ME424 Project 2 Report

11912404 江铁豪

1. Full state space model:

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

where $x = \begin{bmatrix} z \\ \theta \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

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

$$= \begin{bmatrix} x_3 \\ x_4 \\ \dfrac{1}{m_c + m_p \sin^2 x_2}\left[u + m_p \sin x_2\left(L{x_4}^2 + g\cos x_2\right)\right] \\ \dfrac{1}{L\left(m_c + m_p \sin^2 x_2\right)}\left[-u\cos x_2 - m_p L{x_4}^2\cos x_2\sin x_2 - \left(m_c + m_p\right)g\sin x_2\right] \end{bmatrix}$$

$$= \begin{bmatrix} x_3 \\ x_4 \\ \dfrac{1}{10 + \sin^2 x_2}\left[u + \sin x_2\left(0.5{x_4}^2 + 9.81\cos x_2\right)\right] \\ \dfrac{1}{5 + 0.5\sin^2 x_2}\left[-u\cos x_2 - 0.5{x_4}^2\cos x_2\sin x_2 - 107.91\sin x_2\right] \end{bmatrix} = f(x,u)$$

2. Angle dynamic model:

$$x_\theta = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix}$$

$$\dot{x}_\theta = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x}_{\theta 1} \\ \dot{x}_{\theta 2} \end{bmatrix} = \begin{bmatrix} x_{\theta 2} \\ \frac{1}{L\left(m_c + m_p \sin^2 x_{\theta 1}\right)}\left[-u\cos x_{\theta 1} - m_p L x_{\theta 2}{}^2 \cos x_{\theta 1} \sin x_{\theta 1} - \left(m_c + m_p\right) g \sin x_{\theta 1}\right] \end{bmatrix}$$

$$= \begin{bmatrix} x_{\theta 2} \\ \frac{1}{5 + 0.5\sin^2 x_{\theta 1}}\left[-u\cos x_{\theta 1} - 0.5 x_{\theta 2}{}^2 \cos x_{\theta 1} \sin x_{\theta 1} - 107.91 \sin x_{\theta 1}\right] \end{bmatrix} = f_\theta(x_\theta, u)$$

3. Linearized model for angle dynamics:

$$< \hat{x}_\theta, \hat{u} >= ([\pi, 0]^T, 0),$$

where $\cos\hat{\theta} = 1, \sin\hat{\theta} = 0$

$$\dot{x}_\theta = \begin{bmatrix} \dot{x}_{\theta 1} \\ \dot{x}_{\theta 2} \end{bmatrix} = \begin{bmatrix} f_{\theta 1} \\ f_{\theta 2} \end{bmatrix} = \begin{bmatrix} x_{\theta 2} \\ \frac{1}{L\left(m_c + m_p \sin^2 x_{\theta 1}\right)}\left[-u\cos x_{\theta 1} - m_p L x_{\theta 2}{}^2 \cos x_{\theta 1} \sin x_{\theta 1} - \left(m_c + m_p\right) g \sin x_{\theta 1}\right] \end{bmatrix}$$

$$\hat{A} = \left.\frac{\partial f_\theta}{\partial x_\theta}\right|_{< \hat{x}_\theta, \hat{u} >= ([\pi,0]^T, 0)} = \begin{bmatrix} \frac{\partial f_{\theta 1}}{\partial x_{\theta 1}} & \frac{\partial f_{\theta 1}}{\partial x_{\theta 2}} \\ \frac{\partial f_{\theta 2}}{\partial x_{\theta 1}} & \frac{\partial f_{\theta 2}}{\partial x_{\theta 2}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{10791}{500} & 0 \end{bmatrix}$$

$$\hat{B} = \left.\frac{\partial f_\theta}{\partial u}\right|_{< \hat{x}_\theta, \hat{u} >= ([\pi,0]^T, 0)} = \begin{bmatrix} \frac{\partial f_{\theta 1}}{\partial u} \\ \frac{\partial f_{\theta 2}}{\partial u} \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{1}{5} \end{bmatrix}$$

$$\dot{x}_\theta = \begin{bmatrix} 0 & 1 \\ -\frac{10791}{500} & 0 \end{bmatrix} \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{5} \end{bmatrix} u + \begin{bmatrix} 0 \\ -\frac{10791\pi}{500} \end{bmatrix}$$

4. Discrete Time Angel Dynamics Model:

$$x_\theta[k+1] = x_\theta(t+T) = x_\theta(t) + \dot{x}_\theta(t)T$$

$$= \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix} + 0.005 \times \left[\hat{A}x_\theta + \hat{B}u + f(\hat{x}, \hat{u}) \right]$$

$$= I \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix} + 0.005 \times \begin{bmatrix} 0 & 1 \\ \frac{10791}{500} & 0 \end{bmatrix} \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix} + 0.005 \times \begin{bmatrix} 0 \\ \frac{1}{5} \end{bmatrix} u + \begin{bmatrix} 0 \\ -0.10791\pi \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.005 \\ 0.10791 & 1 \end{bmatrix} \begin{bmatrix} x_{\theta 1} \\ x_{\theta 2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0.001 \end{bmatrix} u + \begin{bmatrix} 0 \\ -0.339 \end{bmatrix}$$

$$y(k) = \theta_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_\theta(k)$$

$$A = \begin{bmatrix} 1 & 0.005 \\ 0.10791 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0.001 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

