Advanced Control for Robotics - Homework 4

涂志鑫 12131094

2022.04

Problem 1

Solution:

(a) The pose of frame{A} is ${}^{O}T_{A} = (R, P)$, while it is moving with $\nu = \begin{bmatrix} w \\ v \end{bmatrix}$, so the derivate of the ${}^{o}X_{A}$ is:

$$\frac{d}{dt}[{}^{o}X_{A}] = \begin{bmatrix} \dot{R} & 0 \\ ([p]w)' & \dot{R} \end{bmatrix} = \begin{bmatrix} w \times R & 0 \\ ([\dot{p}]w + [p]\dot{w}) & w \times R \end{bmatrix} = \begin{bmatrix} [w]R & 0 \\ ([\dot{p}]w + [p]\dot{w}) & [w]R \end{bmatrix}$$

Let's denote that $\dot{p} = V + W \times p$,$[a \times b] = [a][b] - [b][a]$, the third term can be simplified as

$$\begin{aligned} ([\dot{p}]w + [p]\dot{w}) &= [v]R + [w \times p]R + [p][w]R \\ &= [v]R + [w][p]R - [p][w]R + [p][w]R \\ &= [v]R + [w][p]R \end{aligned}$$

Therefore,

$$\begin{aligned} \frac{d}{dt}[{}^{o}X_{A}] &= \begin{bmatrix} [w]R & 0 \\ [v]R + [w][p]R & [w]R \end{bmatrix} \\ &= \begin{bmatrix} [w] & 0 \\ [v] & [w] \end{bmatrix} \begin{bmatrix} R & 0 \\ [p]R & R \end{bmatrix} \\ &= \begin{bmatrix} [w] & 0 \\ [v] & [w] \end{bmatrix} {}^{o}X_{A} \end{aligned}$$

Problem 2

Solution:

Assume ${}^0C = [C_x(t), 0, 0]^T$. According to the previous problem, we can easily know the ${}^c\nu = [0, v/r, 0, 0, 0, 0]^T$. The twist of the body in frame{0} is ${}^0\nu = [0, \frac{v}{r}, 0, 0, 0, \frac{C_x(t)v}{r}]^T$. (a) Because the cylinder rolls with a constant velocity v, the derivate of the $C_x(t)$ is $C_x'(t) = v$ Find the ${}^oA,$ since frame{o} is fixed, the expression of ${}^o\mathbb{A}$ is

$$
{}^o\mathbb{A} = \frac{d}{dt}[{}^o\nu] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ C_x'(t)v/r \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{v^2}{r} \end{bmatrix}
$$

(b)

$$
{}^cX_o = \begin{bmatrix} I_{3\times3} & 0 \\ [{}^oP_c] & I_{3\times3} \end{bmatrix}
$$

$$
{}^c\mathbb{A} = {}^c\dot{\nu}_c + {}^c\nu \times {}^c\nu_c = {}^cX_o{}^o\mathbb{A} = \begin{bmatrix} I_{3\times3} & 0 \\ [{}^oP_c] & I_{3\times3} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{v^2}{r} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{v^2}{r} \end{bmatrix}
$$

Problem 3

Solution:

$\dot{M} - 2c = \sum_i(\dot{J}_i^T I_i J_i + J_i^T \dot{I}_i J_i + J_i^T I_i \dot{J}_i) - 2\sum_i(J_i^T I_i \dot{J}_i - J_i^T I_i \nu_i \times J_i - J_i^T \nu_i \times^* I_i J_i)$ Denote that I_i is constant, so $\dot{I}_i = 0$, and I_i is a symmetric matrix, $I_i^T = I_i$.

$$
\dot{M} - 2c = \sum_i(\dot{J}_i^T I_i J_i - J_i^T I_i \dot{J}_i - 2J_i^T I_i \nu_i \times J_i - 2J_i^T \nu_i \times^* I_i J_i)
$$

$$
\begin{aligned}
(\dot{M}-2c)^T &= \sum_i (J_i^T I_i \dot{J}_i - \dot{J}_i^T I_i J_i - 2J_i^T [\nu_i \times]^T I_i J_i - 2J_i^T I_i [\nu_i \times^*]^T J_i) \\
&= \sum_i (J_i^T I_i \dot{J}_i - \dot{J}_i^T I_i J_i + 2J_i^T I_i [\nu \times^*] J_i + 2J_i^T [\nu_i \times]^T I_i J_i)
\end{aligned}
$$

