SDM366 Report for Project 2

Authors: Zhiling Li 12112748, Jinda Dong 12111829

Task 1

According to the description, the dynamics of the system (shown in Figure.1) are given by:

$$\begin{aligned}\ddot{z} &= \frac{1}{m_c + m_p \sin^2\theta}\left[u + m_p \sin\theta\left(L\dot{\theta}^2 + g\cos\theta\right)\right]\\ \dot{\theta} &= \frac{1}{L(m_c + m_p \sin^2\theta)}\left[-u\cos(\theta) - m_p L\dot{\theta}^2\cos\theta\sin\theta - (m_c+m_p)g\sin\theta\right]\end{aligned} \tag{1}$$

where $m_p = 1$, $m_c = 10$, $g = 9.81$, and $L = 0.5$.

define the state variable x as:

$$x = [z, \pi - \theta, \dot{z}, \dot{\theta}]^T$$

then, we can know that:

$$\begin{aligned}\dot{x}_1 &= x_3\\ \dot{x}_2 &= -x_4\\ \dot{x}_3 &= \frac{1}{m_c + m_p\sin^2(x_2)}[u + m_p\sin(x_2)(Lx_4^2 - g\cos(x_2))]\\ \dot{x}_4 &= \frac{1}{L(m_c + m_p\sin^2(x_2))}[u\cos(x_2) + m_pLx_4^2\sin(x_2)\cos(x_2) - (m_c+m_p)g\sin(x_2)]\end{aligned} \tag{2}$$

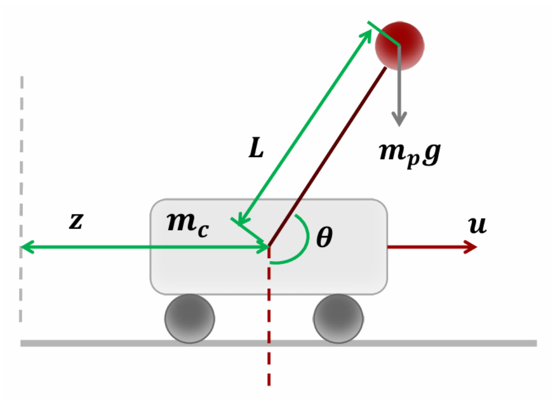
Finally, we get the continuous time state space model:

$$\dot{x} = f(x,u) = \begin{bmatrix} x_3 \\ -x_4 \\ \frac{u+m_p\sin(x_2)(Lx_4^2 - g\cos(x_2))}{m_c+m_p\sin^2(x_2)} \\ \frac{u\cos(x_2)+m_pLx_4^2\sin(x_2)\cos(x_2)-(m_c+m_p)g\sin(x_2)}{L(m_c+m_p\sin^2(x_2))}\end{bmatrix} \tag{3}$$

Task 2

Consider $\hat{x} = [0,0,0,0]^T$, we're going to find the partial derivative for both x and u. The jacobian matrix of $\frac{\partial f(x,u)}{\partial x}$ is:

$$\hat{A} = \frac{\partial f(x,u)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4}\end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & \frac{\partial f_3}{\partial x_2} & 0 & \frac{\partial f_3}{\partial x_4} \\ 0 & \frac{\partial f_4}{\partial x_2} & 0 & \frac{\partial f_4}{\partial x_4}\end{bmatrix} \tag{4}$$



t

Figure 1: The inverted pendulum model in this experiment.

Where:

$$
\begin{aligned}
\frac{\partial f_3}{\partial x_2} =& \frac{-2m_p \sin x_2 \cos x_2}{(m_c + m_p \sin^2 x_2)^2} \cdot [u + m_p \sin x_2 (L x_4^2 - g \cos x_2)] \\
&+ \frac{1}{m_c + m_p \sin^2 x_2} \cdot [m_p \sin x_2 (g \sin x_2) + m_p \cos x_2 (L x_4^2 - g \cos x_2)] \\
\frac{\partial f_3}{\partial x_4} =& \frac{2}{m_c + m_p \sin^2 x_2} (m_p \sin x_2 L x_4) \\
\frac{\partial f_4}{\partial x_2} =& \frac{-2m_p \sin x_2 \cos x_2}{L(m_c + m_p \sin^2 x_2)^2} \cdot [u \cos x_2 + m_p L x_4^2 \sin x_2 \cos x_2 - (m_c + m_p) g \sin x_2] \\
&+ \frac{1}{L(m_c + m_p \sin^2 x_2)} \cdot [-u \sin x_2 + m_p L x_4^2 (\cos^2 x_2 - \sin^2 x_2) - (m_c + m_p) g \cos x_2] \\
\frac{\partial f_4}{\partial x_4} =& \frac{2}{L(m_c + m_p \sin^2 x_2)} \cdot m_p L x_4 \sin x_2 \cos x_2
\end{aligned}
\tag{5}
$$