5. Drake Simulation Setup and Testing

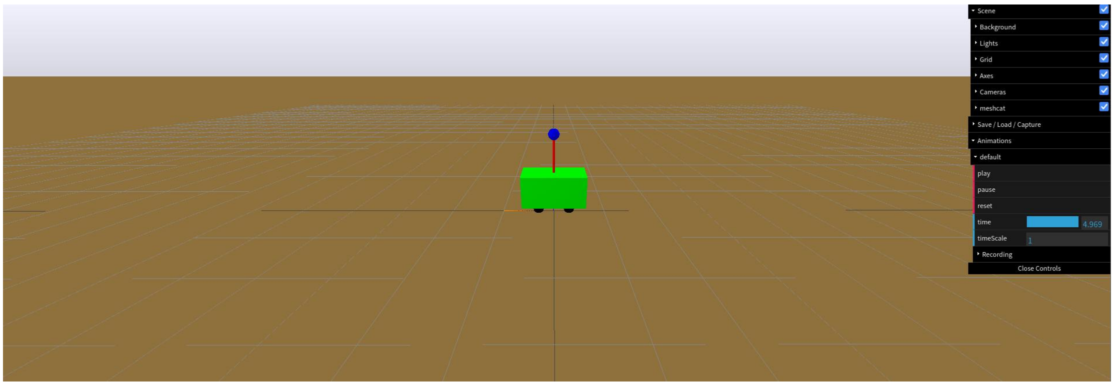
Initial state vectors are shown as below:

```
# start simulation
UprightState = np.array([0, np.pi, 0, 0])   # the state of the cart-pole is organized as [z, theta, zdot, theta_dot]
UprightState = np.array([0, np.pi/2, 0, 0])   # the state of the cart-pole is organized as [z, theta, zdot, theta_dot]
UprightState = np.array([0, np.pi/3, 0, 0])   # the state of the cart-pole is organized as [z, theta, zdot, theta_dot]
```

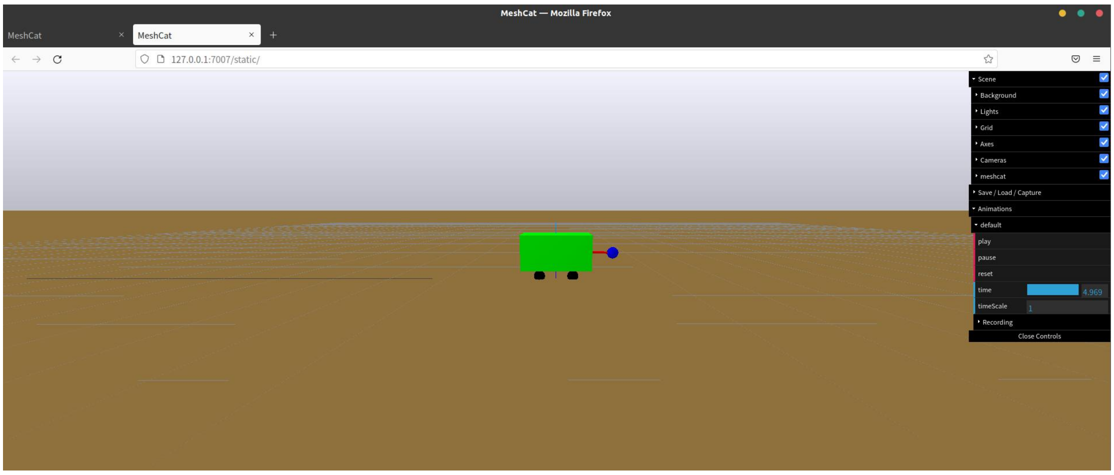
All snapshots shown as below are the simulations with no disturbance.

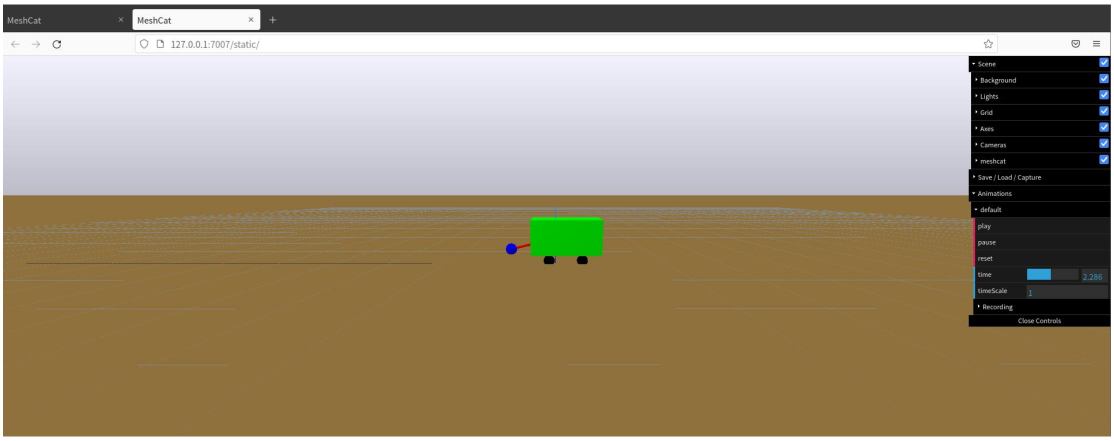
```
context = simulator.get_mutable_context()
# context.SetContinuousState(UprightState + np.array([0.1,0.3,0.3,0.1]))  # 有扰动偏移 disturbance
context.SetContinuousState(UprightState)  # 无扰动偏移 No disturbance
```

