Acrobot swing-up and stabilization

Riccardo De Vidi - 2152869

Abstract

An acrobot is a double pendulum with an active elbow motor. This report presents a comparison between the performance of an iLQR controller and of a DDPG controller capable of solving the acrobot swing-up and stabilization problem on a simulated plant.

1. Introduction

1.1. Experimental setup

A double pendulum is a serial robot composed of three links and two revolute kinematic pairs. A graphical representation of a double pendulum is reported in Figure 1.

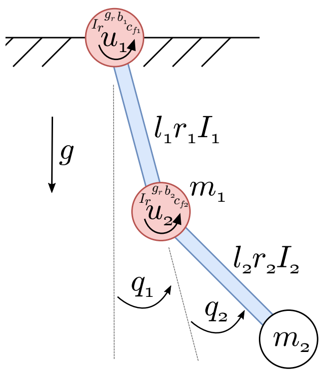

Figure 1. Double pendulum model.

A motor between the first and the second links of the kinematic chain is called shoulder motor, while a motor between the second and the third links of the kinematic chain is called elbow motor. $I_{1,2}$ are the first and the second link inertia, I_r is the shoulder and elbow motor inertia, $b_{1,2}$ are the joints viscous frictions, $c_{f1,f2}$ are the joints coulomb frictions, g is the gravity force, g_r is the motor gear ratio, $l_{1,2}$ are the links lengths, $m_{1,2}$ are the links masses, $r_{1,2}$ are the links center of masses, $q_{1,2}$ are the first and second joint angle from the free-hanging position according to the DH notation and $u_{1,2}$ are the torques applied by the motors. The values of the plant parameters that will be used are reported in Table 2. A double pendulum with only an elbow motor is called acrobot, and a double pendulum with only an active shoulder motor is called pendubot.

1.2. Actuation model of the system

Defining the generalized coordinates $q = [q_1, q_2]^T$, the system state vector $x = [q, \dot{q}]$, and the system input $u = [u_1, u_2]$, the dynamical model of the system can be written as

$$\dot{x} = \begin{bmatrix} \dot{q} \\ M^{-1}(Du - C(q,\dot{q})\dot{q} + G(q) - F(\dot{q})) \end{bmatrix},$$

with M mass matrix, C Coriolis matrix, G gravity vector and F friction vector defined as

$$M = \begin{bmatrix} I_1 + I_2 + l_1^2 m_2 + 2 l_1 m_2 r_2 c_2 + g_r^2 I_r + I_r & I_2 + l_1 m_2 r_2 c_2 - g_r I_r \\ I_2 + l_1 m_2 r_2 c_2 - g_r I_r & I_2 + g_r^2 I_r \end{bmatrix},$$

$$C = \begin{bmatrix} -2\dot{q}_2 l_1 m_2 r_2 s_2 & -\dot{q}_2 l_1 m_2 r_2 s_2 \\ \dot{q}_1 l_1 m_2 r_2 s_2 & 0 \end{bmatrix},$$

$$G = \begin{bmatrix} -g m_1 r_1 s_1 - g m_2 \left(l_1 s_1 + r_2 s_{1+2}\right) \\ -g m_2 r_2] s_{1+2} \end{bmatrix},$$

$$F = \begin{bmatrix} b_1 \dot{q}_1 + c_{f1} \arctan(100\, \dot{q}_1) \\ b_2 \dot{q}_2 + c_{f2} \arctan(100\, \dot{q}_2) \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

1.3. Swing-up and stabilization problem

The acrobot swing-up and stabilization problem consists in the design of a controller capable of swinging an acrobot from the free-hanging position $x = [0, 0, 0, 0]$ into the upright position $x = [\pi, 0, 0, 0]$ and keeping it in such a configuration. This problem is typically solved exploiting standard controllers such as the Linear Quadratic Regulator (LQR) or the Iterative Linear Quadratic Regulator (iLQR), an LQR extension to nonlinear dynamics.

