} - D_0\dot{\theta}_0 = \alpha\ddot{\theta}_0 + \beta\dot{\theta}_0\dot{\theta}_1 + \gamma\ddot{\theta}_1 - \sigma\dot{\theta}_1^2 \tag{23}$$

$$-D_1\dot{\theta}_1 = \gamma\ddot{\theta}_0 + (m_1 l_1^2 + I_1)\ddot{\theta}_1 - \frac{1}{2}\beta\dot{\theta}_0^2 - m_1 g l_1 sin(\theta_1) \tag{24}$$

where,

$$\alpha = I_0 + m_1 L_0^2 + m_1 l_1^2 sin^2(\theta_1) \tag{25}$$

$$\beta = m_1 l_1^2 sin(2\theta_1) \tag{26}$$

$$\gamma = m_1 L_0 l_1 cos(\theta_1) \tag{27}$$

$$\sigma = m_1 L_0 l_1 sin(\theta_1) \tag{28}$$

2.2 Adaptation of equations to real pendulum

After obtaining the model, it has been adapted to the orientations and zero positions of both angles (See Figure 4). This means that the next changes need to be applied to the equations from 23 to 28:

$$\theta_{0,new} = -\theta_{0,old}$$
$$\theta_{1,new} = -(\theta_{1,old} + 180^o)$$

which means:

$$\dot{\theta}_{0,new} = -\dot{\theta}_{0,old}$$
$$\ddot{\theta}_{0,new} = -\ddot{\theta}_{0,old}$$
$$sin(\theta_{0,new}) = -sin(\theta_{0,old})$$
$$cos(\theta_{0,new}) = cos(\theta_{0,old})$$
$$\dot{\theta}_{1,new} = -\dot{\theta}_{1,old}$$
$$\ddot{\theta}_{1,new} = -\ddot{\theta}_{1,old}$$
$$sin(\theta_{1,new}) = -sin(\theta_{1,old})$$
$$cos(\theta_{1,new}) = -cos(\theta_{1,old})$$

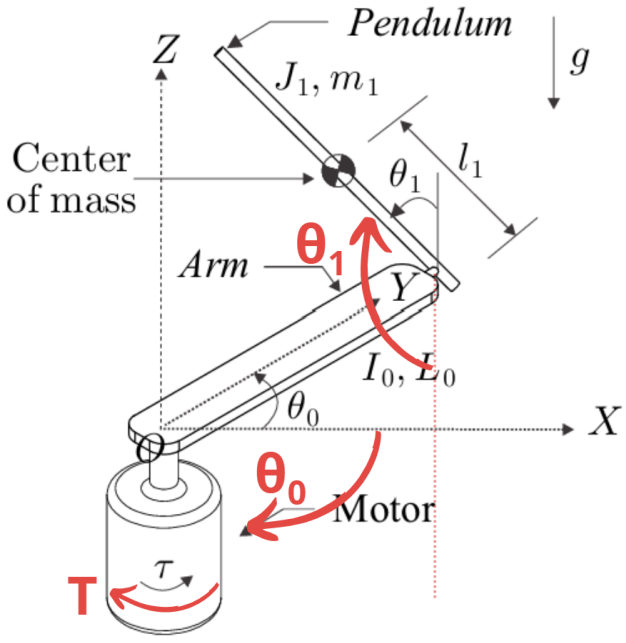


Figure 4: Rotational pendulum 3D representation with variables and parameters adapted to real QUBE Servo 2.

In addition, a spring stiffness k has been added to θ_0, to simulate the effect of the encoders cable, which pushes back the pendulum to its original θ_0 when no voltage is applied. The value of k has been estimated by trial and error (Table 3).

Physical characteristic	Symbology	Value
Spring stiffness	k	0.002 $N.m/rad$

Table 3: Rotational pendulum spring stiffness parameter.

Applying these changes, the final model can be described as:

$$-K_m\frac{V_m - K_m\dot{\theta}_0}{R_m} + D_0\dot{\theta}_0 + k\theta_0 = -\alpha\ddot{\theta}_0 + \beta\dot{\theta}_0\dot{\theta}_1 - \gamma\ddot{\theta}_1 - \sigma\dot{\theta}_1^2 \tag{29}$$

$$D_1\dot{\theta}_1 = -\gamma\ddot{\theta}_0 - (m_1l_1^2 + I_1)\ddot{\theta}_1 - \frac{1}{2}\beta\dot{\theta}_0^2 - m_1gl_1sin(\theta_1) \tag{30}$$

where,

$$\alpha = I_0 + m_1L_0^2 + m_1l_1^2sin^2(\theta_1) \tag{31}$$

$$\beta = -m_1l_1^2sin(2\theta_1) \tag{32}$$

$$\gamma = -m_1L_0l_1cos(\theta_1) \tag{33}$$

$$\sigma = m_1L_0l_1sin(\theta_1) \tag{34}$$

2.3 Simulation

The state of the pendulum is represented as:

$$x = [sin(\theta_0), cos(\theta_0), \dot{\theta}_0, sin(\theta_1), cos(\theta_1), \dot{\theta}_1]^T \tag{35}$$

The simulation is solved using Runge-Kutta 4:

$$x_{k+1} = x_k + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{36}$$

where,

$$k_1 = f(t_k, x_k) \tag{37}$$

$$k_2 = f(t_k + \frac{dt}{2}, x_k + dt\frac{k_1}{2}) \tag{38}$$

$$k_3 = f(t_k + \frac{dt}{2}, x_k + dt\frac{k_2}{2}) \tag{39}$$

$$k_4 = f(t_k + dt, x_k + dtk_3) \tag{40}$$

$$f(t_k, x_k) = \dot{x}_k = [\dot{\theta}_0cos(\theta_0), -\dot{\theta}_0sin(\theta_0), \ddot{\theta}_0, \dot{\theta}_1cos(\theta_1), -\dot{\theta}_1sin(\theta_1), \ddot{\theta}_1]_k^T \tag{41}$$

and $\ddot{\theta}_0$ and $\ddot{\theta}_1$ are found by solving the dynamics of the system defined in equations 29 and 30, which can be represented in matrix form as:

$$\begin{bmatrix} \alpha & \gamma \\ \gamma & m_1l_1^2 + I_1 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_0 \\ \ddot{\theta}_1 \end{bmatrix} = \begin{bmatrix} K_m\frac{V_m - K_m\dot{\theta}_0}{R_m} - D_0\dot{\theta}_0 + \beta\dot{\theta}_0\dot{\theta}_1 - \sigma\dot{\theta}_1^2 \\ -D_1\dot{\theta}_1 - \frac{1}{2}\beta\dot{\theta}_0^2 - m_1gl_1sin(\theta_1) \end{bmatrix} \tag{42}$$

where α, β, γ and σ are given in equations 31, 32, 33 and 34 respectively.

2.4 Model validation

To validate the model, it will be compared with data collected from the real pendulum. Two tests will be made: (1) with a voltage step input; and (2) with a sinusoidal voltage input. The results can be seen in Figures 5 and 6, respectively.

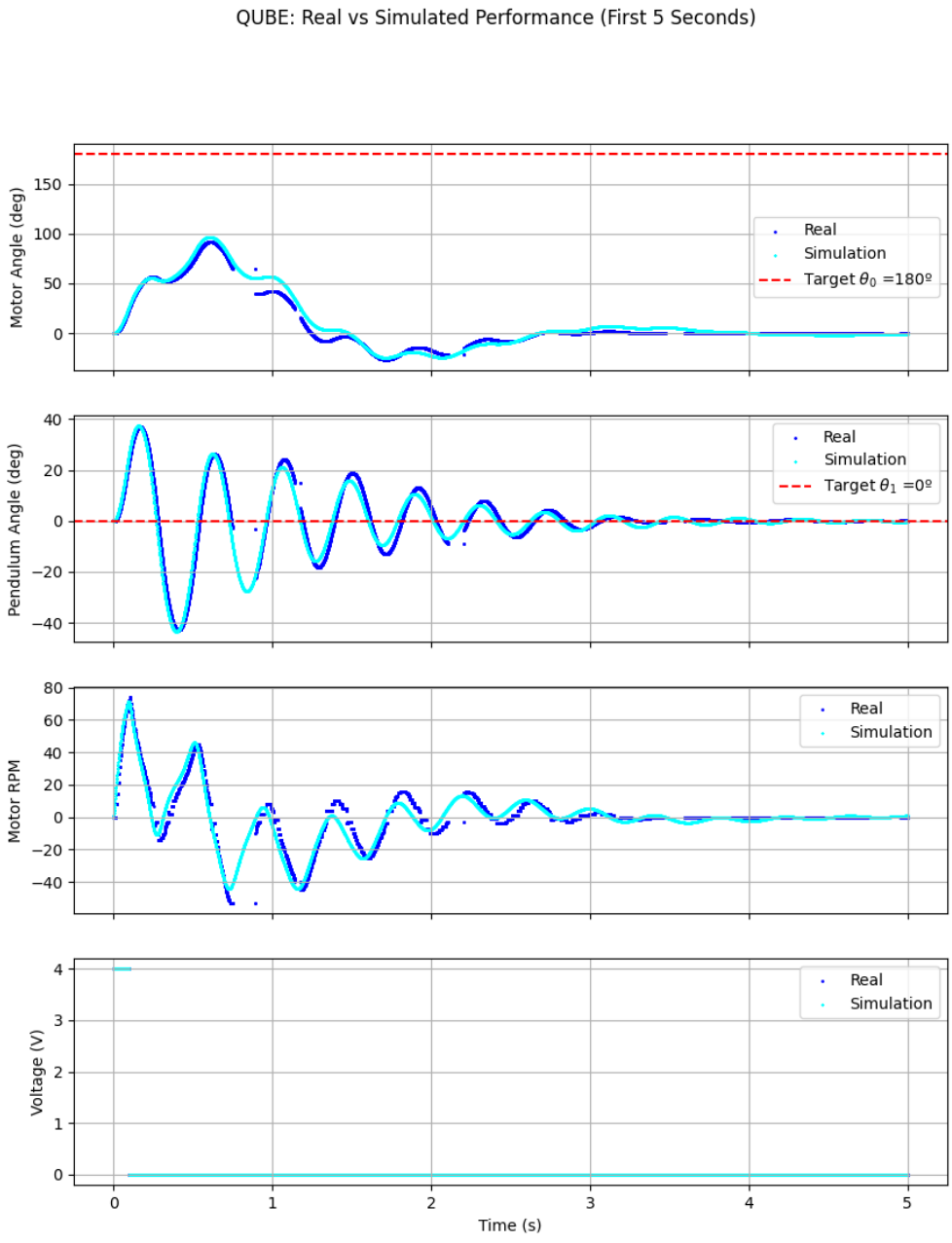


Figure 5: Step voltage input (4V) for the simulation and for the real pendulum.