$\theta_{init} = \pi$

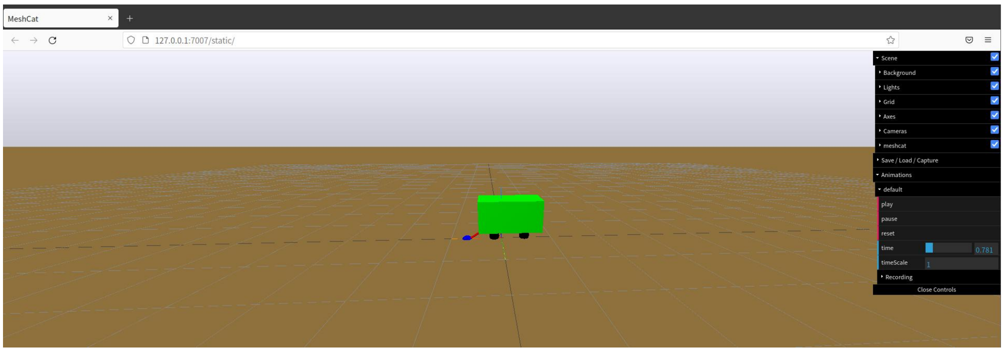


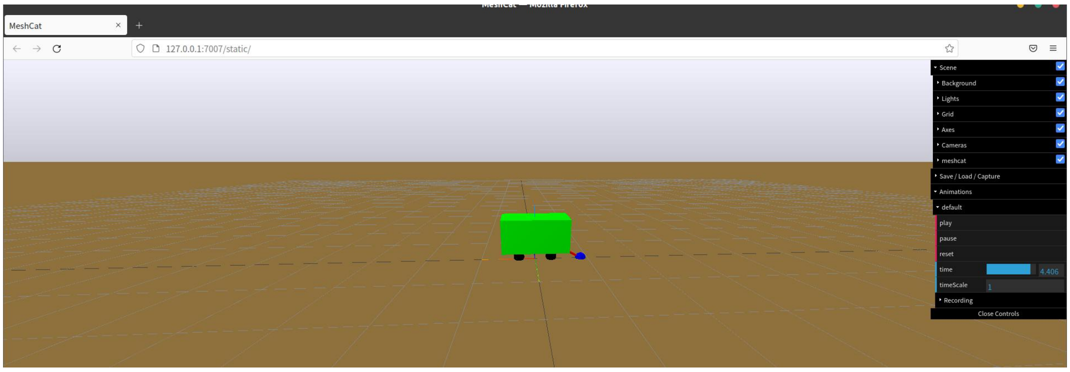
$\theta_{init} = \pi/2$ (GIF file(q5_0.5pi.gif) is attached in the document)





$\theta_{init} = \pi/3$ (GIF file(q5_0.33pi.gif) is attached in the document)





6. Simulation studies for state-feedback control

(a) Eigenvalue assignment

To find K such that A-BK has eigenvalue $z_{1,2} = e^{s_{1,2} * T}$, where $s_{1,2} = -2 \pm j$, we should:

The first step:

converting A, B matrices to a python code form:

The matrix shown as below is $f(x,u)$

```
vector_field
```

✓ 0.3s Python

$$\begin{bmatrix} \theta_d \\ \frac{-L\theta_d^2 m_p \sin(\theta)\cos(\theta) - f_x \cos(\theta) - g(m_c+m_p)\sin(\theta)}{L(m_c+m_p \sin^2(\theta))} \end{bmatrix}$$

The second step:

Calculating Ahat and Bhat matrix based on Part 3.

```
pfpx = vector_field.jacobian([theta,theta_d])
pfpu = vector_field.jacobian([fx])

# g = 9.81
# mc = 10
# mp = 1
# L =0.5

pfpx_f=sym.lambdify((g,fx,mc,mp,L,theta,theta_d), pfpx)

pfpu_f = sym.lambdify((g,fx,mc,mp,L,theta,theta_d), pfpu)
Ahat= pfpx_f(9.81,0,10,1,0.5,np.pi,0,)
Bhat = pfpu_f(9.81,0,10,1,0.5,np.pi,0,)
✓ 0.3s                                                    Python

Ahat
✓ 0.3s                                                    Python
array([[ 0.    ,  1.    ],
       [21.582, -0.    ]])

Bhat
✓ 0.8s                                                    Python
array([[0. ],
       [0.2]])
```

The third step:

Computing the matrix A and the matrix B in Part 4, the form of discrete time angel dynamics model based on the continuous-time linearized model in Part 3.

```
import scipy.signal as sig
I = np.eye(2)
A = I + 0.005*Ahat
B = 0.005*Bhat
print(A)
print(B)
✓ 0.7s                                                    Python
[[1.      0.005  ]
 [0.10791 1.     ]]
[[0.   ]
 [0.001]]
```

$$A = \begin{bmatrix} 1 & 0.005 \\ 0.10791 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0.001 \end{bmatrix}$$

The fourth step:

Applying the function *sig.place_poles(matrix A, matrix B, z_desired).gain_matrix* to find gain K, which leads to the desired eigenvalues of A-BK.

```
s_desired = np.array([-2+1j, -2-1j])
z_desired = np.exp(s_desired*0.005)
cart_pole_gain = sig.place_poles(A, B, z_desired).gain_matrix
print(cart_pole_gain)
✓ 0.1s                                                    Python
[[132.66140054  19.9250837 ]]
```