This project proposes a controller designed exploiting Reinforcement Learning techniques to solve such a problem on a simulated plant.

1.4. Plant simulation

To simulate the plant, the implementation reported in `https://github.com/dfki-ric-underactuated-lab/double_pendulum` (Wiebe et al., 2024) is used. All simulations are performed using a step size of $5ms$, the Runge-Kutta

Table 1. Acrobot model parameters.

VARIABLE	PARAMETER	VALUE	
I_1, I_2	LINKS INERTIAS	$0.02, 0.05$	$[kg\ m^2]$
I_r	MOTOR INERTIA	$\approx 7.66e-05$	$[kg\ m^2]$
b_1, b_2	JOINTS VISCOUS FRICTIONS	$0.001, 0.001$	$[N]$
c_{f1}, c_{f2}	JOINTS COULOMB FRICTIONS	$0.05, 0.093$	$[N]$
g	GRAVITY	9.81	$[m/s^2]$
g_r	MOTOR GEAR RATIO	6.0	
l_1, l_2	LINKS LENGTHS	$0.1, 0.4$	$[m]$
m_1, m_2	LINKS MASSES	$0.2, 0.6$	$[kg]$
r_1, r_2	LINKS CENTER OF MASSES	$0.1, 0.4$	$[m]$
$t_{l1\pm}$	ELBOW MOTOR TORQUE LIMIT	± 0.0	$[Nm]$
$t_{l2\pm}$	SHOULDER MOTOR TORQUE LIMIT	± 10.0	$[Nm]$

integration method, an initial time of $0s$ and a final time of $5s$ (every simulation is 1000 steps long).

1.5. Machines used for the simulations

All the simulations are performed on a consumer laptop (OS: Windows 11, CPU: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, RAM: 16GB, GPU: none) on Docker Desktop.

2. iLQR controller design

2.1. Brief introduction to model predictive control

Model predictive control (MPC) (García et al., 1989) is an optimal control technique where the calculated control actions minimize a cost function over a finite and receding horizon. At each time step, an MPC controller receives or estimates the current state of the plant, calculates the sequence of control actions that minimizes the cost over the horizon by solving an optimization problem that relies on an internal plant model and depends on the current system state, and then applies to the plant only the first computed control action. In general, minimizing the cost function involves reducing the error between the future plant outputs and a reference trajectory.

2.2. iLQR controller

In an iLQR controller, fixed an initial state x_0, a cost matrix Q_f to penalize the final state, a cost matrix Q to penalize the intermediate states, a cost matrix R to penalize the system input signal and a function $f_d(\cdot)$ discretization of the system dynamics, the system input is computed as

$$argmin_{u_1,\ \ldots,\ u_{N-1}} \quad x_N^T Q_f x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i,$$

$$x_{i+1} = f_d(x_i, u_i).$$

In this setup, $x_0 = [0, 0, 0, 0]$, the chosen stage cost weights are

$$Q = Q_f = \begin{bmatrix} 0.1 & & & \\ 0 & 0.1 & & \\ 0 & 0 & 0.01 & \\ 0 & 0 & 0 & 0.1 \end{bmatrix},$$

the chosen final cost weights are

$$Q = \begin{bmatrix} 100 & & & \\ 0 & 10 & & \\ 0 & 0 & 10 & \\ 0 & 0 & 0 & 10 \end{bmatrix},\ Q_f = \begin{bmatrix} 10 & & & \\ 0 & 10 & & \\ 0 & 0 & 10 & \\ 0 & 0 & 0 & 10 \end{bmatrix},$$

$R = 0$ always and $f_d(\cdot)$ is given by the chosen implementation (see Section 1.4), as the whole control loop.

It is observed that at least one step of the controller synthesis introduces some sort of randomness in the obtained regulator. Given that, the results reported in the following sections refer to a specific iLQR regulator, and it is assumed that each iLQR controller generated by the chosen implementation guarantees similar performance.