Similarly, we get $\hat{B}$:

$$
\hat{B} = \frac{\partial f(x,u)}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{u} \\ \frac{\partial f_2}{u} \\ \frac{\partial f_3}{u} \\ \frac{\partial f_4}{u} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c + m_p \sin^2(x_2)} \\ \frac{1}{L} \cdot \frac{1}{m_c + m_p \sin^2(x_2)} \end{bmatrix} \tag{6}
$$

Applying $\hat{x} = [0,0,0,0]^T$, we get:

$$
\hat{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -\frac{m_p g}{m_c} & 0 & 0 \\ 0 & -\frac{(m_c + m_p) g}{L m_c} & 0 & 0 \end{bmatrix}, \quad \hat{B} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c} \\ \frac{1}{L m_c} \end{bmatrix} \tag{7}
$$

If we use the parameters in Table.1, we can get the linearized continuous time model around $\hat{x}$ in (8).

parameter	m_p	m_c	g	L
value	1(kg)	10(kg)	$9.81(m/s^2)$	0.5(m)

Table 1: Values for the parameters.

$$\hat{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -0.981 & 0 & 0 \\ 0 & -21.582 & 0 & 0 \end{bmatrix}, \quad \hat{B} = \begin{bmatrix} 0 \\ 0 \\ 0.1 \\ 0.2 \end{bmatrix} \tag{8}$$

Task 3

Assuming that the sampling time duration is ΔT. Now we have:

$$\begin{aligned} \frac{x[k+1] - x[k]}{\Delta T} &= \hat{A}x[k] + \hat{B}u[k] \\ x[k+1] &= (\hat{A}\Delta T + I)x[k] + \hat{B}\Delta T u[k] \end{aligned} \tag{9}$$

That is,

$$\hat{A}_d = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & -\Delta T \\ 0 & -\frac{m_p g}{m_c}\Delta T & 1 & 0 \\ 0 & -\frac{(m_c+m_p)g}{Lm_c}\Delta T & 0 & 1 \end{bmatrix}, \quad \hat{B}_d = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c}\Delta T \\ \frac{1}{Lm_c}\Delta T \end{bmatrix} \tag{10}$$

Also, taking the parameters from Table.1, we get:

$$\hat{A}_d = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & -\Delta T \\ 0 & -0.981\Delta T & 1 & 0 \\ 0 & 21.582\Delta T & 0 & 1 \end{bmatrix}, \quad \hat{B}_d = \begin{bmatrix} 0 \\ 0 \\ 0.1\Delta T \\ 0.2\Delta T \end{bmatrix} \tag{11}$$

Here, A_d, B_d contribute to the discrete time model $x[k+1] = Ax[k] + Bu[k]$.

Task 4

In this section, we set up and test the simulation of the cart-pole system. The simulation code was downloaded and modified as `cart_pole_falling.py`.

The simulation was run for both initial conditions, and the results were recorded in GIF format. These GIFs illustrate the behavior of the cart-pole system when starting from different initial angles.

The simulations show the behavior of the cart-pole system under different initial conditions. As expected, the pole falls over time due to the lack of control input to maintain its upright position.

Additionally, the initial condition 'zero' is also tested. Whether it is initially at the "pi" position or at the "0" position, the system will remain stationary.

In this setup and testing phase, we successfully simulated the cart-pole system under different initial conditions. The recorded GIFs provide a clear visual representation of the system's dynamics in these cases. (Animated GIF images may require Adobe to open the document to view them.)