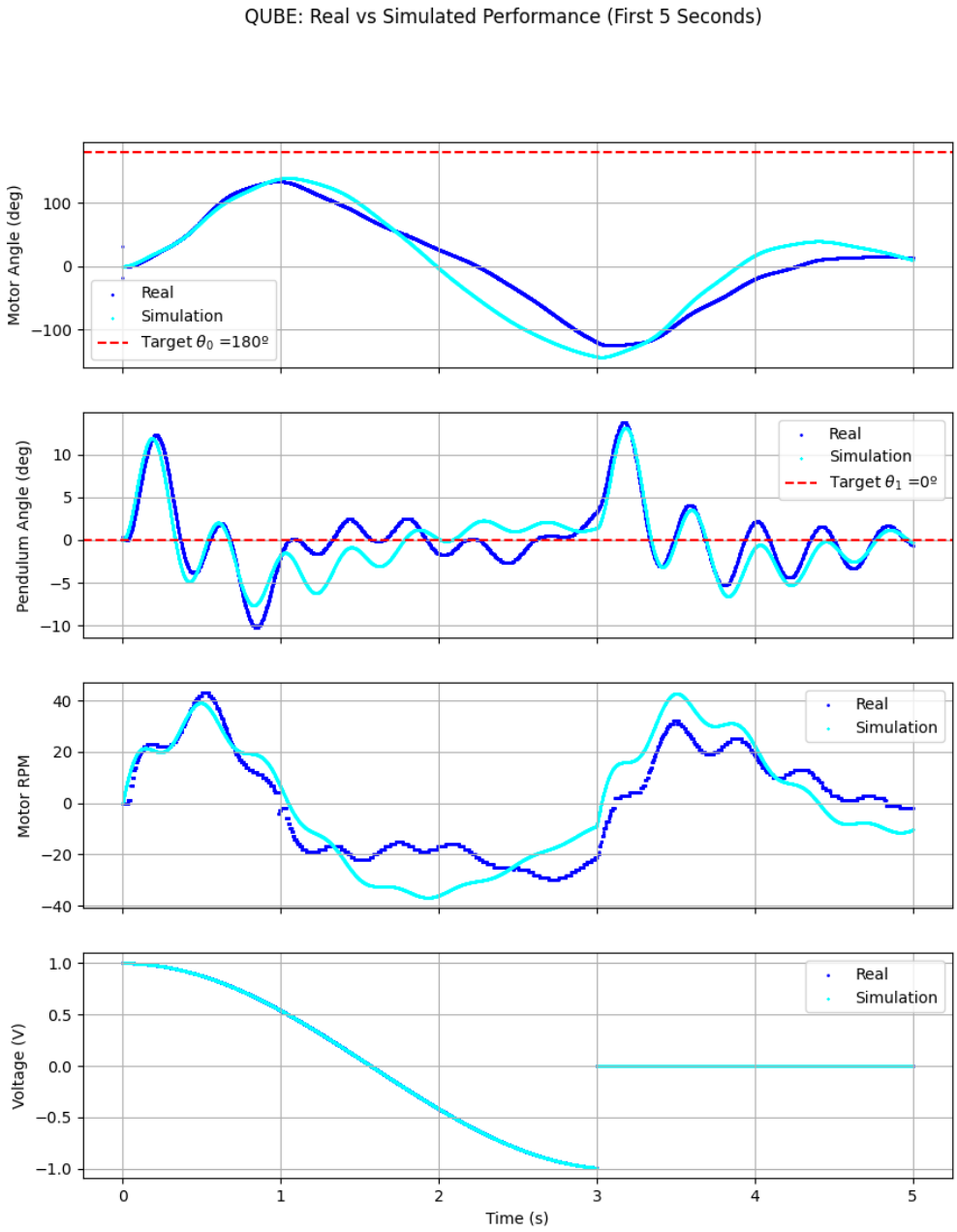


Figure 6: Sinusoidal input for the simulation and for the real pendulum.

As observed in the results, the simulation follows reality very closely with the step input, but it's not very accurate when the input is sinusoidal. However, at this point, this model is considered good enough for training.

2.5 Control validation

The objective of the control validation is to make sure that the model of the pendulum is can be stabilized in the vertical position. For this, a swing up operation(including energy) and LQR controller are used to stabilize the pendulum. Once this is achieved, it can be guaranteed that if the RL fails to stabilize the inverted pendulum, it's not because of any issue in the model. As it can be seen in Figure 7, the pendulum's virtual model is possible to stabilize.

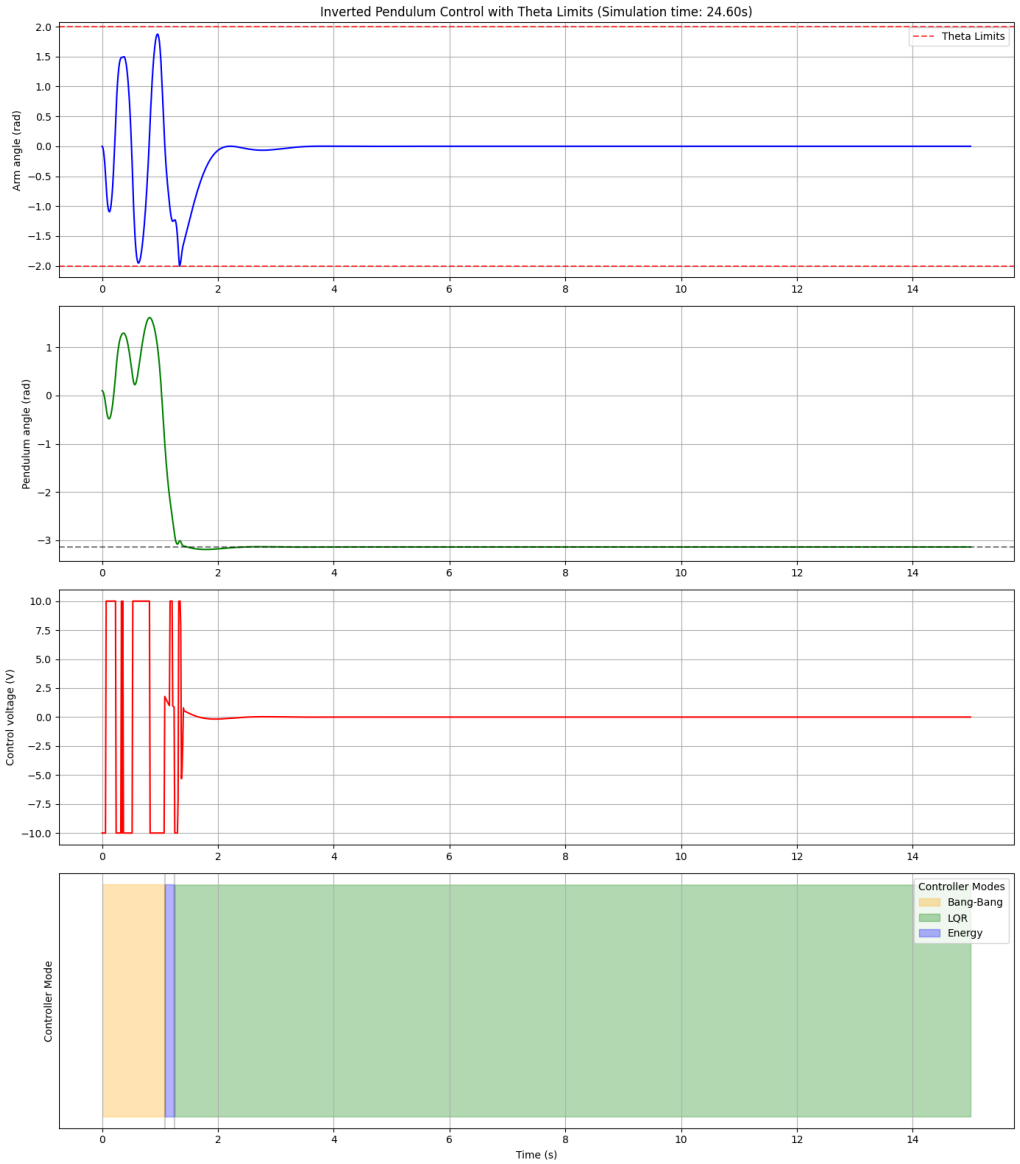


Figure 7: Model stabilization using LQR.

3 Reinforcement learning for simulated pendulum

This chapter focuses on training a RL algorithm capable of controlling and achieving the inverted equilibrium in pendulum simulation. Different aspects related to RL will be discussed, finally showing the RL algorithm achieved.

3.1 RL algorithms

In RL there are different algorithms to train agents. For training an agent to control an inverted pendulum, it has been found that PPO failed to achieve stable performance, while SAC succeeded. A small explanation of each algorithm is provided below:

Proximal Policy Optimization (PPO)
PPO is a policy gradient algorithm that refines an agent's decision-making strategy by directly optimizing its policy based on rewards from freshly collected episodes. It's an on-policy method, meaning it uses data sampled from the current policy and discards it after each update, which can limit sample efficiency. PPO ensures stability by clipping its objective function, preventing overly large policy changes in a single step. Computationally, its sequential sampling often runs efficiently on CPUs, especially in simpler setups, though GPUs help with larger-scale problems.

Soft Actor-Critic (SAC)
SAC is an off-policy algorithm that simultaneously learns a policy (the actor) and a value function (the critic) while maximizing both expected rewards and policy entropy. Unlike on-policy methods, SAC reuses past experiences stored in a replay buffer, boosting its sample efficiency. The entropy term encourages exploration by rewarding slightly random actions, which helps the agent avoid getting stuck in suboptimal behaviors. Computationally, its batch-based updates and multiple networks benefit from GPU acceleration, particularly for complex, high-dimensional tasks.

In the next two figures, the results for a PPO (Figure 8) and SAC trained model (9) are shown. The models are trained using the same reward function: $reward = 2(1 - cos(\theta_1))$, full observability and a frame history of 2. More details about the reward function, observability and frame history will be explained and discussed in the next sections. The PPO model is trained using CPU, while SAC is trained using GPU. This affects training time, but does not affect the final result.

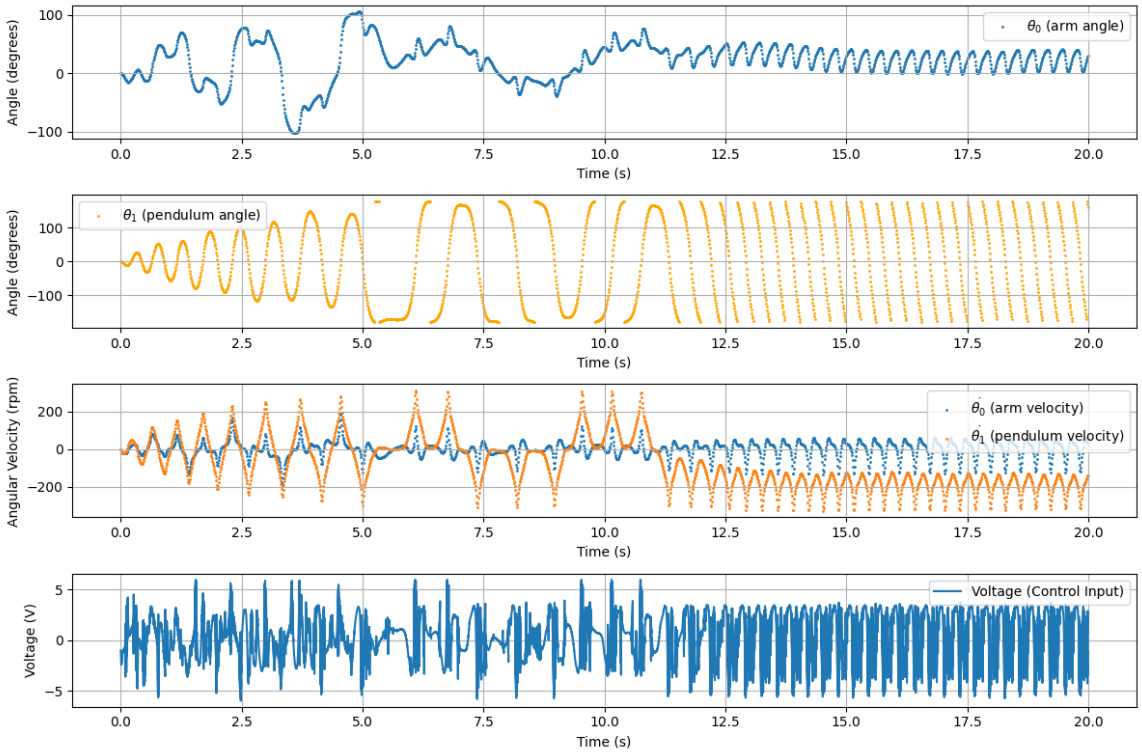


Figure 8: PPO with full observability, frame history = 2 and $reward = 2(1 - cos(\theta_1))$.