(b) Closed-loop simulation

After calculating K, we should test the control $u = -kx_\theta$ is feasible or not, or the efficiency of adjustment to stable.

```
class myController(LeafSystem):
    def __init__(self, K):
        LeafSystem.__init__(self)
        self.DeclareVectorInputPort("u", BasicVector(4))
        self.DeclareVectorOutputPort("y", BasicVector(1), self.CalcOutputY)
        self.K = K
    def CalcOutputY(self, context, output):
        statex = self.get_input_port(0).Eval(context)
        y = -np.dot(self.K, (statex-np.array([0, np.pi, 0, 0])))
      #  print(statex, y, statex-np.array([0, np.pi, 0, 0]))
        output.SetFromVector([y])
```

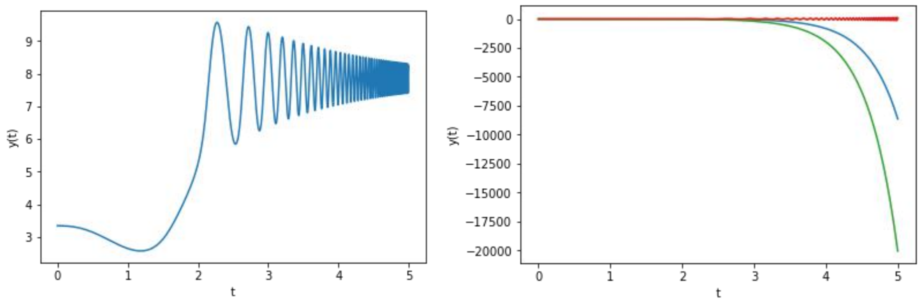
Since K_1 and K_3 is meant for z, and factor of z could make a difference to θ, so we prepared 2 sets of K with different K_1 and K_3.

(i) $eigenvalues = \ -2 \pm j,$

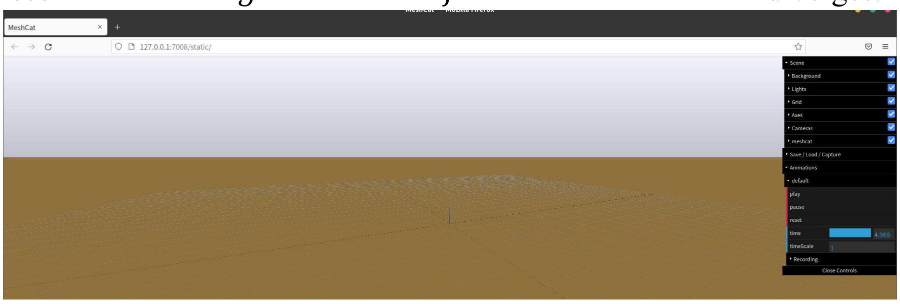
$K = [-15.29051988 \ 132.66140054 \ -18.85830785 \ 19.9250837]$

```
builder = DiagramBuilder()
# K = np.array([-15.29051988, 220.55525994, -18.85830785,  44.42915392])
K = np.array([-15.29051988, 132.66140054,-18.85830785,  19.9250837]) # Question7 -2-j -2+j
# K = np.array([-15.29051988, 444.48486861,-18.85830785,  21.48420104]) # Question7 -2-8j -2+8j
# K = np.array([0, 132.66140054, -6,  19.9250837]) # Question7 -2-j -2+j
#K = np.array([0, 444.48486861, -6,  21.48420104]) # Question7 -2-8j -2+8j
```

The diagram is shown as below:



We cans see that θ $converges\ to\ a\ value\ far\ more\ than\ \pi\ and\ z$ diverges.



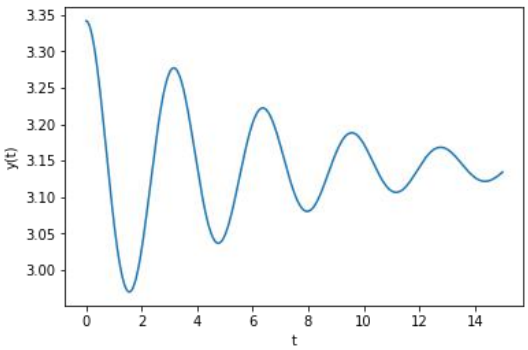
The cart is lost in the simulation, which indicates the failure(seen in q6b_1.gif) We can see that under such conditions, the cart pole is unstable. To eliminate the effect of other factor we used another set of K.