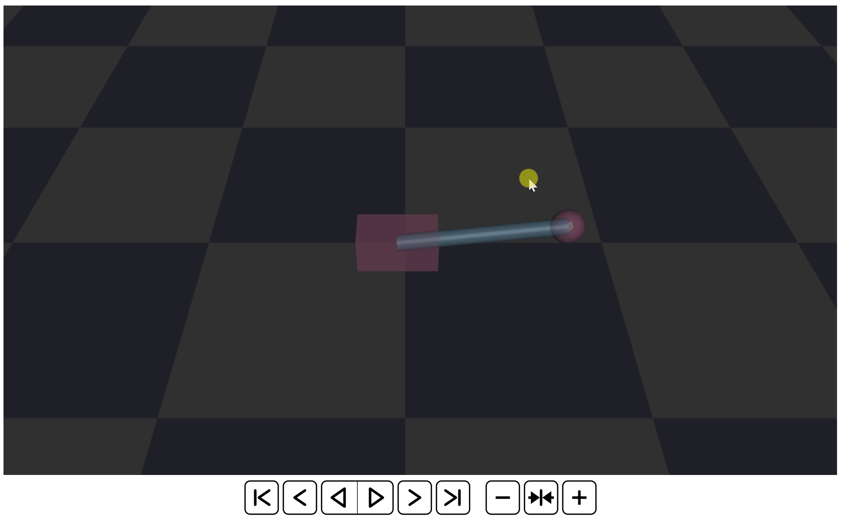


Figure 2: Simulation of the cart-pole system with initial condition `data.qpos[pole_id] = 2.33`.

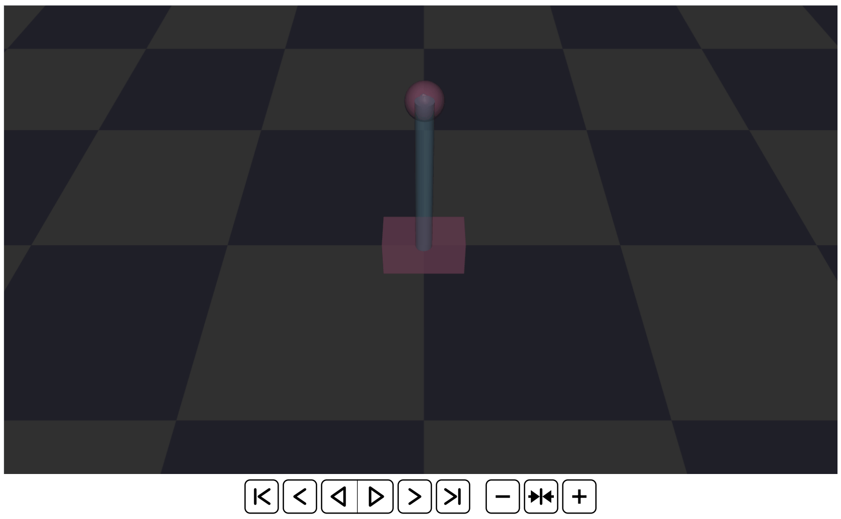


Figure 3: Simulation of the cart-pole system with initial condition `np.pi - 0.002`.

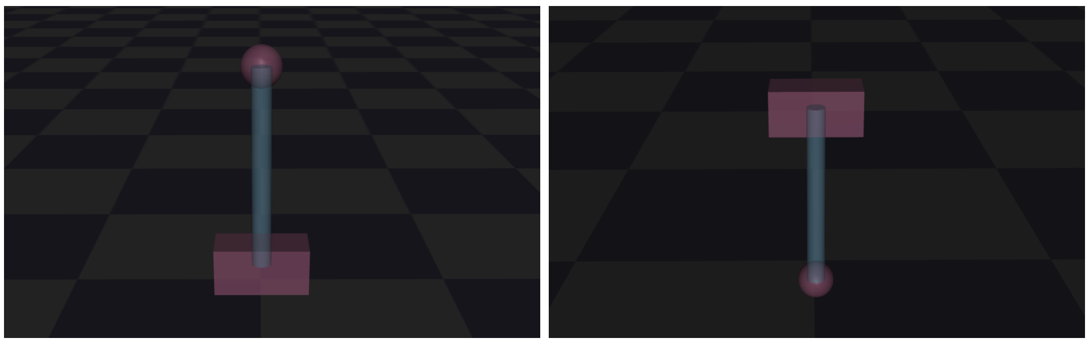


Figure 4: Simulation of the system with initial conditions at π position (left) and 0 position (right).

Task 5

LQR Algorithm Analysis

The Linear Quadratic Regulator (LQR) is an optimal control strategy that minimizes a quadratic cost function. For a discrete-time system, the objective is to find a control input u that minimizes the cost function:

$$J_\infty = \sum_{k=0}^{\infty} \left(x_k^T Q x_k + u_k^T R u_k \right) \tag{12}$$

where Q is a positive semi-definite matrix and R is a positive definite matrix. These matrices define the trade-off between state deviation and control effort.