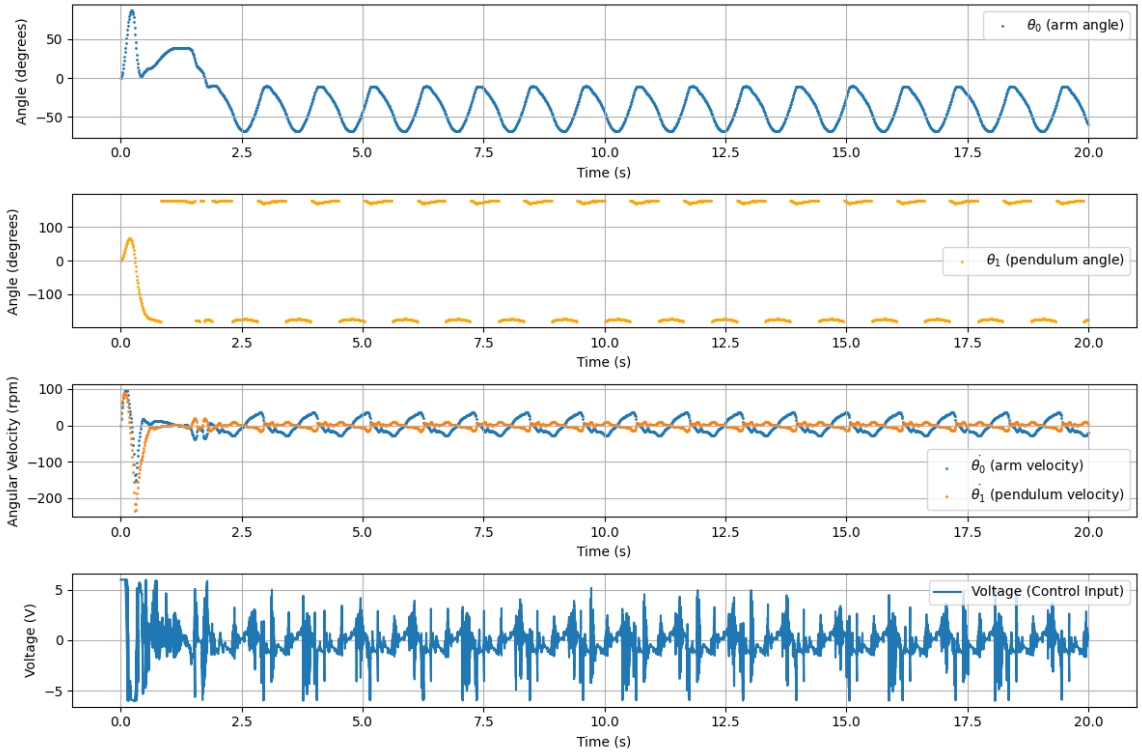


Figure 9: SAC with full observability, frame history = 2 and $reward = 2(1 - cos(\theta_1))$.

As it can be seen in the results, the PPO model fails to achieve equilibrium, while SAC achieves it.

	Total training time	Total training steps	Average frames per second (fps)
PPO	0 h 41 min	2 M	810
SAC	1 h 40 min	0.27 M	45

Table 4: PPO vs SAC training.

When it comes to training time, SAC took more training time to compute, but less total steps than PPO while obtaining better result.

Following how each of the algorithms is built and trying to find the logic behind the models, it is most likely that PPO's on-policy approach and clipped updates limited its adaptability and exploration in this continuous control task. Conversely, SAC's off-policy data reuse and entropy-driven exploration enabled it to handle the pendulum's dynamics and converge effectively.

3.2 Reward function

The reward is a scalar value that reflects the immediate benefit (or penalty) of taking action a in state s and transitioning to s', based on how desirable or undesirable the outcome is. The agent's ultimate aim is to maximize the total cumulative reward over time, which guides its learning process.

Designing the reward function for an agent is a very important task. An incorrect reward function will lead to bad results, no matter how good the RL algorithm is.

For the pendulum, the next reward function has been designed:

$$R = r_{upright} + r_{arm} + r_{energy} + r_{armlimit} + r_{up-stable} + r_{down-stable} r_{up-middle-stable}$$

Pendulum's angle position reward.
This gives the pendulum a positive reward when the pendulum is upwards, this is when $\theta_1 = 180^o$

$$r_{upright} = -2cos(\theta_1)$$

Arm in the center reward.

This compensates the pendulum for having the arm close to its origin, this is when $\theta_0 = 0$.

$$r_{arm} = -0.1tanh(\frac{\theta_0^2}{2})$$

Pendulum's energy reward.
This energy based reward is maximum at any position where the pendulum has the same amount of energy(kinetic + potential) as in upward position.

$$r_{energy} = 2 - 0.007|m_1 g l_1 - (m_1 g l_1 cos(\theta_1) + \frac{1}{2} I_1 \dot{\theta}_1^2)|^2$$

Arm's limits penalty.
The arm's limit penalty is computed by comparing the proximity of θ_0 to the upper $(2.2rad)$ and lower $(-2.2rad)$ bounds. The functions $\theta_{\text{max\_dist}}$ and $\theta_{\text{min\_dist}}$ clip the value between 0 and 1 (with a linear decay over 0.5) to measure closeness. Their maximum is cubed and scaled by -10.0 to sharply increase the penalty as θ_0 nears a limit.

$$\theta_{\text{max/min\_dist}} = \text{clip}\left(1.0 - \frac{|\theta_0 \mp 2.2|}{0.5},\, 0,\, 1\right)$$
$$r_{\text{armlimit}} = -10.0\Big(\max\big(\theta_{\text{max\_dist}},\, \theta_{\text{min\_dist}}\big)\Big)^3$$

Pendulum upright and stable bonus.
This reward increases when the pendulum angle is upright and not moving ($\theta_1 = 180^o$ and $\dot{\theta}_1 = 0$).

$$up_1 = e^{-10(|\theta_1| - \pi)^2}; stable_1 = e^{-\dot{\theta}_1^2}$$
$$r_{up-stable} = 3up_1 stable_1$$

Pendulum downright and stable cost. This penalization increases (more negative) when the pendulum angle is downright and not moving ($\theta_1 = 0^o$ and $\dot{\theta}_1 = 0$).

$$down_1 = e^{-10\theta_1^2}; stable_1 = e^{-\dot{\theta}_1^2}$$
$$r_{down-stable} = -3down_1 stable_1$$

Pendulum upright and stable + arm in the middle and stable bonus.
This reward increases when the pendulum angle is upright and not moving ($\theta_1 = 180^o$ and

$\dot{\theta}_1 = 0$) and when the arms angle is in the middle and not moving ($\theta_0 = 0^o$ and $\dot{\theta}_0 = 0$). Both at the same time.

$$up_1 = e^{-10(|\theta_1|-\pi)^2}; stable_1 = e^{-\dot{\theta}_1^2}$$
$$middle_0 = e^{-\theta_0^2}; stable_0 = e^{-\dot{\theta}_0^2}$$
$$r_{up} = up_1 stable_1 middle_0 stable_0$$

For a better understanding of the reward function, see the 3D visualization shown in Figures 10 and 11.

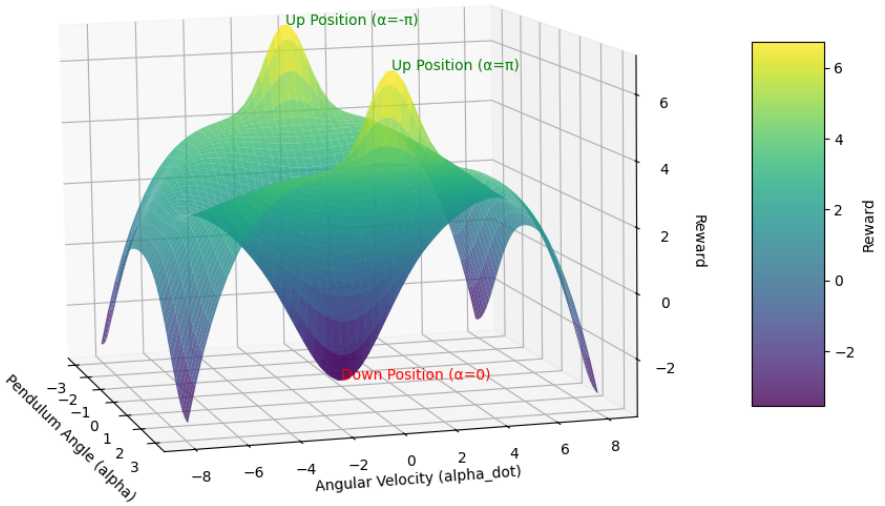


Figure 10: Reward function for $\theta_0 = 0$ and $\dot{\theta}_0 = 0$.

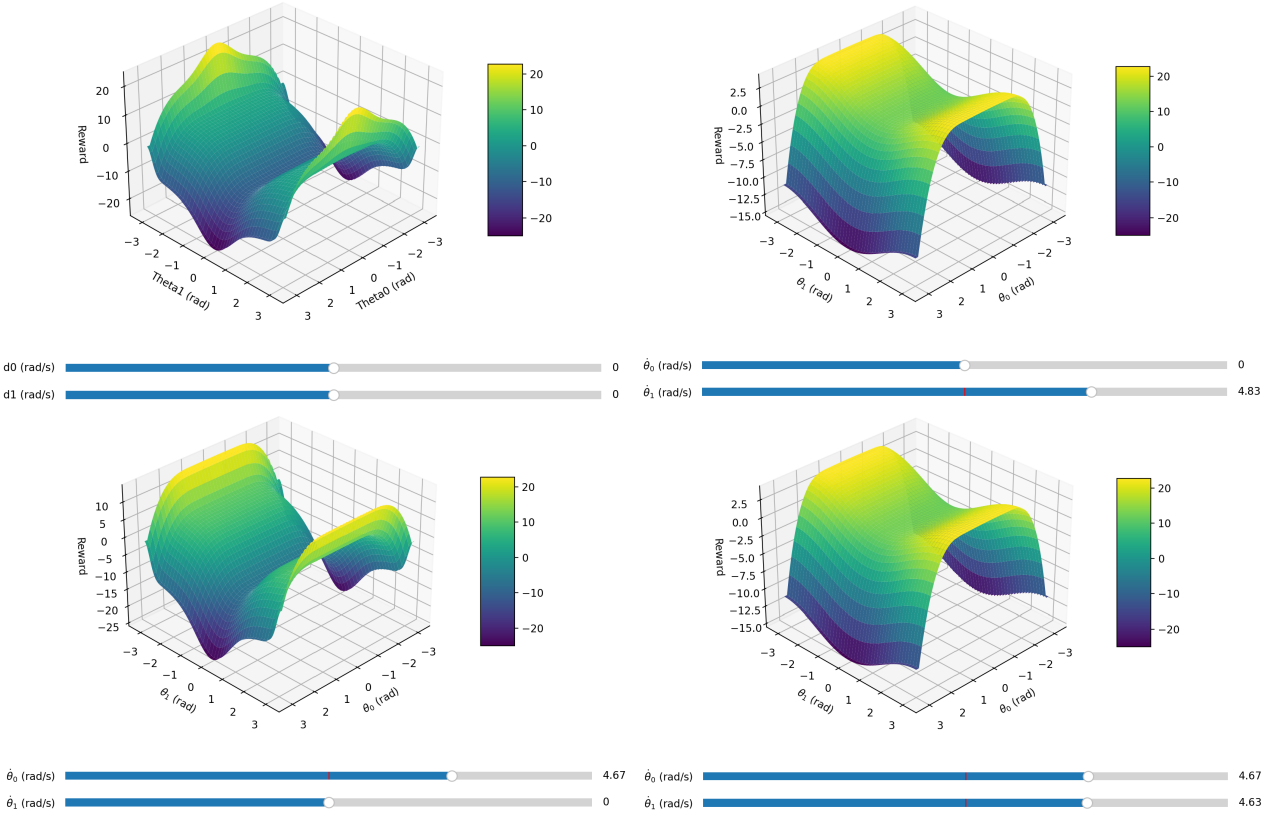
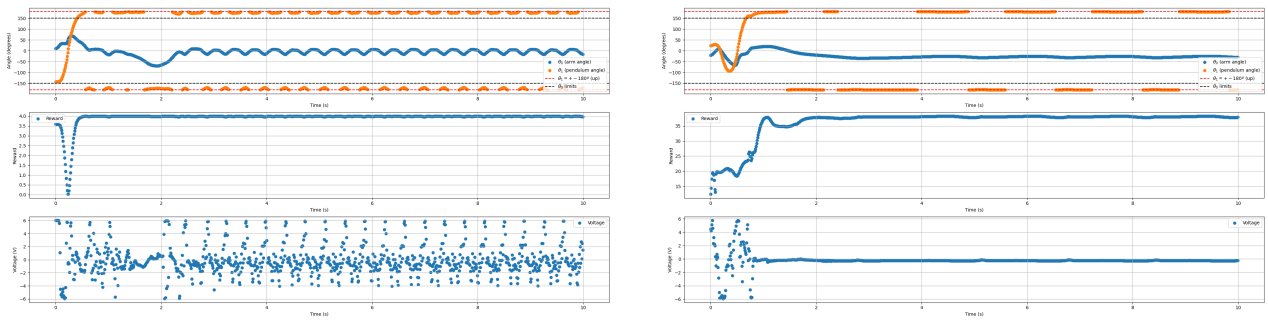


Figure 11: Reward function 3D representations.