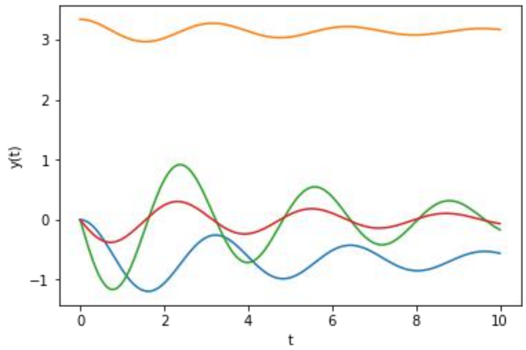
(ii) $eigenvalues = \ -2 \pm j,$

$K = [0 \ 132.66140054 \ -6 \ 19.9250837]$

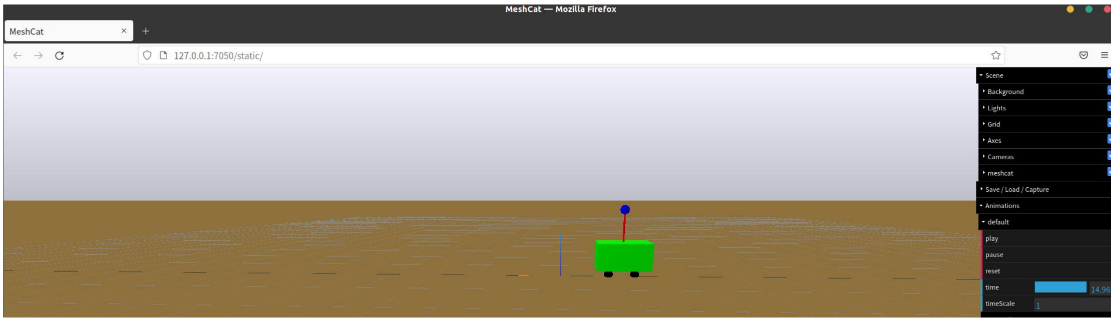
```
builder = DiagramBuilder()
# K = np.array([-15.29051988, 220.55525994, -18.85830785,  44.42915392])
# K = np.array([-15.29051988, 132.66140054,-18.85830785,  19.9250837]) # Question7 -2-j -2+j
# K = np.array([-15.29051988, 444.48486861,-18.85830785,  21.48420104]) # Question7 -2-8j -2+8j
K = np.array([0, 132.66140054, -6,  19.9250837]) # Question7 -2-j -2+j
#K = np.array([0, 444.48486861, -6,  21.48420104]) # Question7 -2-8j -2+8j
#K = np.array([-0. ,        124.96  ,       -6.13149847,  17.06574924])
```

The diagram is shown as below:





We can see that θ *and* z *both converge, and* θ *converges to* π



[q6b_2.gif is attached]

7. Repeat with different eigenvalues

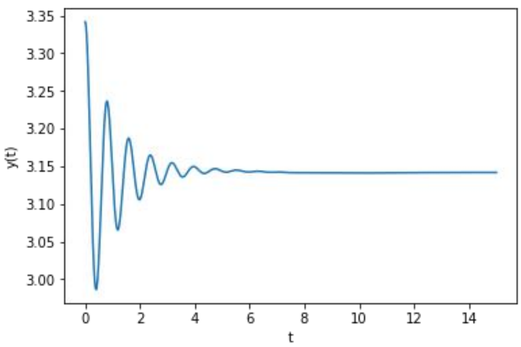
First calculate K:

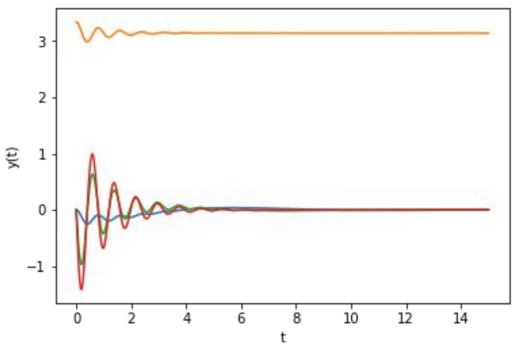
```
s_desired = np.array([-2+8j, -2-8j])
# s_desired = np.array([-2+1j, -2-1j])
z_desired = np.exp(s_desired*0.005)
cart_pole_gain = sig.place_poles(A, B, z_desired).gain_matrix
print(cart_pole_gain)
```

```
[[444.48486861  21.48420104]]
```

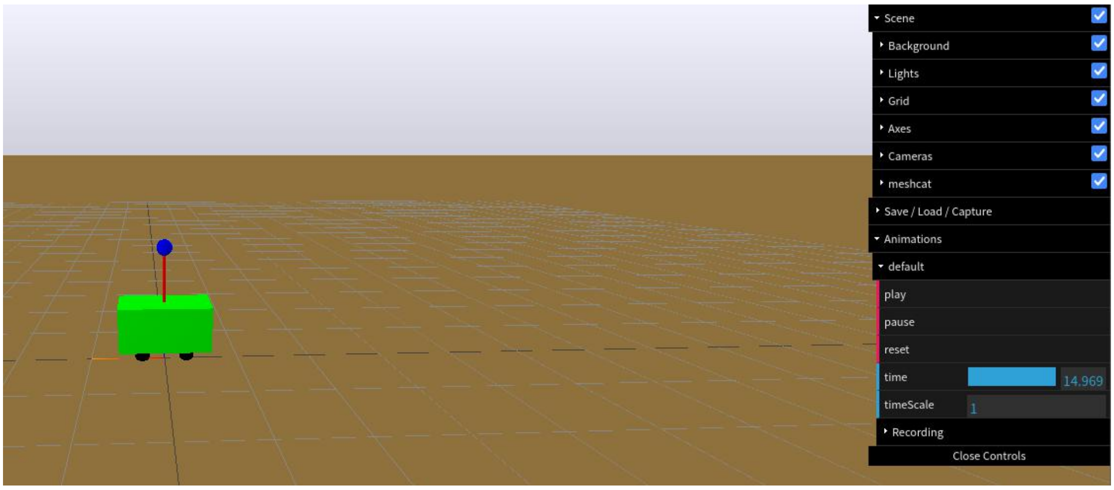
(i) $\boldsymbol{eigenvalues = \ -2 \pm 8j,}$