Table 2. iLQR controller parameters.

PARAMETER	VALUE
HORIZON TIMESTEPS	100
TIMESTEP	$0.005s$
MAX NUMBER OF OPT. PER STEP	100
INITIAL REGULARIZATION	1
MAX. REGULARIZATION	10000
MIN. REGULARIZATION	$1e-6$
BREAK COST	$1e-6$
INTEGRATOR	RUNGE-KUTTA
TRAJECTORY STABILIZATION	TRUE
TIMESTEPS SHIFTING	1

3. Reinforcement learning controller design

3.1. Brief introduction to DDPG

Deep deterministic policy gradient (DDPG) (Lillicrap et al., 2019) is an actor-critic, model-free, off-policy reinforcement learning algorithm based on the deterministic policy gradient (DPG) algorithm that can be used over continuous action spaces. The algorithm learns a policy and a Q function at the same time using two different neural networks. Compared to other reinforcement learning algorithms, DDPG is found to be more sample efficient, but still requires a large number of episodes to tune the two neural networks. In general, DDPG robustly solves challenging problems in a wide variety of domains. As with most reinforcement learning algorithms, the use of neural networks as function approximators does not provide any convergence guarantees. In this setting, the implementation reported in `https://stable-baselines3.readthedocs.io/en/v1.0/modules/ddpg.html#ddpg-policies` is used.

3.2. Environments

Given a state vector $x = [q_1, q_2, \dot{q}_1, \dot{q}_2]$ that describes the acrobot in a given time instant of a simulation the observation processed by the DDPG algorithm to train the agent or predict an action is defined as $o = [o_1, o_2, o_3, o_4]$ with

$$o_1 = \frac{q_1\%(2\pi) - \pi}{\pi},\ o_2 = \frac{q_2\%(2\pi) - \pi}{\pi},$$

$$o_3 = \frac{\min(\max(\dot{q}_1, -max\_velocity), max\_velocity)}{max\_velocity},$$

$$o_4 = \frac{\min(\max(\dot{q}_2, -max\_velocity), max\_velocity)}{max\_velocity},$$

where $max\_velocity$ is the maximum absolute value that $\dot{q}_1$ and $\dot{q}_2$ can assume and it is fixed to $20rad/s$ (that it is assumed to be large enough). Note that with these transformations $x_o \in [-1;+1]^4$, the upright position $[\pi, 0, 0, 0]$ is mapped into $[0, -1, 0, 0]$ and the free-hanging position $[0, 0, 0, 0]$ is mapped into $[-1, -1, 0, 0]$. The action vector is defined as $a = [a_1, a_2] \in [-1;+1]^2$, and so $u = [u_1, u_2]$ is defined as

$$u_1 = t_{l1+}a_1,\ u_2 = t_{l2+}a_2.$$

Observe that to control the acrobot only the value of a_2 is relevant, since $t_{l1+} = 0$. To train the agent and validate/test the agent, two different environments have been used.

For what concerns the training environment, its dynamics function has been initialized with the acrobot one, its reward function has been chosen among the three functions reported in the following, its termination function has been set to `False`, its reset function has been set to return the free-hanging position state $[-1, -1, 0, 0]$, its maximum number of steps per episode has been set to 1000, and in it each action is affected by a zero mean Gaussian noise to enforce exploration. For what concerns the validation/test environment, it is completely equal to the training one, except that in it there is no action noise.

3.3. Reward functions

Three different reward functions have been used in the training of the agent:

- $r_1(x_o, a) = (1 - |o_1|)(|o_2|)(1 - |x_o|)(1 - |o_4|)(1 - |a_2|)$, which links the state of the system with the last action taken;
- $r_2(x_o) = (1 - |o_1|)(|o_2|)(1 - |o_3|)(1 - |o_4|)$, which is based only on the state of the system;
- $r_3(x_o) = (1 - |o_1|)(|o_2|)$, which is based on a reduced version of the state of the system.