For seeing the effect of such a reward function, it has been compared with a super simple reward function ($reward = 2(1 - cos(\theta_1))$). For this, two models have been trained using the same parameters and conditions, except of the reward function. As it can be observed in Figures 12a and 12b, the result of the designed reward function is more smooth, as with the simple reward function the pendulum reaches equilibrium but it dances around.



(a) (b)

Figure 12: SAC with full observability, frame history = 2 and (a) $reward = 2(1 - cos(\theta_1))$; (b) designed reward function.

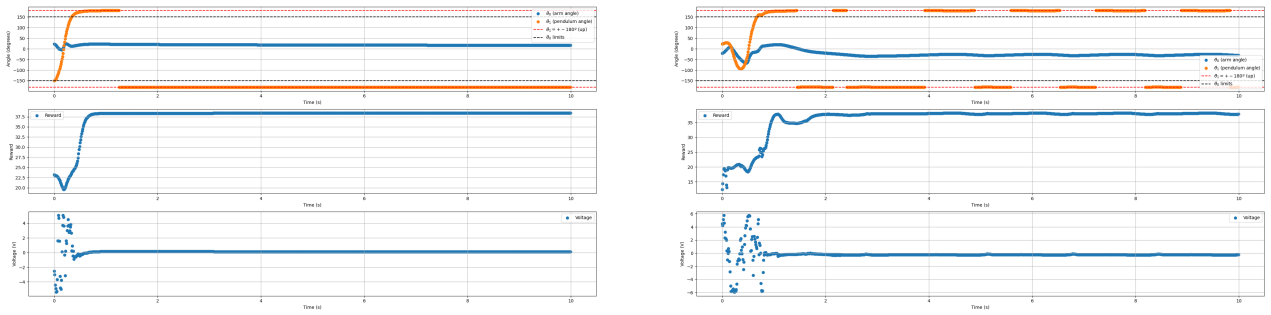
3.3 Frame history

Many RL environments provide the agent with a single frame at each time step. However, a single frame often doesn't contain enough information to understand the full context, like the direction an object is moving or the velocity of an agent. To solve this, the agent maintains a history of frames, usually stacking a fixed number of recent frames together as input.

Frame history refers to the sequence of past observations (or frames) that an agent uses to make decisions. This is then directly fed into the agent's policy or value function as part of the current state. It's typically small and only covers the immediate past.

This concept is not the same as replay buffer. A replay buffer is a storage mechanism that holds a much larger collection of past experiences of state, action, reward, next state from the agent's interactions with the environment. This is then used to train the agent by sampling random batches of experiences, breaking the correlation between consecutive time steps and improving learning stability.

For the inverted pendulum's simulation, it has been tested if increasing the frame steps leads to better results. For this, two different models have been trained, both under same conditions with SAC, full observability, same reward function and only changing the frame history. One of the models uses just one frame step, as the other uses two frame steps. See the results in Figures 13a and 13b.



(a) (b)

Figure 13: SAC with full observability and (a) frame history = 1; (b) frame history = 2.

As it can be observed, both achieve equilibrium, but the model with a single frame step achieves an arm angle much closer to 0. In this case, the simplest version provides better results, so there is no need of increasing the frame step number.

Later in the report, when facing partial observability, frame history will be used again.

4 Reinforcement learning for real pendulum

This chapter focuses on training a RL algorithm capable of controlling and achieving the inverted equilibrium in the real pendulum.

The "sim-to-real gap" in RL refers to the discrepancy between the performance of an RL policy trained in a simulated environment and its performance when deployed in the real world. This gap arises because simulations, no matter how detailed, cannot perfectly replicate the complexities, variability, and noise of real-world systems.

In this way, the main objective of this chapter is to use the RL algorithm trained in simulation, make some changes to fill the sim-to-real gap and use it in the physical pendulum. The content of this chapter will cover these changes.

4.1 Domain randomization

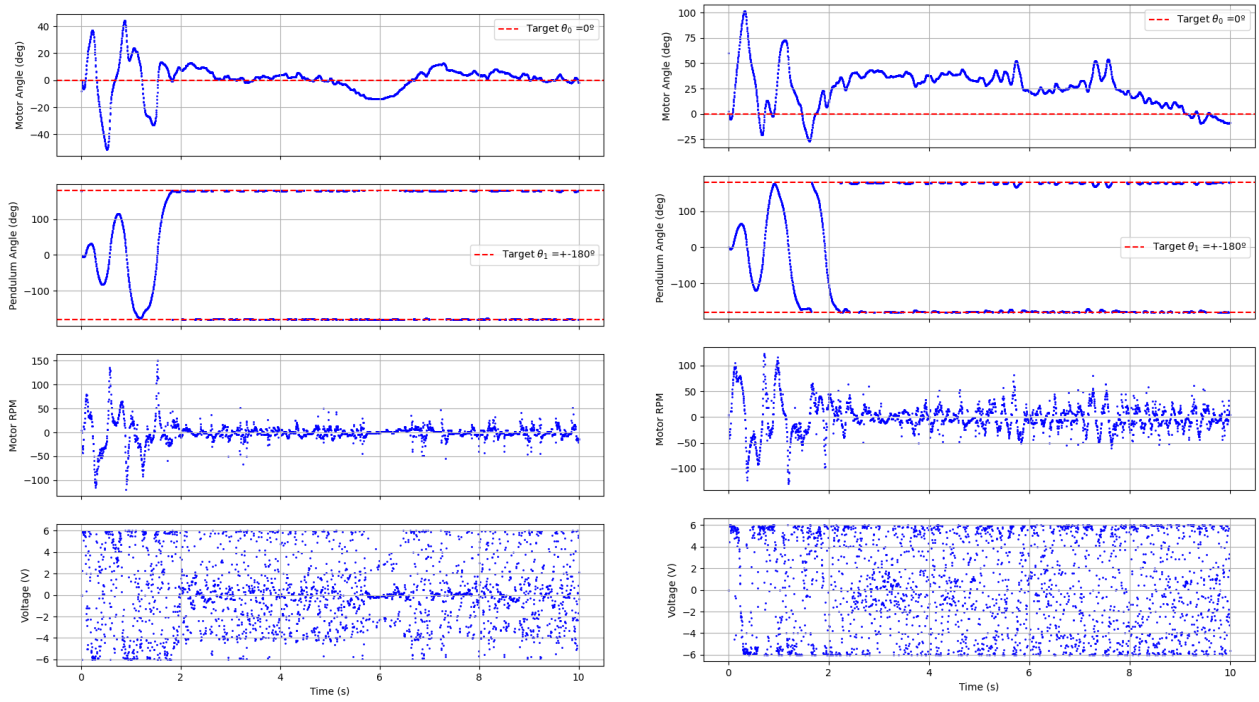
Domain randomization is a technique in RL to address the sim-to-real gap. It involves training an RL policy across a wide range of simulated environments by deliberately varying key parameters, dynamics, or conditions within the simulator.

If this feature is enabled, every new episode/epoch will have new randomized environment parameters for simulation. For the rotating pendulum, these parameters can be divided in the next groups:

- Voltage range: maximum and minimum voltage allowed from episode to episode.
- Parameter variance: parameters used in the simulation vary from episode to episode.
- Time step (dt) variance: dt changes from step to step with set deviation, maximum and minimum values.

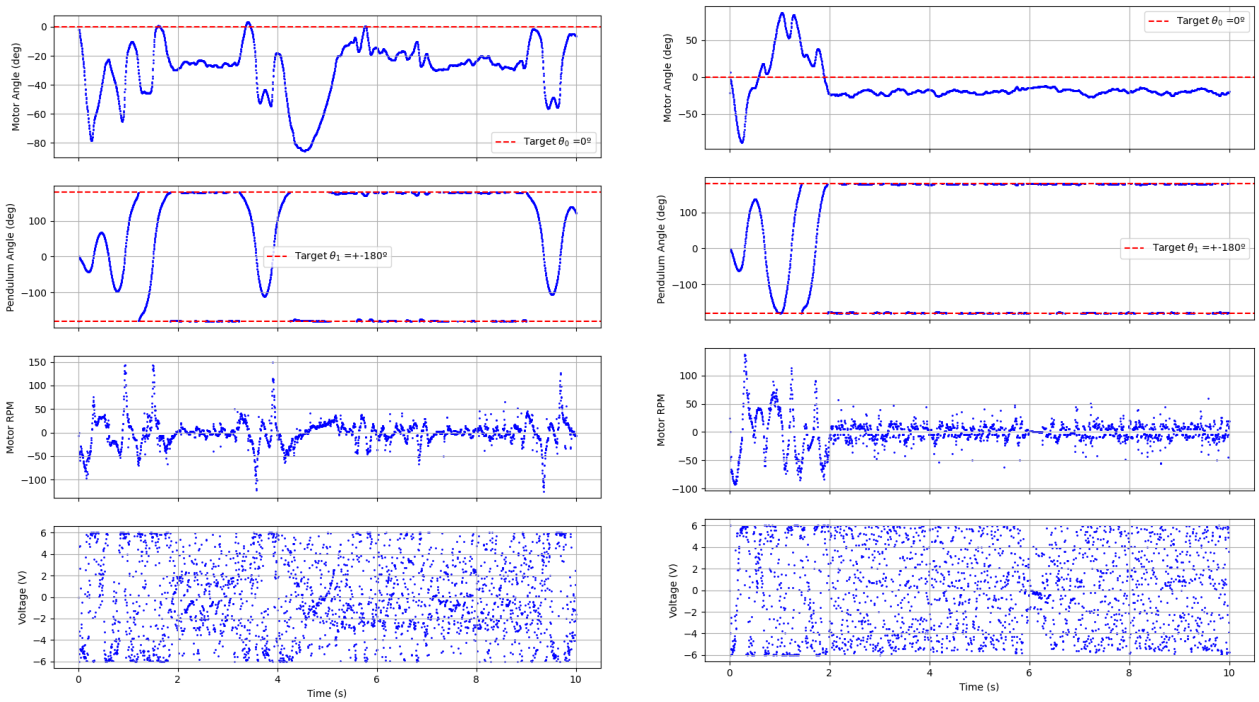
Simulation parameters for the rotating pendulum, which will be randomized, are showed in Tables 1 and 3.

To observe the effect of domain randomization, multiple tests have been made in the real pendulum, using RL algorithm in the same conditions and only changing the domain randomization. Figures 14 and 15 show real pendulum results for 0%, 20%, 30% and 40% parameter variation.



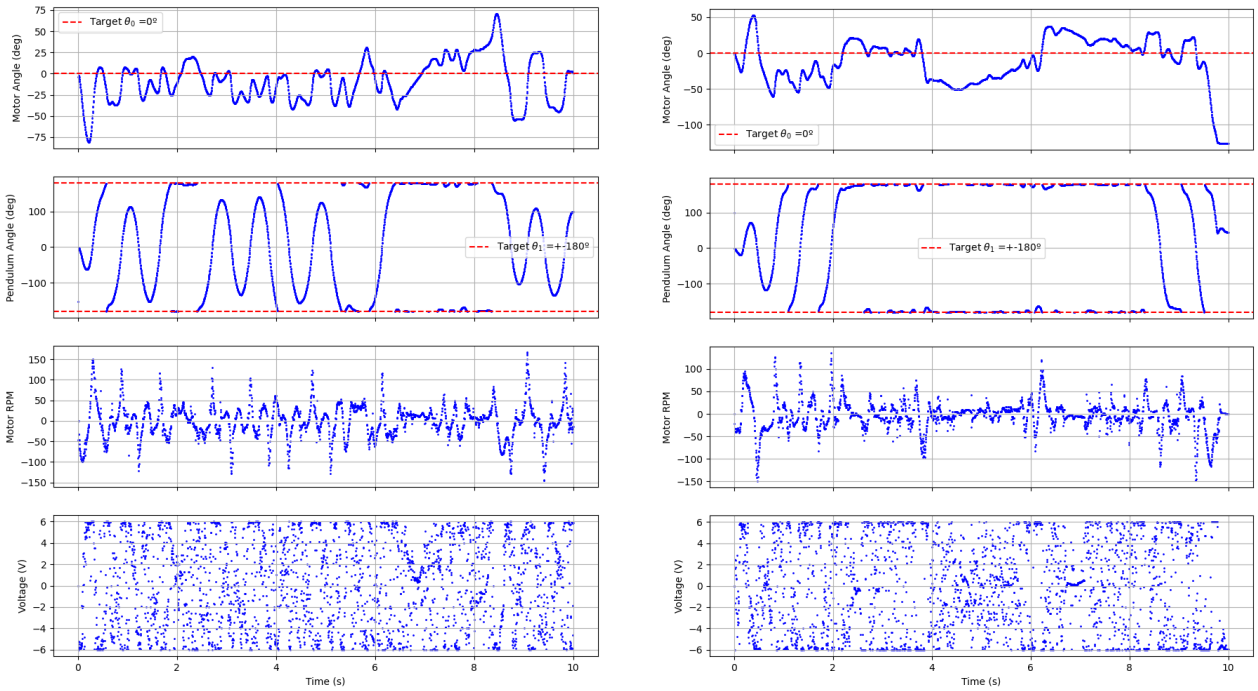
(a) (b)

Figure 14: SAC with (a) no randomization and (b) 20% randomization in real pendulum.