$K = [-15.29051988\ \ 444.48486861\ \ -18.85830785\ \ 21.48420104]$





Both converges



q7b_1.gif is attached

Analysis:

Compared with

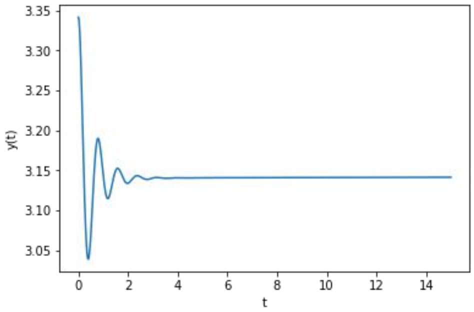
$K = [-15.29051988\ 132.66140054\ -18.85830785\ 19.9250837]$ of eigenvalues $= -2 \pm j$

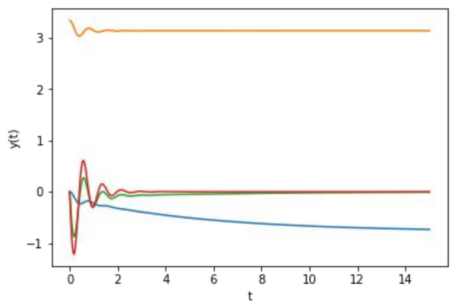
$K = [-15.29051988\ 444.48486861\ -18.85830785\ 21.48420104]$ of $eigenvalues = -2 \pm 8j$ takes shorter time to converge to π and be in a stable state, which is around 5 second, whose respond is faster.

But for the previous one, it doesn't show a typically control plotting, and it also converges to a value much larger than π. And its 's converge time is much more than 5 seconds.

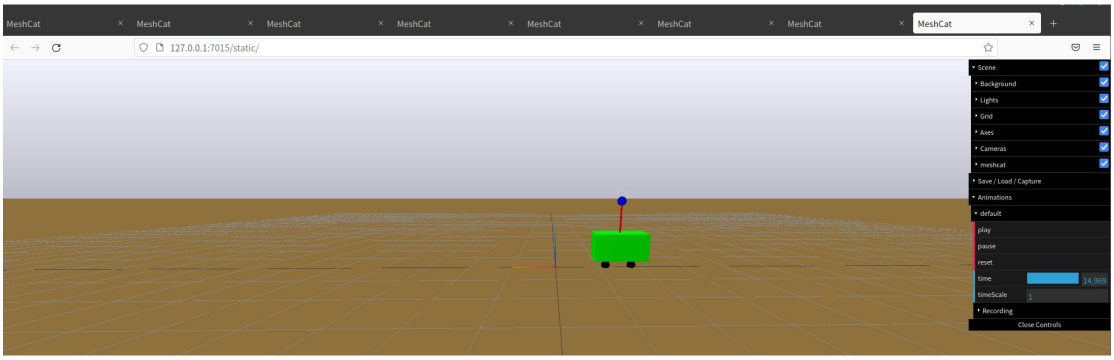
(ii) $\boldsymbol{eigenvalues = -2 \pm 8j}$,

$K = [0\ 444.48486861\ -6\ 21.48420104]$





Both converges



q7b_2.gif is attached

Analysis:

Compared with

$K = [0 \ 132.66140054 \ -6 \ 19.9250837]$ of eigenvalues $= -2 \pm j$

$K = [0 \ 444.48486861 \ -6 \ 21.48420104]$ of $eigenvalues = -2 \pm 8j$ takes shorter time to converge to π and be in a stable state, where the previous one is more than 15 seconds, where the latter is around 2 second, whose respond is faster.

8. Output feedback control design:

The code of observer is shown as below:

The first one is to compute L of the specific eigenvalues $-9 \pm 2j$

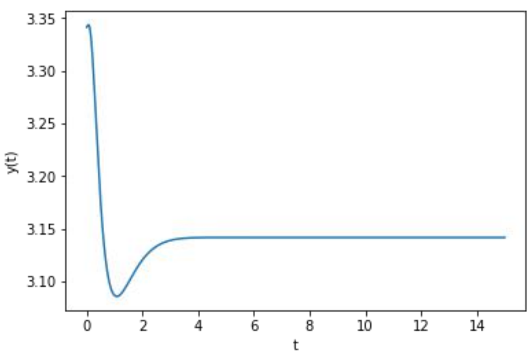
```
# Calculate L
import scipy.signal as sig
A = np.array([[1, 0.005],[0.10791, 1]])
B = np.array([[0],[0.001]])
C = np.array([[1, 0]])
s_desired = np.array([-9+2j, -9-2j])
z_desired = np.exp(s_desired*0.005)
L = sig.place_poles(A.T, C.T, z_desired).gain_matrix
L = L.T
print(L)
✓ 0.3s
[[0.08810064]
 [0.51427411]]
```

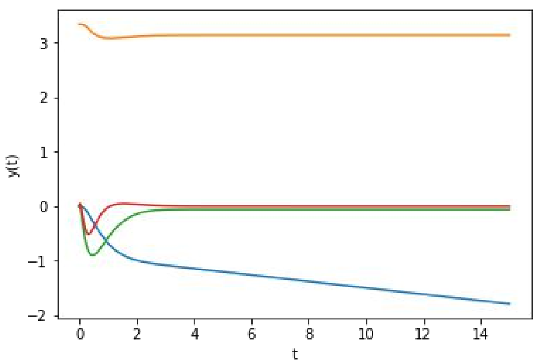
The second one is to set up the model of DT Luenberger observer