The global maximum of all functions is reached when the plant is in the upright position, while all global minimums of the functions are reached when the plant is in the free-hanging position or when one component of the state is as far as it can be from the optimum. Moreover, for $r_1(\cdot)$, the global maximum is reached only when the torque applied by the active motor is zero. This is done to explicitly link the motor torque to the state of the system. The latter connection is also present in $r_2(\cdot)$ and $r_3(\cdot)$, since the upright position once reached does not require any torque to be maintained, but it is in some sense hidden. A similar reasoning can be used to justify the differences between the first two reward functions and $r_3(\cdot)$.

Each evaluation of $r_1(\cdot)$, $r_2(\cdot)$, and $r_3(\cdot)$ gives a value in the interval $[0, +1]$, so the total reward of each episode will be in the interval $[0, +1000]$ for all the reward functions. In the following, the agent trained with $r_1(\cdot)$ will be addressed as $DDPG_1$, the agent trained with $r_2(\cdot)$ will be addressed as $DDPG_2$, and the agent trained with $r_3(\cdot)$ will be addressed as $DDPG_3$.

3.4. Rewards obtained by the iLQR controller

It is possible to apply the three reward functions to the sequences of system states and control input that are obtained solving the problem with the iLQR controller. These results are reported in Table 3, and are useful for comparing how good a reinforcement learning agent learned with a certain reward function is with respect to the iLQR controller, taking as goodness measure the reward function.

Table 3. iLQR controller rewards.

REWARD FUNCTION	REWARD
$r_1(\cdot)$	759.4131
$r_2(\cdot)$	773.6405
$r_3(\cdot)$	804.6738

3.5. Training

The same training procedure has been performed for each reward function. Each training is performed in a vectorized environment with 100 training environments to speed up the process. All training parameters are reported in Table 4. The results of the evaluations performed during the training are reported in Figure 2. For each reward function the agent that obtained the best score during evaluation is kept.

Table 4. Training parameters.

PARAMETER	VALUE
POLICY	$MlpPolicy$
ENVIRONMENTS	100
EPISODE LENGTH	1000 STEPS
LEARNING STARTS	100 EPISODES
TRAIN FREQUENCY	1 STEP
TRAINING EPISODES	20000
LEARNING RATE	0.001
EVALUATION FREQUENCY	2000 STEPS
EVALUATION EPISODES	1
ACTION NOISE	$N(0, 0.01)$

Notice that for every reward function the training procedure finds an agent that performs better than the iLQR controller with respect to that function. Note that this does not mean that all found agents will solve the swing-up and stabilization problem better than the iLQR controller.

Moreover, note that, while for $r_3(\cdot)$ the evaluations are stable after 500 steps, such a thing is not observable for $r_1(\cdot)$ and $r_2(\cdot)$. In fact, note that the evaluations of $r_1(\cdot)$ and $r_2(\cdot)$ keep changing significantly over time. The latter suggests that some of the chosen training parameters are not optimal for the training of the agent with all the reward functions, i.e. the learning rate could be reduced. It is significant to highlight that a different set of training parameters could bring to better agents.

4. Results

In the following, the results obtained by the iLQR controller and the reinforcement learning controllers are compared.

The values used for the comparison are the steady state error (measured for $t = 5s$) and the 1%, 5% and 10% settling times for all the values of the state. The choice of settling