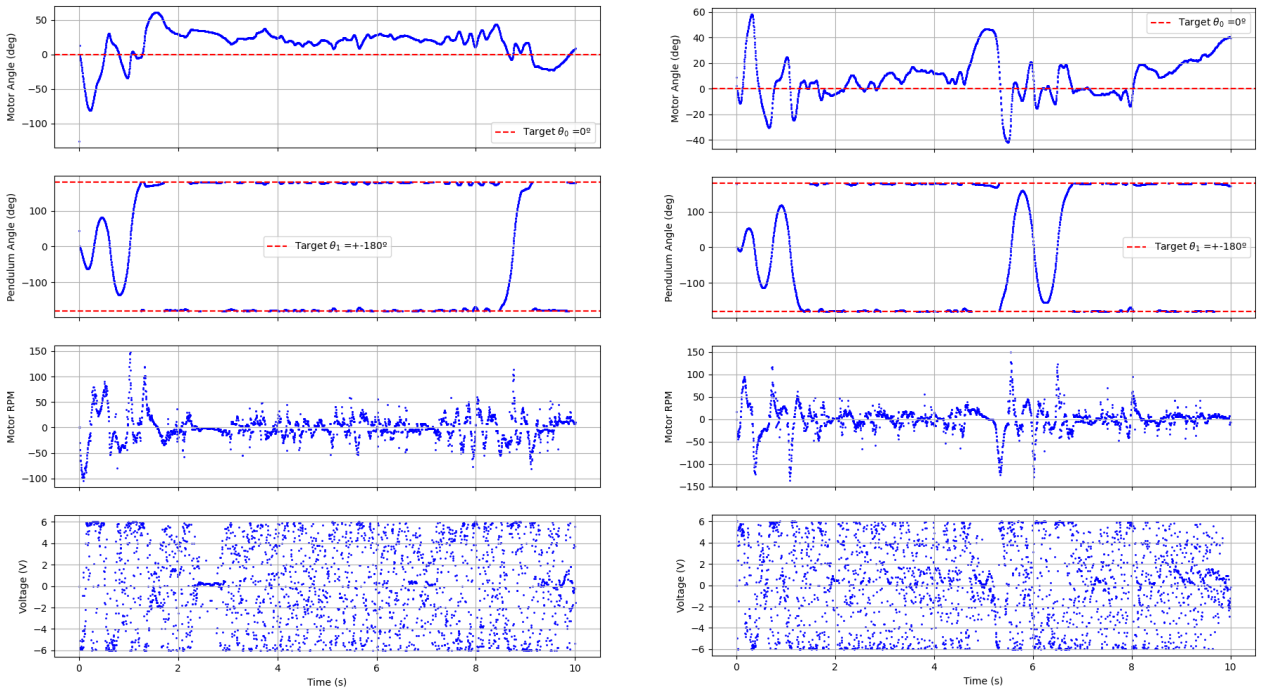
(a) (b)

Figure 15: SAC with (a) 30% randomization and (b) 40% randomization in real pendulum.



(a) (b)

Figure 16: SAC with (a) 50% randomization and (b) 60% randomization in real pendulum.



(a) (b)

Figure 17: SAC with (a) 70% randomization and (b) 80% randomization in real pendulum.

As it can be observed, more randomization does not necessarily provide better results, as the pendulum falls from equilibrium in some situations.

4.2 Variation on Population Based Training (PBT)

PBT is an advanced training method used to optimize both model parameters (neural network weights) and hyperparameters (learning rate, reward weights) simultaneously during training. Unlike traditional RL where a single model is trained with fixed hyperparameters, PBT maintains a population of multiple models, each with its own set of hyperparameters, and evolves them over time by selecting and adapting the best performers.

In this way, the implemented PBT methodology utilizes multiple parallel training episodes within identical domain randomization settings to identify optimal hyperparameters. The algorithm works as follows (see Figure 18):

1. A population of parallel training instances is initialized, each representing a distinct hyperparameter configuration.
2. In the initial episode, one instance executes with default hyperparameter settings while remaining instances utilize randomized hyperparameters, with variations sampled according to a standard deviation relative to the default settings.
3. Upon completion of an episode across all parallel instances, performance evaluation is conducted based on cumulative rewards. The hyperparameter configuration that achieved the highest reward is designated as the new default setting for subsequent episodes.
4. In succeeding episodes, lower-performing configurations are replaced with randomized variations of the best-performing configuration, thereby maintaining the exploration-exploitation balance throughout the training process.

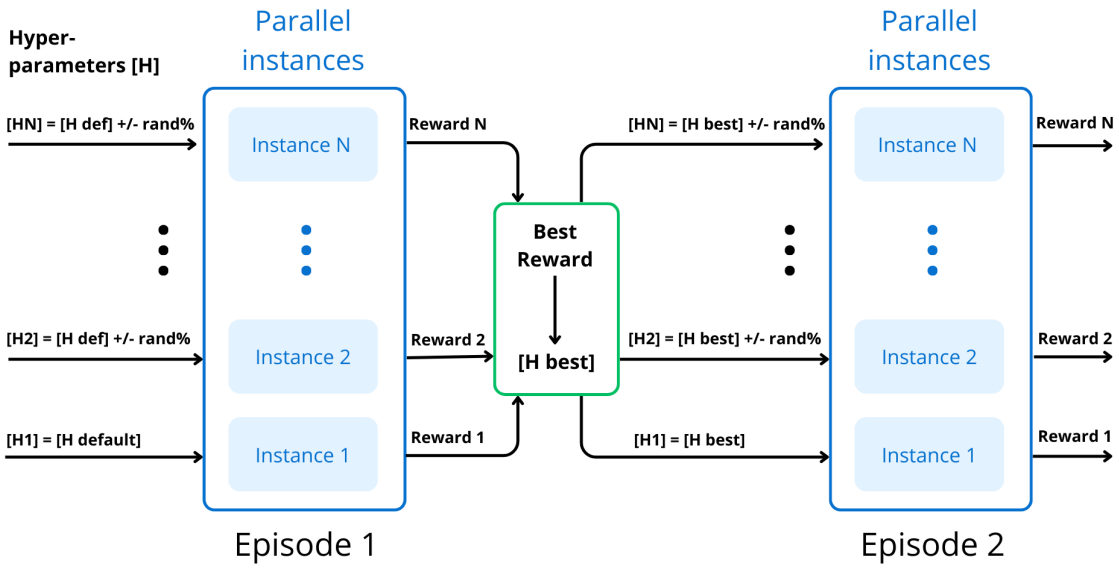


Figure 18: Population Based Training explanation sketch.

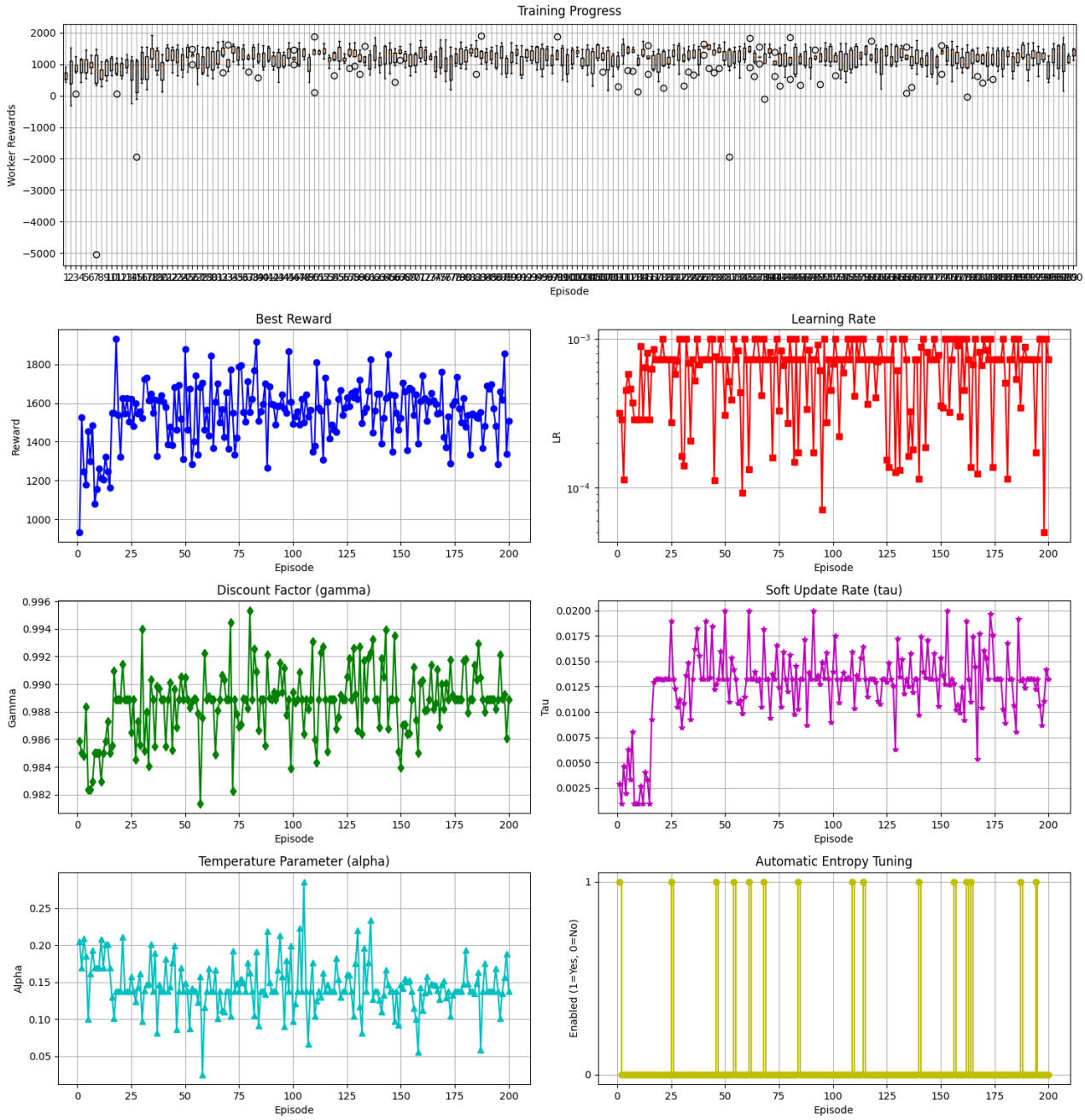


Figure 19: 200 episode training using PBT.

In the experiments, PBT demonstrated interesting hyperparameter adaptation but failed to achieve the same level of performance as single episode training with fixed hyperparameters. While single model training typically converged at around 5000 reward per episode within 20-100 episodes, the PBT implementation plateaued at approximately 1600 reward per episode, as shown in Figure 19.