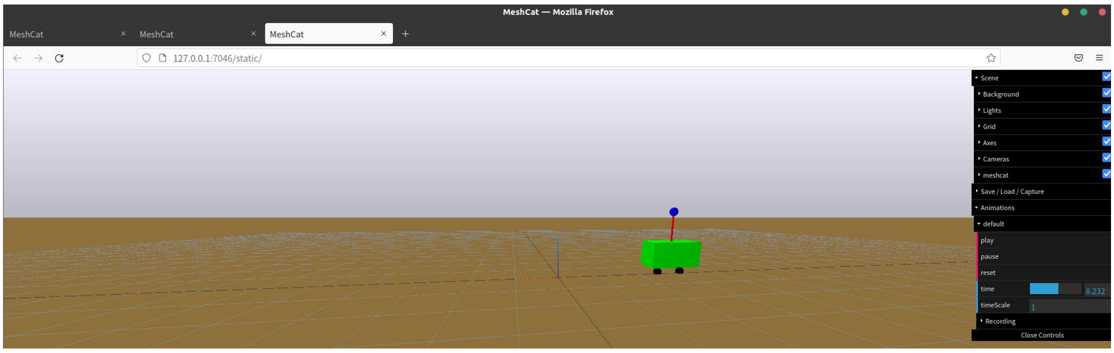
```python
# Define the system.
class DTObserver(LeafSystem):
    def __init__(self,A,B,C,L,T):
        LeafSystem.__init__(self)
        self.DeclareDiscreteState(2)
        # Define the input
        self.DeclareVectorInputPort("uk", BasicVector(1))
        self.DeclareVectorInputPort("yk", BasicVector(4))
        # Define the output
        self.DeclareVectorOutputPort("x_estimated", BasicVector(2), self.CalcOutputY, set([self.all_state_ticket()]))
        self.DeclarePeriodicDiscreteUpdate(T)  # One second timestep.
        self.A = A
        self.B = B
        self.C = C
        self.L = L
    def DoCalcDiscreteVariableUpdates(self, context, events, discrete_state):
        xk = context.get_discrete_state_vector().CopyToVector()
        # xk1 = np.array([[xk[1]],[xk[3]]])
        # print(xk1)
        uk = self.get_input_port(0).Eval(context)
        yk = self.get_input_port(1).Eval(context)
        yk1 = yk[1]-np.pi
        A = self.A
        B = self.B
        C = self.C
        L = self.L
        # print(A.dot(xk1))
        # xk1 = np.mat(xk1.reshape((2,1)))
        xnext = A.dot(xk) + B.dot(uk) + L.dot(yk1 - C.dot(xk))
        # xnext1 = np.array([[xk[0]],[xnext[0]],[xk[2]],[xnext[1]]])
        discrete_state.get_mutable_vector().SetFromVector(xnext)

    def CalcOutputY(self, context, output):
        x = context.get_discrete_state_vector().CopyToVector()
        y = x
        output.SetFromVector(y)
```

Disturbance = 0.2

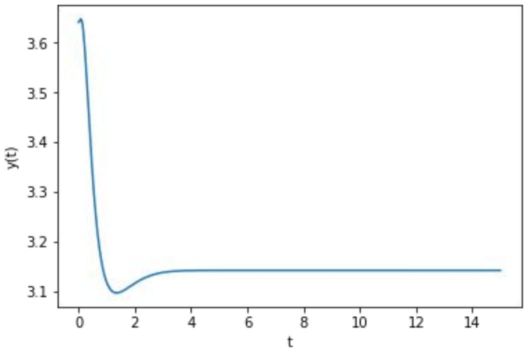


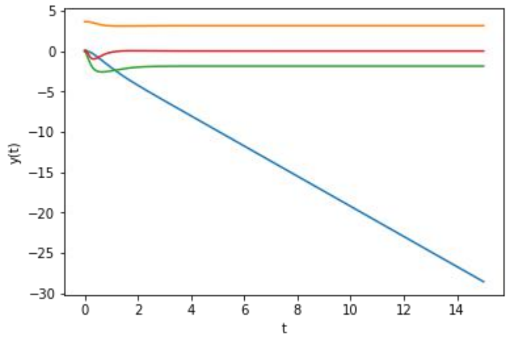


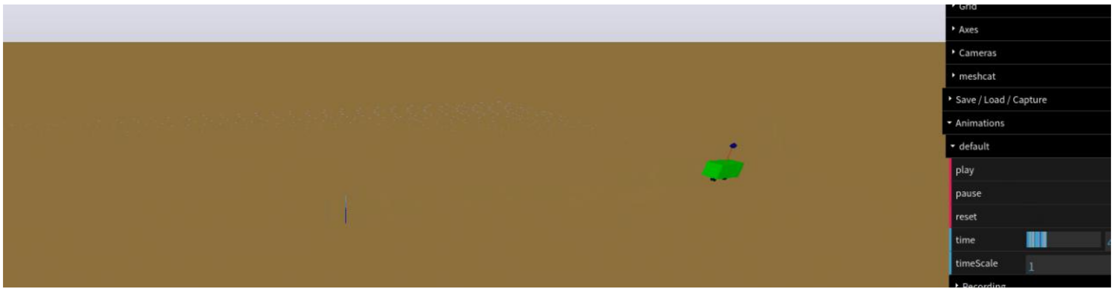


[q81.gif](q81.gif)