Given the discrete-time state-space representation:

$$x_{k+1} = A_d x_k + B_d u_k \tag{13}$$

the optimal control law is:

$$u_k = -K x_k \tag{14}$$

where K is the feedback gain matrix computed using the discrete-time Algebraic Riccati Equation (DARE). The DARE is solved iteratively to find the matrix P that satisfies:

$$P = A_d^T P A_d - A_d^T P B_d (R + B_d^T P B_d)^{-1} B_d^T P A_d + Q \tag{15}$$

Once P is found, the gain matrix K is computed as:

$$K = (R + B_d^T P B_d)^{-1} B_d^T P A_d \tag{16}$$

Code Implementation

The implementation of the LQR controller in the simulation is divided into several parts. The main simulation scripts are `cart_pole_simulation.py` and `cart_pole_control.py` (differences are shown in Appendix 1), which call functions from `pendulum.py` and `lqr_discrete.py`.

The main simulation scripts: `cart_pole_simulation.py` set up the simulation environment, initialize the state and apply the LQR controller. The implementation of the LQR controller in the simulation is divided into several parts. Below is the excerpt of the key parts of the script where the initial conditions are set, and the state of the pendulum is updated.

```python
import numpy as np
import tools.pendulum

# Initial perturbation angle for the pendulum
delta_theta = 0.15

# Initial state vector:
x0 = np.array([[0],[delta_theta],[0],[0]])

# Time step from the model
T = model.opt.timestep
print('T:', T)
```

```python
# Initialize the pendulum with the initial state and time step
pendulum = tools.pendulum.Pendulum(x=x0, Q=Q, R=R, T=T)

# In the running of simulation

# Update the state of the pendulum in each simulation step
# Here, the state vector xi and control input ui are updated
xi, ui = pendulum.step_in()

# Directly set the position of the cart and pole
# Set the angle of the pole (pendulum)
data.qpos[pole_id] = np.pi - xi[1][0]
# Set the position of the cart
data.qpos[cart_id] = xi[0][0]
```

In the above code, we first define the initial perturbation angle `delta_theta` for the pendulum. Next, we initialize the pendulum with this state and the time step from the model. During the simulation, in each step, the pendulum's state is updated by calling `pendulum.step_in()`. The updated state is then directly set to the position of the cart and pole in the simulation environment.

The discrete time step class is established in `pendulum.py`. This script contains the class to initialize the pendulum, compute the discrete-time system matrices, and update the state using LQR control. Below are the core parts of this class:

```python
import numpy as np
from . import lqr_discrete as lqr

class Pendulum:
    def __init__(self,
                 T=0.001,
                 Q=np.eye(4),
                 R=np.eye(1),
                 x=np.array([[0.0],[0.0],[0.0],[0.0]]),
                 u=np.array([[0.0]]),
                 mc=10,
                 mp=1,
                 l=0.5,
                 g=9.81,
                 epochs=5000):
        # Initialize the parameters and matrices for the pendulum system
        self.T = T
        self.A = np.array([
            [1, 0, T, 0],
            [0, 1, 0, -T],
            [0, -mp*g/mc * T, 1, 0],
            [0, -(mc+mp)*g/l/mc * T, 0, 1]
        ])
        self.B = np.array([[0.0], [0.0], [1/mc * T], [1/l/mc * T]])
        self.Q = Q.copy()
        self.R = R.copy()
        self.K = None
        self.updataK(epochs=epochs)
        self.x = x.copy()
        self.u = u.copy()
        self.zs = []
        self.thetas = []

    def updataK(self, epochs=500):
        # Update the feedback matrix K using the LQR algorithm
        self.K = lqr.getDiscreteKN(self.A, self.B, self.Q, self.R, epochs, 4, 1)

    def step_in(self):
        # Update the state of the pendulum in one step
        self.u = -self.K @ self.x
        self.x = self.A @ self.x + self.B @ self.u
        self.zs.append(self.x[0][0])
        self.thetas.append(float(np.pi)-self.x[1][0])
        return self.x.copy(), self.u.copy()
```

The `Pendulum` class defines the system matrices A and B based on the given parameters and initializes the state and control vectors. The method `updataK()` updates the LQR feedback gain K using the specified Q and R matrices. The `step_in()` method updates the state of the pendulum at each time step based on the LQR control law and appends the current state to the history lists `zs` and `thetas` for tracking the position and angle over time.