4.3 State Observability

State observability refers to how much of the environment's true state an agent can directly perceive. A system is fully observable if the agent can see the full state, in this case been $\{\theta_0, \theta_1, \dot{\theta}_0, \dot{\theta}_1\}$. If some parts of the state are hidden or noisy, it's partially observable. In this case, the real pendulum doesn't capture the full state, it just captures $\{\theta_0, \theta_1, \dot{\theta}_0\}$.

In addition, the level of state observability directly influences the choice of NN architecture. When the state is fully observable (Markovian), a simple feed-forward network—such as one with two layers and 256 hidden units each—can effectively capture the environment's dynamics. Conversely, if the state is only partially observable, the network must incorporate temporal context to account for unseen or historical information (Sutton and Barto, 2018, Section 17.3), which in modern deep learning is often implemented using RNNs or Transformer-based models. This alignment between state characteristics and network design is key to achieve robust performance in tasks like the inverted pendulum simulation.

There are several ways to face partial observability:

Estimation of not observable state variables.
The idea is to explicitly compute hidden state variables using observations and a model of the system dynamics. Then this estimation is used, along the observable state variables, to form a "full state". This is used as an input to the RL algorithm, which has been trained with full observability (see Figure 20).

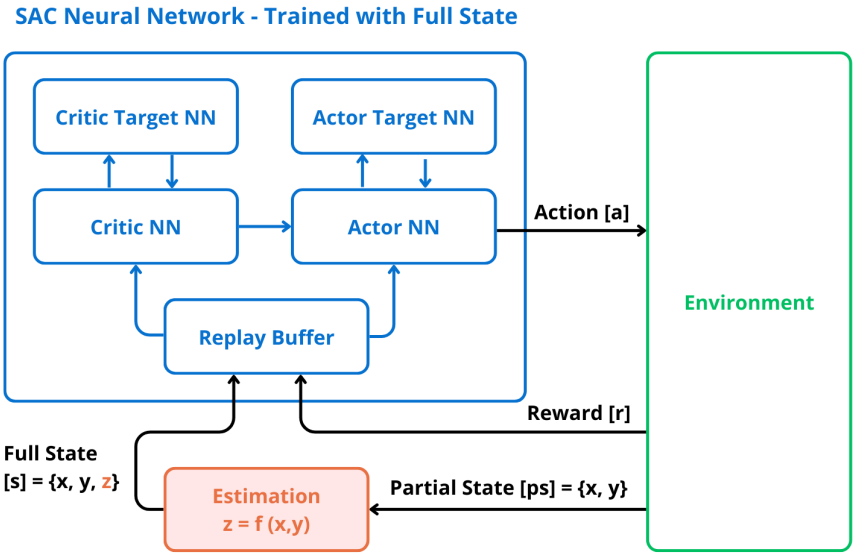


Figure 20: Estimation of non observable states.

In this case, the missing state variable $\dot{\theta}_1$ can be estimated as:

$$\hat{\dot{\theta}}_1 = \frac{\theta_1(t) - \theta_1(t-1)}{dt} \tag{43}$$

This method works, but can give problems if $dt = 0$ or if there are continuity issues with θ_1, like for example if θ_1 is clipped between -180º and 180º. However, this problem does not happen in the real pendulum, as the encoders provide continuous angle value (not clipped). See the result in Figure 21.

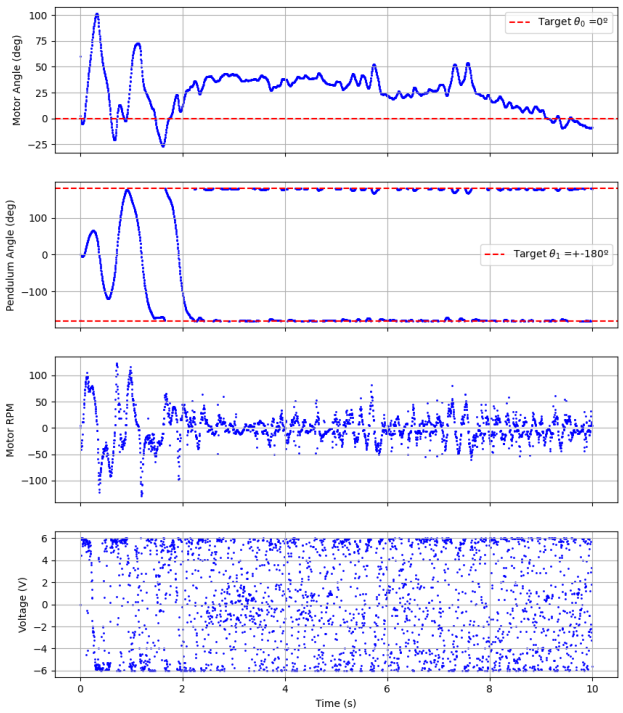


Figure 21: SAC with full observability, frame history = 1, 20% randomization and state estimation in the real pendulum.

RNN for estimation of not observable states.
This idea is closely related to the previous one, as a RNN is used to estimate the missing state variables. The RNN processes the observation history (past angles) and maintains a hidden state that encodes trends or unobservable variables. Unlike explicit estimation, it learns this from data during training, no hand-crafted filter needed (see Figure 22).

This method has not been tried, but it has been found interesting to mention.

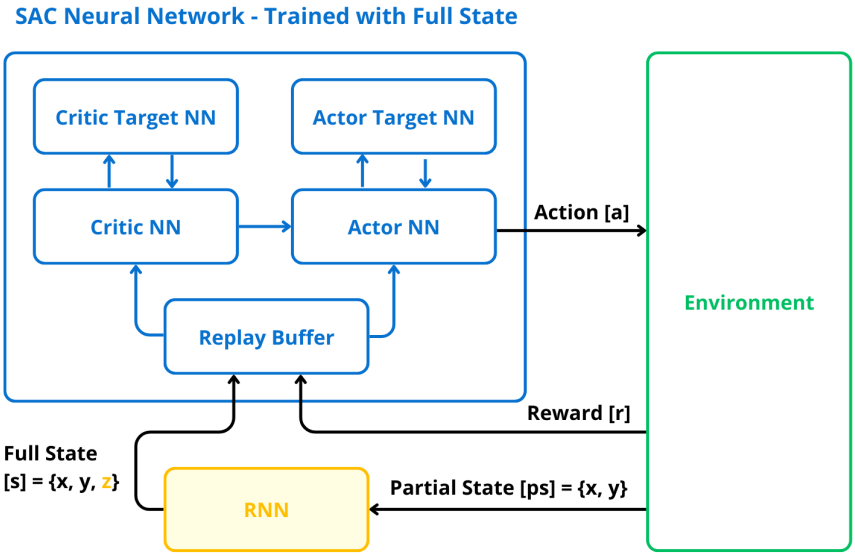


Figure 22: Estimation of non observable states with RNN.

Training with partial observable state.
This idea is to train the RL agent directly on the partial observations, forcing it to adapt without ever seeing the full state (see Figure 23.

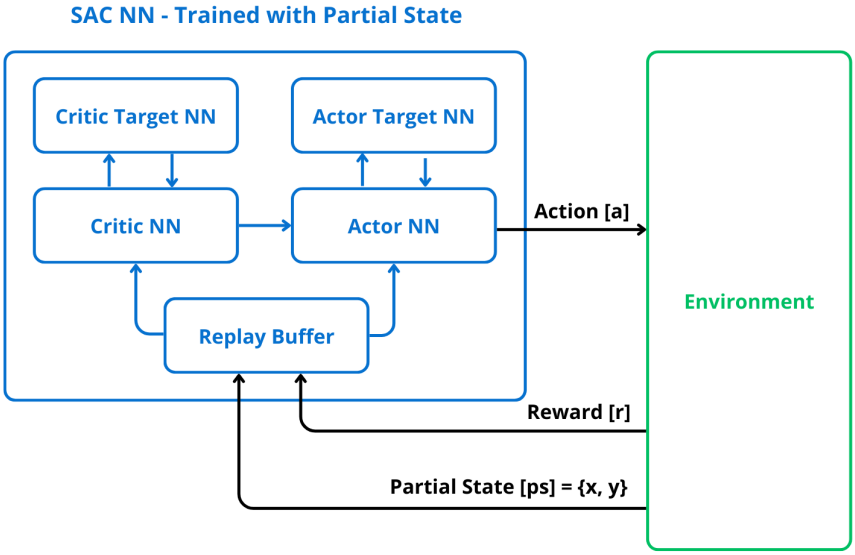


Figure 23: Training with partial observable state.

As it can be seen in Figure 24, the model trained with this method didn't achieve the inverted

equilibrium with the pendulum.

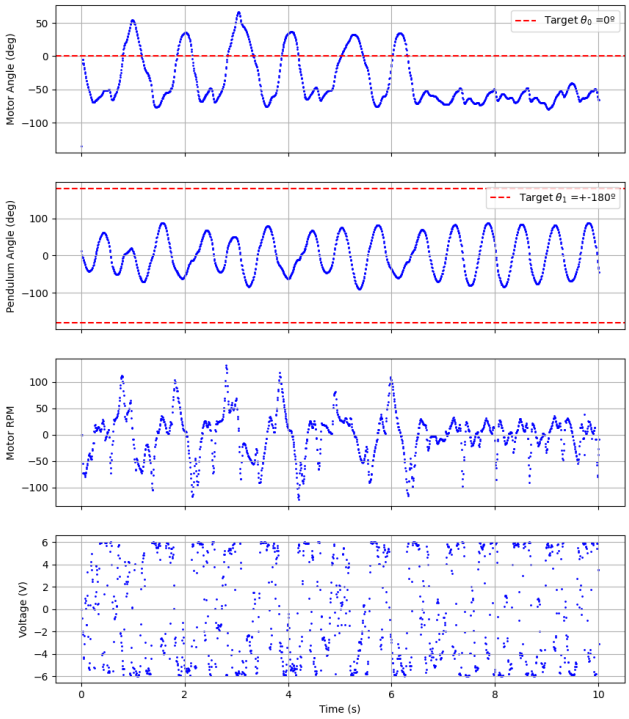


Figure 24: SAC with partial observability and frame history = 1 in real pendulum.

Using frame history.
The idea here is to use a sequence of past partial observations as an input to the NN, instead of using just one frame as in the previous method. For example, if using a frame history = 2, the state would be defined as $\{\theta_0(t), \theta_1(t), \dot{\theta}_0(t), \theta_0(t-1), \theta_1(t-1), \dot{\theta}_0(t-1)\}$. See Figure 25 for a visual representation.