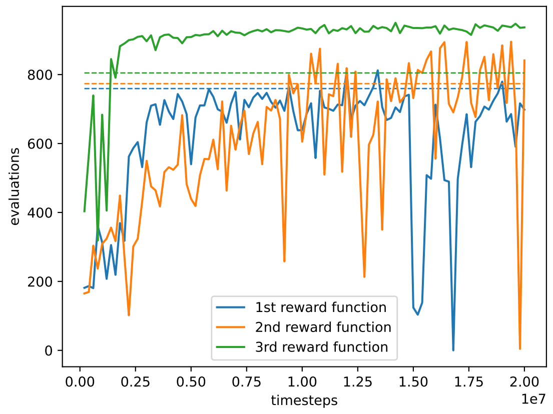

Figure 2. Training evaluations. The results corresponding to the RL agents are reported with solid lines, the horizontal dashed lines correspond to the iLQR controller rewards (also reported in Table 3)

times as a measure to compare the controllers is given by the fact that they capture both how fast the controllers are capable to swing-up the acrobot and if the controllers are capable to stabilize the system around the upright position. The steady state error is chosen to give another measure about the latter argument.

4.1. iLQR controller

The controller is capable of solving the problem, the behavior of the system with the controller is reported in Figure 3 and the performance of the controller are reported in Table 5.

Table 5. iLQR performance.

	t_{s1}	t_{s5}	t_{s10}	ERROR
q_1	$2.455s$	$1.875s$	$1.53s$	$1.9e-3rad$
q_2	$3.135s$	$2.33ss$	$1.575s$	$3.7e-3rad$
$\dot{q}_1$	$3.315s$	$2.435s$	$2.015s$	$5e-3rad/s$
$\dot{q}_2$	$4.97s$	$2.665s$	$2.425s$	$7.8e-3rad/s$

4.2. DDPG controllers

As can be observed in Figures 4, 5 the controllers $DDPG_1$ and $DDPG_3$ are not capable of properly solving the problem. The $DDPG_1$ controller is able to swing-up the system, but it is not capable of keeping it in the upward position avoiding unwanted wobbles. The $DDPG_3$ controller is able to swing-up the system, but it is not capable of keeping it in the upward position with velocities close to zero. For the $DDPG_1$ controller it is not possible to evaluate all the defined metrics, since two state values keep changing