Additional, `lqr_discrete.py` contains the function to compute the LQR gain matrix K:

```
import numpy as np
import numpy.linalg as la

def getDiscreteKN(A, B, Q, R, N, nx, nu):
    P = np.zeros((nx,nx))
    K = np.zeros((nu,nx))
    for j in range(N):
        K = la.inv(R + B.T @ P @ B) @ B.T @ P @ A
        P = Q + A.T @ P @ A - A.T @ P @ B @ K
    K = la.inv(R + B.T @ P @ B) @ B.T @ P @ A
    return K
```

The code shown above is just a sample. For details, please see the original files.

Q and R Matrix Selection

In this implementation, the matrices Q and R are chosen to balance the trade-off between state deviations and control effort. The specific choice of Q and R is:

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \quad R = \begin{bmatrix} 1 \end{bmatrix} \tag{17}$$

This choice of Q penalizes deviations in all state variables equally, including position, angle, velocity, and angular velocity. The value 3 in the diagonal elements reflects a moderate penalty on these deviations. The matrix R is chosen to be 1, which imposes a penalty on the control effort. This balance ensures that the controller effectively reduces the state deviations while not exerting excessive control force, leading to a stable and efficient system.

Conclusion

The screenshot during the simulation is as follows:

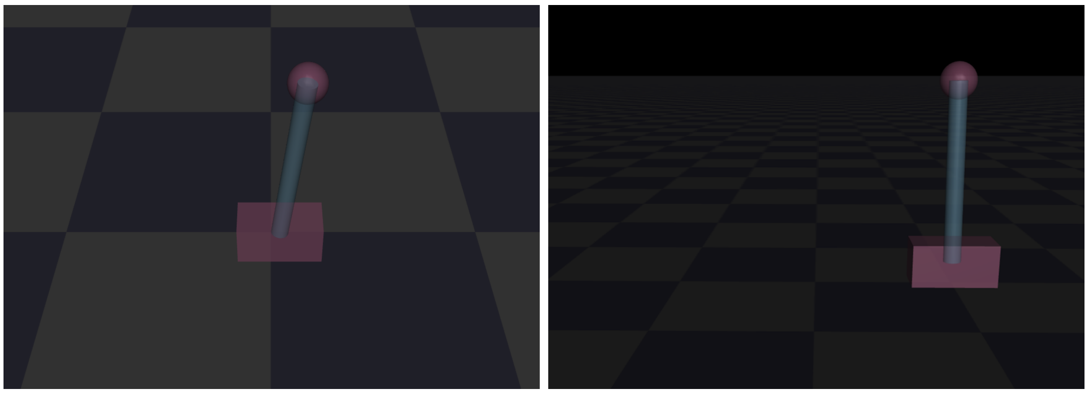


Figure 5: Simulation of the system with controller.

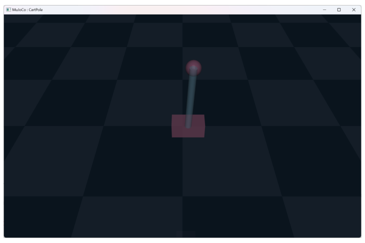
In this section, we analyzed the LQR algorithm and implemented it in a simulation of the cart-pole system. The results show that the LQR controller can effectively stabilize the system around the upright position. The choice of Q and R matrices impacts the performance and should be carefully selected based on the specific requirements of the system.

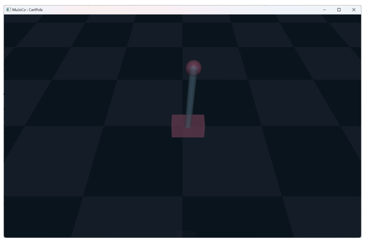
Task 6

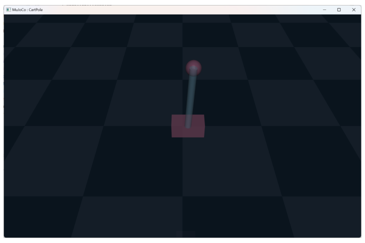
In `cart_port_control.py`, we set a series of Q and R rather than 2 sets of Q and R. After simulating for 0.1 and 0.2 seconds, we print the center position and the angular of the system (i.e, z and $x_2 = \pi - \theta$), to judge the result. The result is shown in Table.2, and the Q, R sets are in Appendix 2. The initial state of θ is $\pi - 0.1$.