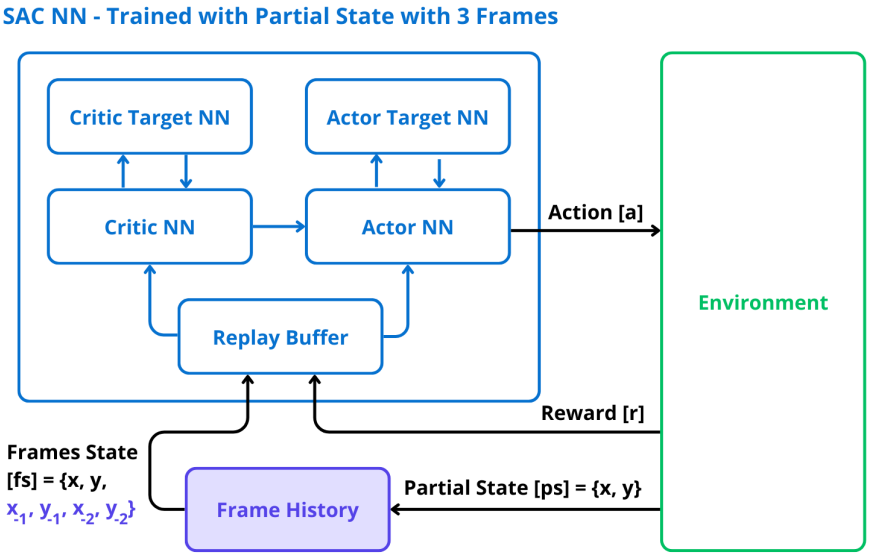
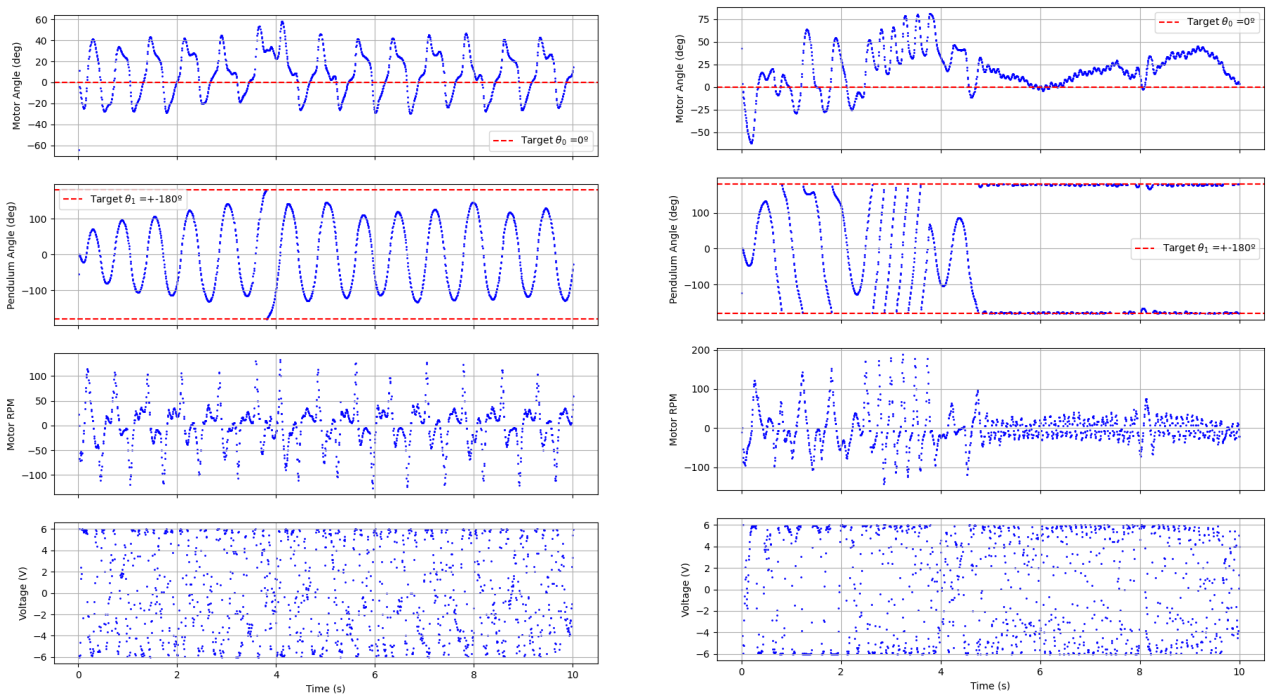


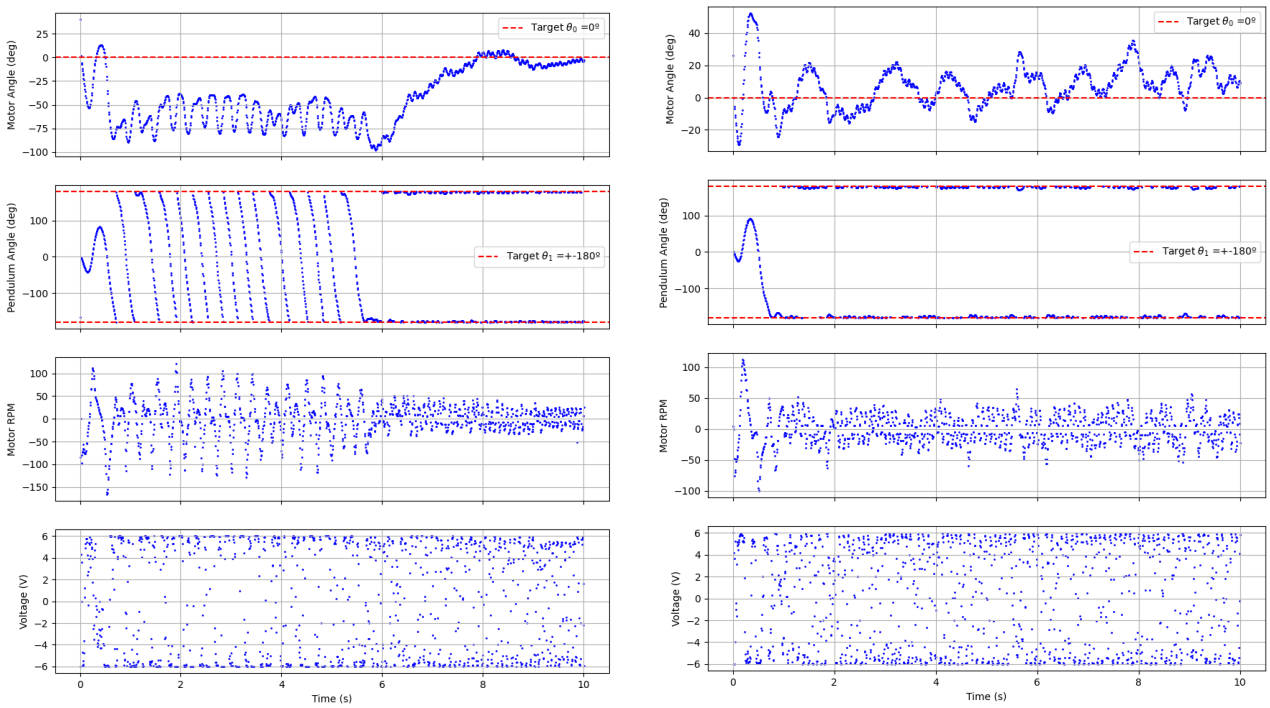
Figure 25: Training using partial observable state with frames.

As observed in Figures 26 and 27, results improve as the frame history increases. However, even the best model (Figure 27b) was quite unstable, as the motor angle was moving up and down around $\theta_0 = 0$.



(a) (b)

Figure 26: SAC with partial obs. and frame history (a) = 2; (b) = 3 in real pendulum.



(a) (b)

Figure 27: SAC with partial obs. and frame history (a) = 4; (b) = 5 in real pendulum.

4.4 Communication frequency

The communication frequency with the real pendulum defines a different time step value (δt) for each time step. This value must be similar to the value of δt used while training. If not, the model might not control the pendulum as expected.

The average communication frequency between the computer and the pendulum is between 1000 Hz and 2000 Hz ($\delta t = 0.001 sec$). Using this δt value for training, resulted in slow training and not convergent results (see Figure 28)

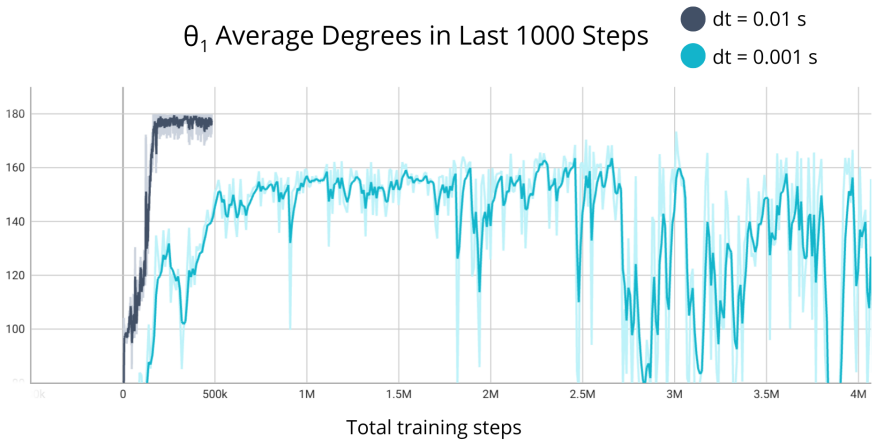


Figure 28: Training SAC model with full observability, frames history = 1, 20% randomization and different δt.

To solve this problem, a bigger time step $\delta t = 0.01sec$ was used for training, which provided faster training and convergence (see Figure 28).

In this way, the communication frequency between the computer and the real pendulum needed to be slowed down to 100 Hz ($\delta t = 0.01sec$), to align with training.

Finally, Figure 29 shows the results for both time step values used in the real pendulum.

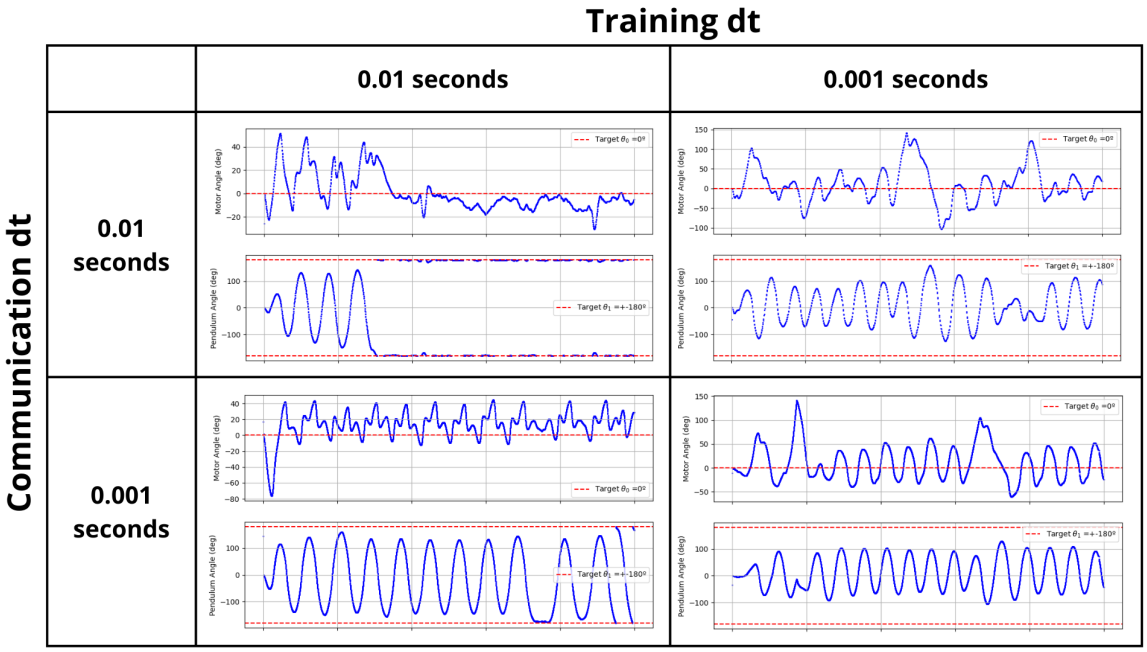


Figure 29: SAC model with full observability, frames history = 1, 20% randomization and different δt for training and communication in real pendulum.

4.5 RL output Filter

The trained RL controllers produced highly variable output signals. This is because RL policy outputs actions are based on learned neural network parameters that respond to state changes with rapid adjustments. When implemented on physical hardware, these abrupt changes in control signals lead to several issues:

1. Mechanical stress on motor components
2. Excessive power consumption
3. Audible noise during operation
4. Potential instability in the control system

To address these challenges, a first-order low-pass filter was implemented to smooth the RL controller's voltage output. The filter is described by the difference equation:

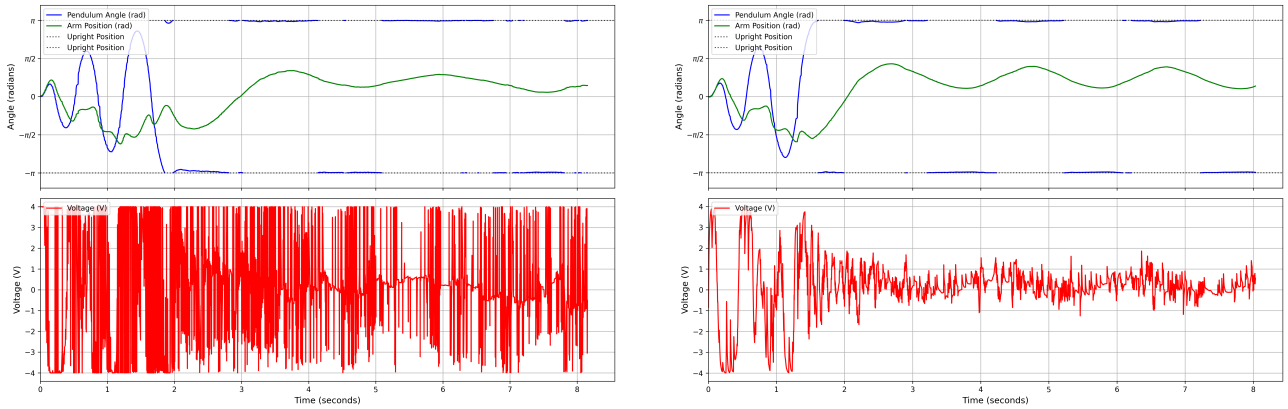
$$y_k = y_{k-1} + f_c\,\delta t\left(x_k - y_{k-1}\right) \tag{44}$$

Where:

- y_k is the filtered output at time step k.
- y_{k-1} is the previous filtered output (at time step $k-1$).
- x_k is the raw input at time step k.
- f_c is the cutoff frequency parameter (in Hz).
- δt is the time step duration.