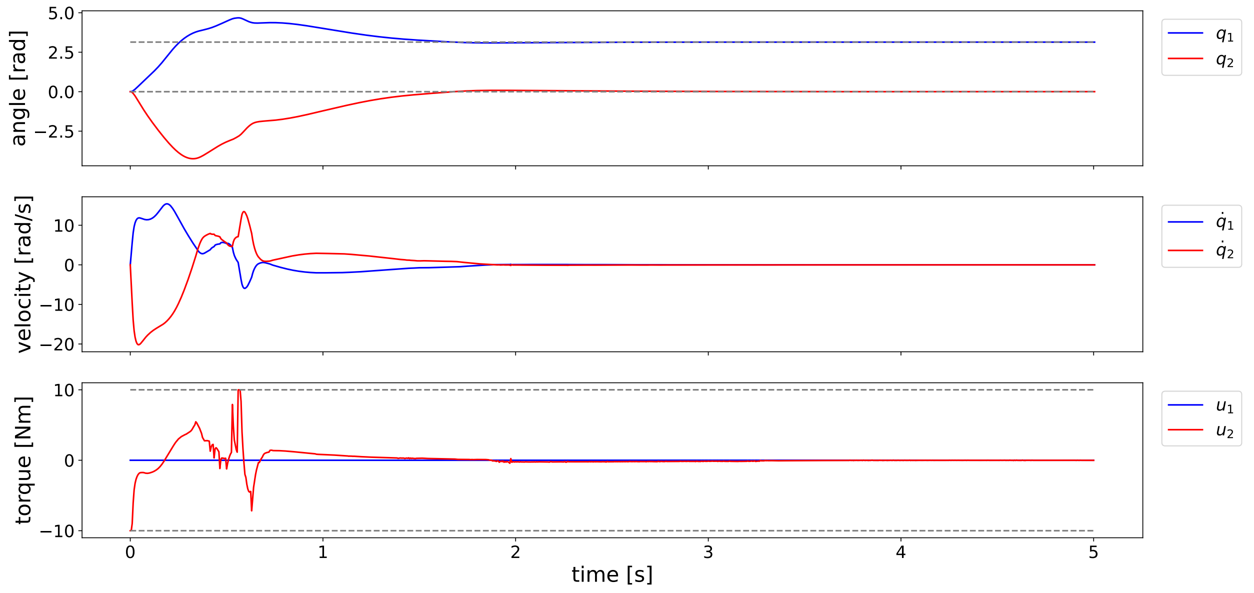

Figure 3. iLQR timeseries

over time, while for the $DDPG_3$ controller it is possible to define the metrics only for the first two components of the state, i.e. the positions. The results are reported in Tables 6, 7.

Table 6. $DDPG_1$ performance.

	t_{s1}	t_{s5}	t_{s10}	ERROR
q_1	$\times$	$\times$	$\times$	$\times$
q_2	$4.92s$	$4.575s$	$4.53s$	$\times$
$\dot{q}_1$	$\times$	$\times$	$\times$	$\times$
$\dot{q}_2$	$4.995s$	$4.945s$	$4.905s$	$\times$

Table 7. $DDPG_3$ performance.

	t_{s1}	t_{s5}	t_{s10}	ERROR
q_1	$\times$	$1.08s$	$1.04s$	$12e-3rad$
q_2	$1.445s$	$1.095s$	$1.085s$	$0.5e-3rad$
$\dot{q}_1$	$\times$	$\times$	$\times$	$\times$
$\dot{q}_2$	$\times$	$\times$	$\times$	$\times$

On the other hand, the controller $DDPG_2$ is able to solve the problem. The behavior of the system with the $DDPG_2$ controller is reported in Figure 6 and the performance of the controller are reported in Table 8.

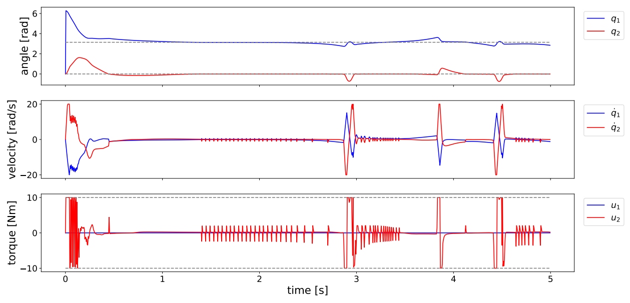

Figure 4. $DDPG_1$ timeseries

Table 8. $DDPG_2$ performance.

	t_{s1}	t_{s5}	t_{s10}	ERROR
q_1	$4.6s$	$1.54s$	$0.53s$	$10e-3rad$
q_2	$\times$	$3.105s$	$1.035s$	$0.5e-3rad$
$\dot{q}_1$	$3.53s$	$0.59s$	$0.565s$	$0.2rad/s$
$\dot{q}_2$	$\times$	$0.64s$	$0.565s$	$0.3rad/s$

5. Conclusions

Both the iLQR and $DDPG_2$ controllers are capable of solving the problem. The iLQR controller obtains smaller t_{s1} for all the metrics, but the $DDPG_2$ controller obtains smaller t_{s5} for three metrics out of four and significantly smaller t_{s10} for all the values. The controller obtains comparable

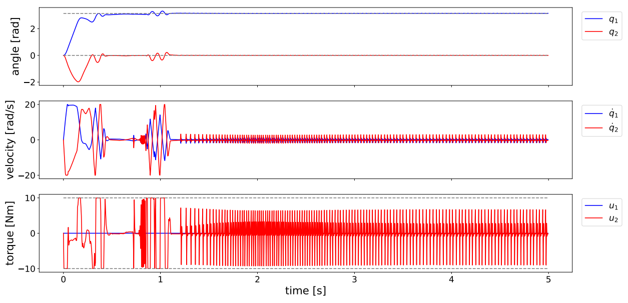

Figure 5. $DDPG_3$ timeseries

asymptotic errors. Focusing only on the first two components of the state, also the $DDPG_3$ controller performs better than the iLQR controller for both t_{s5} and t_{s10} and obtains comparable asymptotic errors.