Disturbance = 0.5

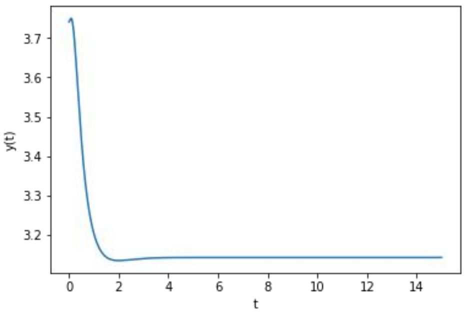


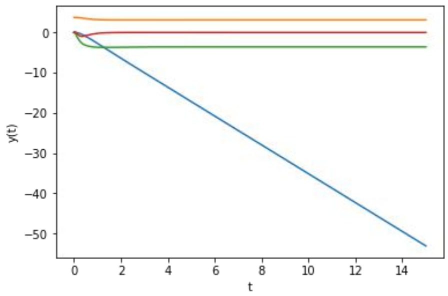


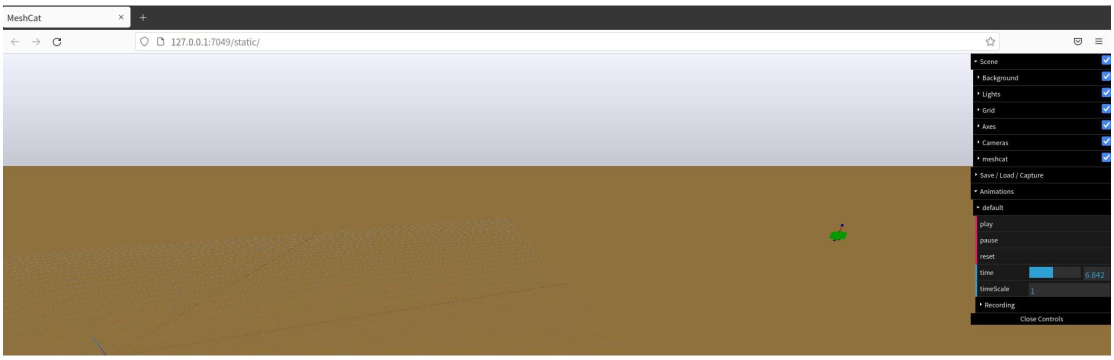


[q82.gif]

Disturbance = 0.6

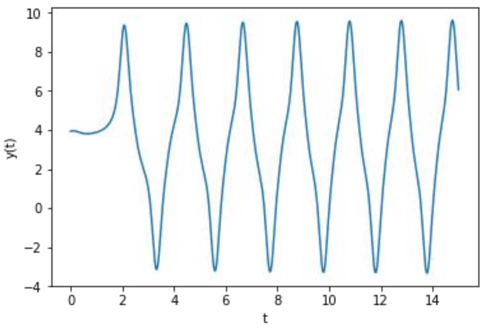


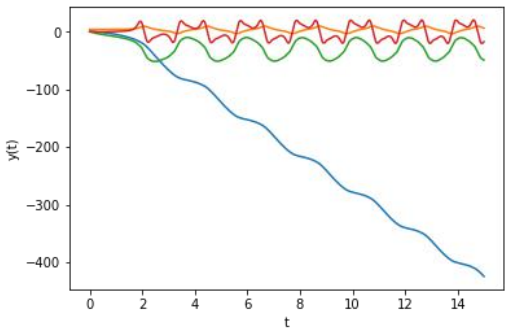


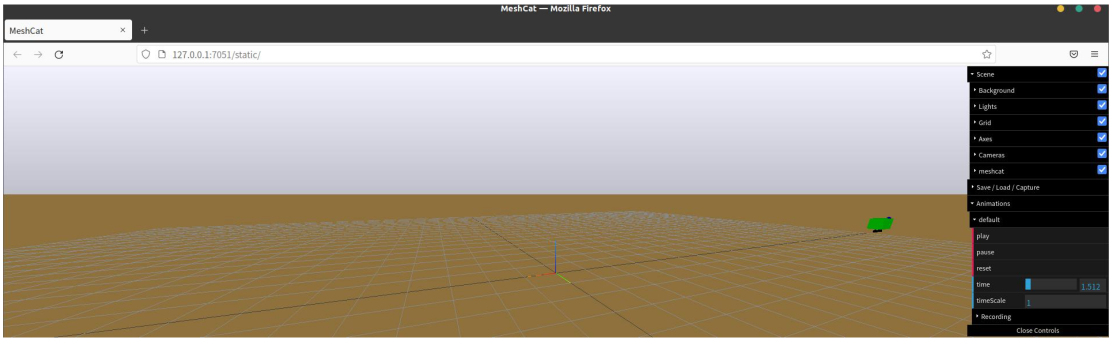


[q83.gif]

Disturbance = 0.8







q84.gif

We find out that when the initial conditions is in a specific range, the ange of the pole will remain in a stable and desired angle. If the disturbance exceed the range, like the diagrams above when disturbance = 0.8, it may oscillate in a period, and also for z, it will diverge.