The cutoff frequency f_c determines how aggressively the filter smooths the signal. Lower values produce smoother outputs but introduce more delay, while higher values allow faster response but with less smoothing.

After extensive testing with various cutoff frequencies ranging from 80Hz to 1000Hz, optimal performance was achieved with a cutoff frequency of 400Hz. This value provided the best balance between noise reduction and responsive control for the inverted pendulum.



(a) (b)

Figure 30: RL controller performance at 4 volts limit without filter(a) and with filter(b).

Figure 30 compares the controller performance with and without filtering. Without filtering Figure 30a, the RL controller produces highly oscillatory voltage commands with rapid fluctuations between maximum and minimum values, causing noticeable vibration and noise in the hardware. With the 400Hz filter applied Figure 30b, the control signal is significantly smoother while still maintaining the responsiveness needed to stabilize the pendulum.

The filtered control signal not only improves the auditory experience of the system by reducing mechanical noise but also extends the potential lifespan of the hardware components by reducing mechanical wear from high-frequency oscillations.

4.6 Performance of RL vs PID

This section presents an evaluation of the performance of PID controller with an integrated swing-up algorithm, implemented in Arduino code given by teacher, in comparison with a reinforcement learning (RL) control algorithm. The maximum voltage for the PID controller is regulated by modifying the parameter u_max which directly affects the applied voltage according to the equation

$$\text{voltage} = u_{\text{sat}} \cdot \frac{8.4 \times 0.095 \times 0.085}{0.042} \tag{45}$$

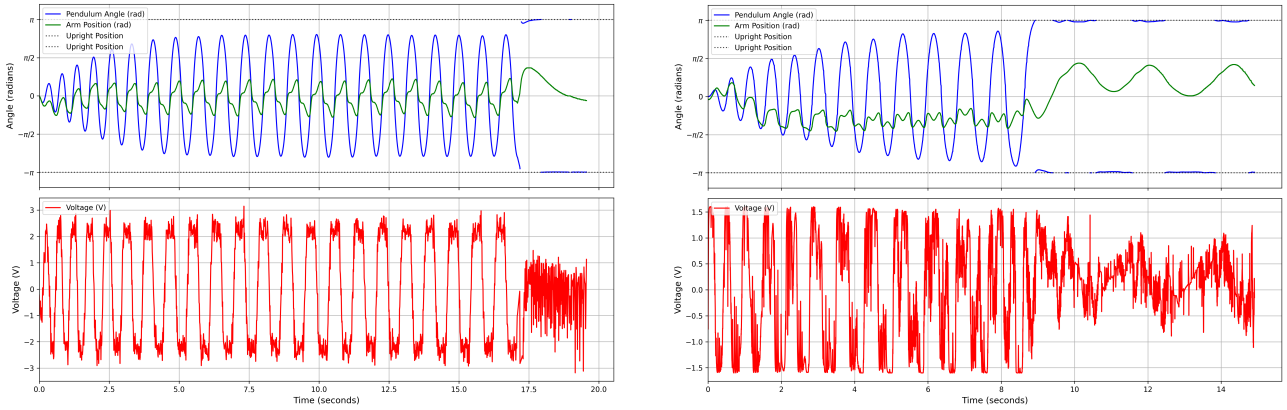
where

$$u_{\text{sat}} = \min\Big(u_{\max},\ \max\big(-u_{\max},\ u\big)\Big). \tag{46}$$

In contrast, the maximum voltage for the RL algorithm is controlled by adjusting the output multiplier. As the RL algorithm consistently produces outputs within the range $[-1, 1]$, scaling these outputs directly modifies the maximum voltage that can be applied.

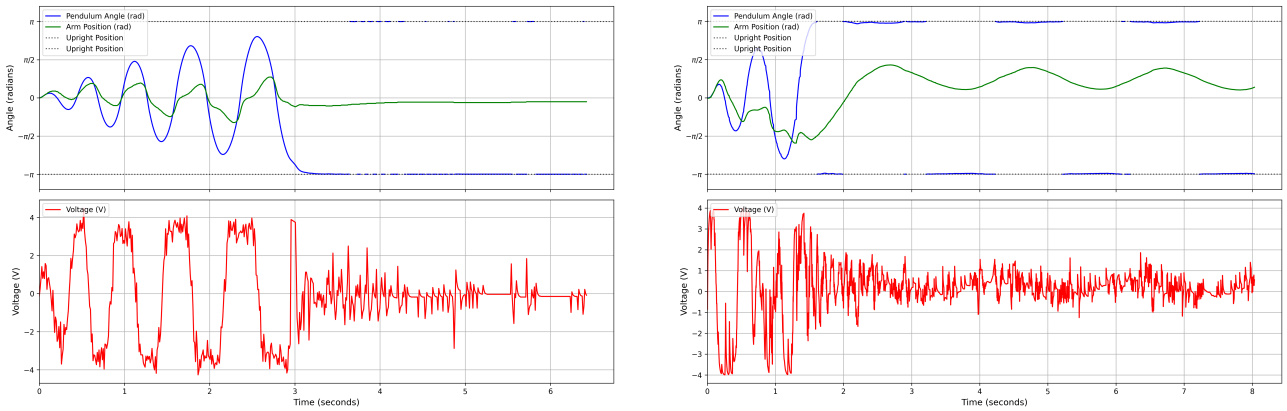
4.6.1 Performance at Different Voltage limits

The performance comparison focuses on determining the minimum allowable voltage at which each controller is capable of inverting the pendulum and relative performance with the same voltage limits. Figures 31, 32, and 33 illustrate the behavior of the controllers under low, medium, and high voltage limits, respectively.



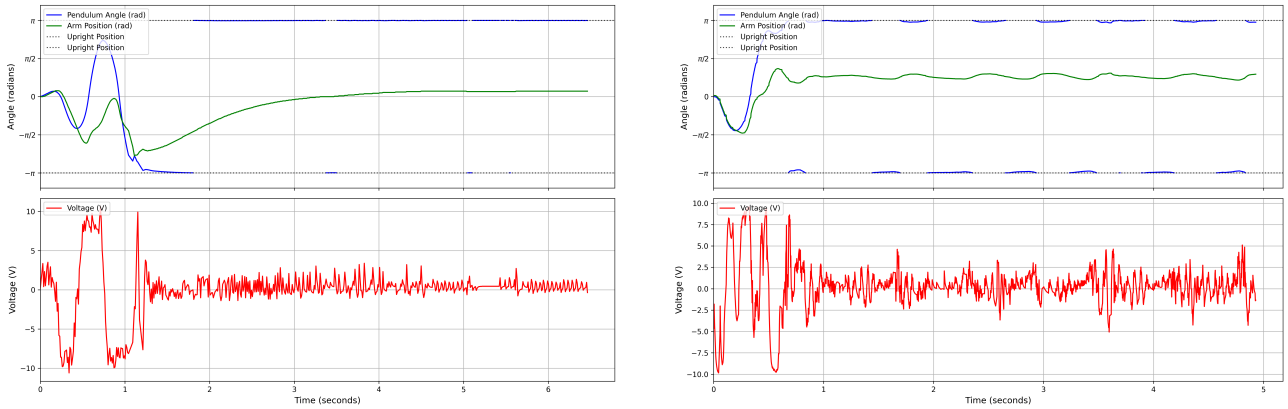
(a) (b)

Figure 31: (a) PID controller and (b) RL controller performance at low voltage limits.



(a) (b)

Figure 32: (a) PID controller and (b) RL controller performance at medium voltage limits.



(a) (b)

Figure 33: (a) PID controller and (b) RL controller performance at high voltage limits.

At the lowest voltage limit (Figure 31), the RL algorithm inverts the pendulum in approximately 9 seconds using no more than 1.6 volts, while the swing-up and PID controller requires about 17 seconds and reaches up to 2.5 volts.

Under a 4-volt limit (Figure 32), the RL algorithm demonstrates a response time nearly twice as fast as that of the Arduino controller, achieving the inverted position in roughly 1.5 seconds.

At the high voltage limit (Figure 33), the RL algorithm again outperforms the PID approach, requiring only a single swing to invert the pendulum as opposed to two swings by the PID controller. Additionally, the PID controller hits a mechanical constraint on the arm at approximately 2.37 radians.

Overall, both control strategies successfully invert the pendulum. The RL controller demonstrates faster inversion times and lower voltage requirements. However, its voltage output is considerably noisier, even after filtering. In contrast, the PID controller exhibits superior performance in stabilizing the arm at the zero position. The RL algorithm occasionally settles with an offset from the zero position and may exhibit oscillatory behavior.

It may be noted that the RL models for the 0–8 V and 8–12 V ranges were different. One was trained on voltage 4 volts while other was trained on 10 volts.

4.7 Alternatives if it didn't work

Finally, in this chapter, some ideas are included to implement in the case none of the previous would have worked.

- To make the model more robust against external forces, random pendulum velocities could be added while training.
- Collect data from the real pendulum using a RL model that doesn't achieve equilibrium.

Save this dataset as a $< state(i)action(i), reward(i), state(i+1) >$ sequence, and train a new model using this data.

- Train in the real pendulum straight forward using a new non-trained model, without using a virtually trained model for data collection as in the previous idea. This could be dangerous at the beginning, when the model is exploring new actions, as it could harm the pendulum if it uses a very high voltage.

5 Conclusion

This project successfully demonstrated the application of reinforcement learning (RL) to control the QUBE-Servo 2 inverted pendulum, achieving stabilization in both simulated and real-world environments.

The Soft Actor-Critic (SAC) algorithm, trained in a virtual simulation derived from Lagrangian mechanics, outperformed Proximal Policy Optimization (PPO) due to its off-policy efficiency and entropy-driven exploration.

The carefully designed reward function, incorporating upright positioning, arm centering, and energy considerations, significantly improved stabilization smoothness compared to simpler alternatives.

Techniques such as domain randomization, a tailored communication frequency, and a low-pass filter effectively bridged the sim-to-real gap, enabling robust performance on the physical pendulum.

Comparative analysis revealed that the RL controller achieved faster inversion times and lower voltage requirements than the PID controller across various voltage limits, though it exhibited noisier outputs and occasional arm offset.

While partial observability posed challenges, state estimation proved effective, and frame history offered limited improvements.

Future work could explore direct real-world training with safety constraints or incorporate recurrent neural networks for enhanced state estimation.

Overall, this study underscores the potential of RL for complex control tasks, providing a foundation for further advancements in intelligent machine applications.

A Video Demonstrations

This appendix contains links to video demonstrations of the inverted pendulum control.

A.1 Pendulum Performance with filter, with 4V limit

Video link: https://www.youtube.com/shorts/x7c1cgU1H0Y

A.2 Pendulum Performance without filter, with 10V limit

Video link: https://www.youtube.com/shorts/QIQI3dzDD88

A.3 Pendulum Performance without filter, with 2V limit

Video link: https://www.youtube.com/shorts/1P-MtejzHes

References

AntonioCruz, Mayra et al. (2015). 'Modeling, simulation, and construction of a Furuta pendulum test-bed'. In: *Proceedings of the 2015 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. Cholula, Mexico, pp. 72–79. DOI: 10.1109/CONIELECOMP.2015.7086928.

Le, Hoai Nam et al. (2020). 'System identifications of a 2DOF pendulum controlled by QUBE-servo and its unwanted oscillation factors'. In: *Archive of Mechanical Engineering* 67.4, pp. 435–450. DOI: 10.24425/ame.2020.131699.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. Cambridge, Massachusetts: The MIT Press.