The difference between the performance of $DDPG_2$ and $DDPG_3$ underlines that $r_3(\cdot)$ is in some sense less informative than $r_2(\cdot)$ to solve the problem. It is important to note that with a longer training phase maybe also an agent obtained using $r_3(\cdot)$ could be capable of scoring values similar to the ones of $DDPG_2$.

The difference between the scores of $DDPG_2$ and $DDPG_1$ shows that $r_1(\cdot)$ is also, in some sense, less informative than $r_2(\cdot)$ to solve the problem, or it is too complex to obtain an acceptable result in a training phase of this length.

References

García, C. E., Prett, D. M., and Morari, M. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(89)90002-2. URL https://www.sciencedirect.com/science/article/pii/0005109889900022.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971.

Wiebe, F., Kumar, S., Shala, L. J., Vyas, S., Javadi, M., and Kirchner, F. Open source dual-purpose acrobot and pendubot platform: Benchmarking control algorithms for underactuated robotics. *IEEE Robotics & Automation Magazine*, 31(2):113–124, 2024. doi: 10.1109/MRA.2023.3341257.

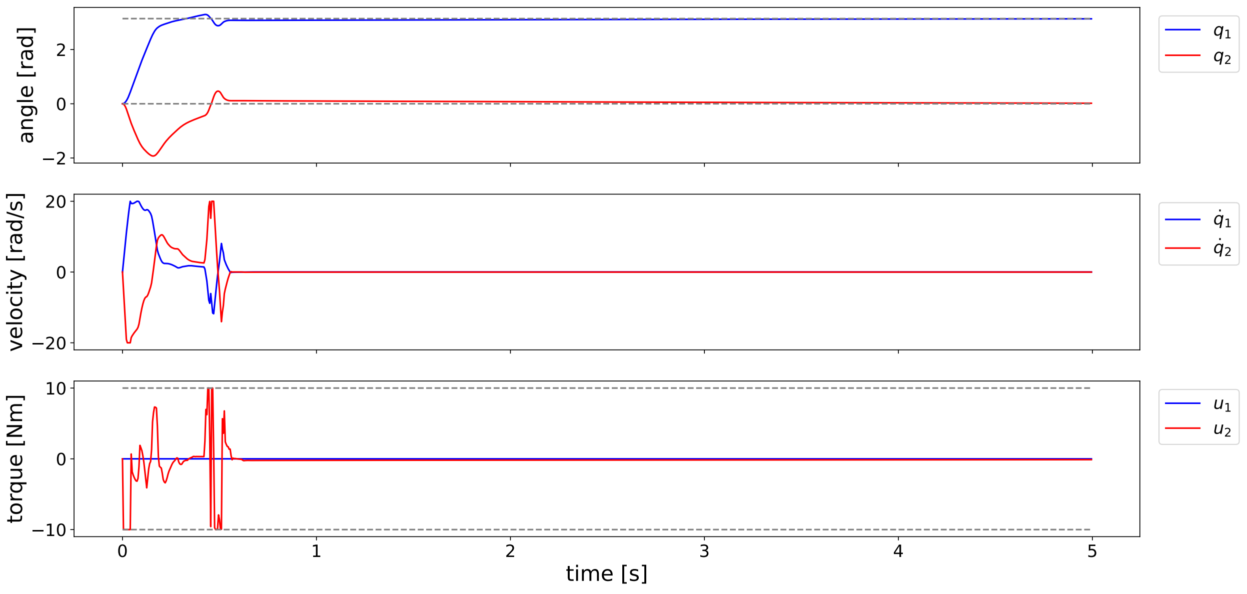

Figure 6. $DDPG_2$ timeseries