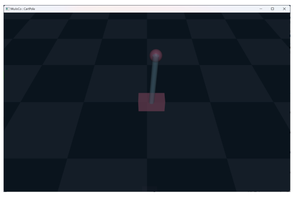
Q	R	z for 0.1 second	x_2 for 0.1 seconds	z for 0.2 second	x_2 for 0.2 seconds
Q_1	R_1	1.48×10^{-2}	8.01×10^{-2}	4.19×10^{-2}	5.22×10^{-2}
Q_2	R_2	1.44×10^{-2}	8.08×10^{-2}	4.12×10^{-2}	5.38×10^{-2}
Q_3	R_3	1.41×10^{-2}	8.14×10^{-2}	4.06×10^{-2}	5.51×10^{-2}
Q_4	R_4	1.49×10^{-2}	7.98×10^{-2}	4.22×10^{-2}	5.15×10^{-2}
Q_5	R_5	1.45×10^{-2}	8.07×10^{-2}	4.13×10^{-2}	5.34×10^{-2}
Q_6	R_6	1.44×10^{-2}	8.08×10^{-2}	4.12×10^{-2}	5.38×10^{-2}
Q_7	R_7	1.50×10^{-2}	7.97×10^{-2}	4.24×10^{-2}	5.13×10^{-2}

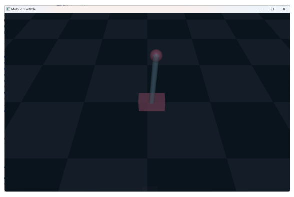
Table 2: Results for 7 Q,R sets.

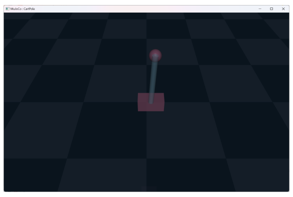

(a) Set 1


(b) Set 2


(c) Set 3


(d) Set 4


(e) Set 5


(f) Set 6

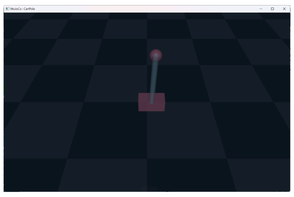

(g) Set 7

Figure 6: Simulation results for 0.2 seconds.

Different configuration on Q and R reaches different results. According to the results, we discover that, generally, if the Q matrix is larger or the R matrix is smaller, then the movement along z is larger, and the θ returns to π at a faster speed. Besides, comparing Experiment 6,7 to Experiment 1, where we pay more attention to the angular in Experiment 6 and focus more on the position in Experiment 7, we conclude that in these experiments, if we concentrate more on the punishing position, the inverted pendulum will recover faster.

Appendix

Appendix 1 - Project Structure

We provide the project file structure in the file `project2.zip`, whose structure is shown below:

```
|- mujoco_files
|    - cart_pole.xml
|- tools
|   |- lqr_discrete.py      # .py file for calculating feedback matrix K.
|   |- pendulum.py          # .py file for inversed pendulum.
|    - animation.py         # .py file for self simulation.
|- cart_pole_falling.py     # mujoco simulation for free-falling.
|- cart_pole_simulation.py  # mujoco simulation.
|- cart_pole_control.py     # mujoco control simulation.
 - self_simulation.ipynb    # self simulation trails.
```

There are some differences between `cart_pole_simulation.py` and `cart_pole_control.py`. In `cart_pole_simulation.py`, we simulate the result locally and directly set the position and angular for the model in the mujoco model, while in `cart_pole_control.py`, we get the state feedback from the mujoco model and set input `u` to control the pendulum.

Besides, you can also access our model on Optima_Estimation.

Appendix 2 - Q, R sets for task 6

$$Q_1 = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \quad R_1 = \begin{bmatrix} 1 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 1 \end{bmatrix}$$

$$Q_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_3 = \begin{bmatrix} 3 \end{bmatrix}$$

$$Q_4 = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_4 = \begin{bmatrix} 1 \end{bmatrix}$$

$$Q_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}, \quad R_5 = \begin{bmatrix} 1 \end{bmatrix}$$

$$Q_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}, \quad R_6 = \begin{bmatrix} 1 \end{bmatrix}$$

$$Q_7 = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_7 = \begin{bmatrix} 1 \end{bmatrix}$$