Denote that $[\nu_i \times]^T = -[\nu \times^*]$. Therefore,

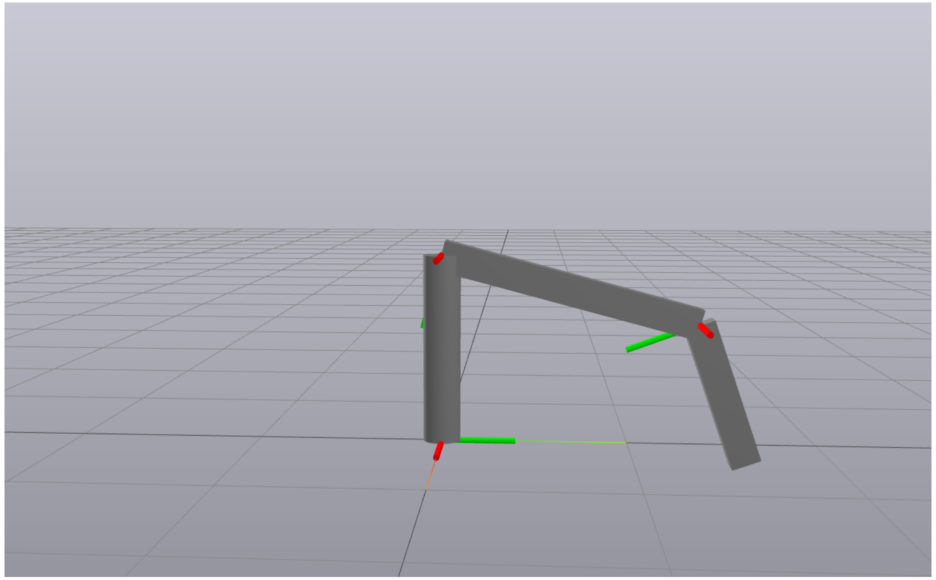
$$
(\dot{M}-2c) + (\dot{M}-2c)^T = 0
$$

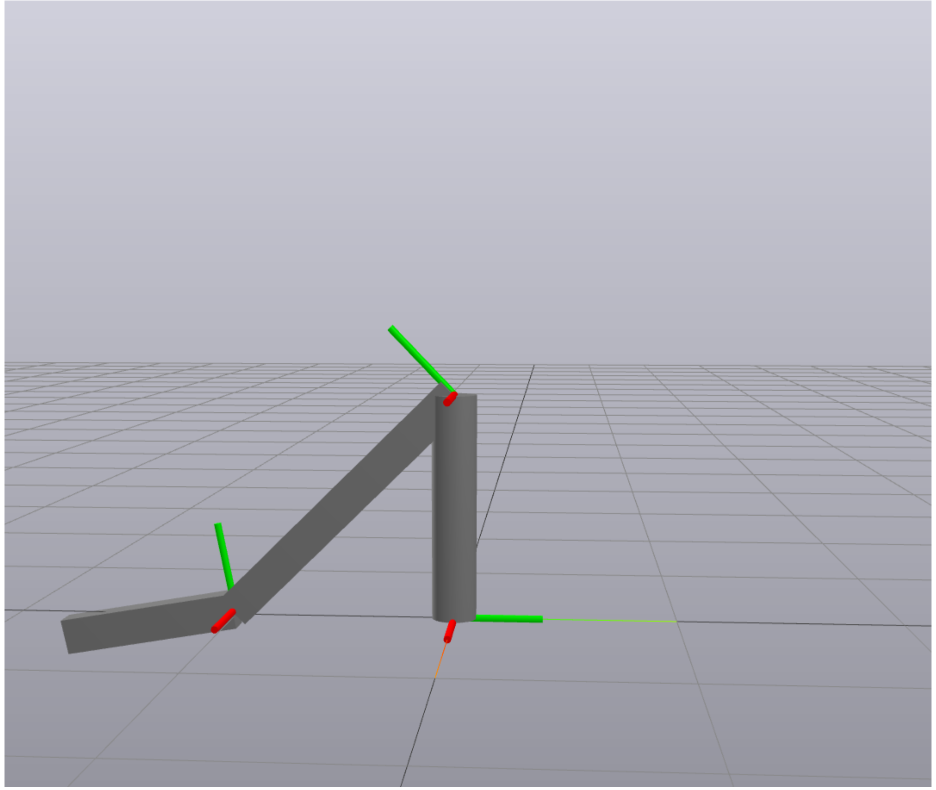
That is prove that $\dot{M}-2c$ is skew symmetric.

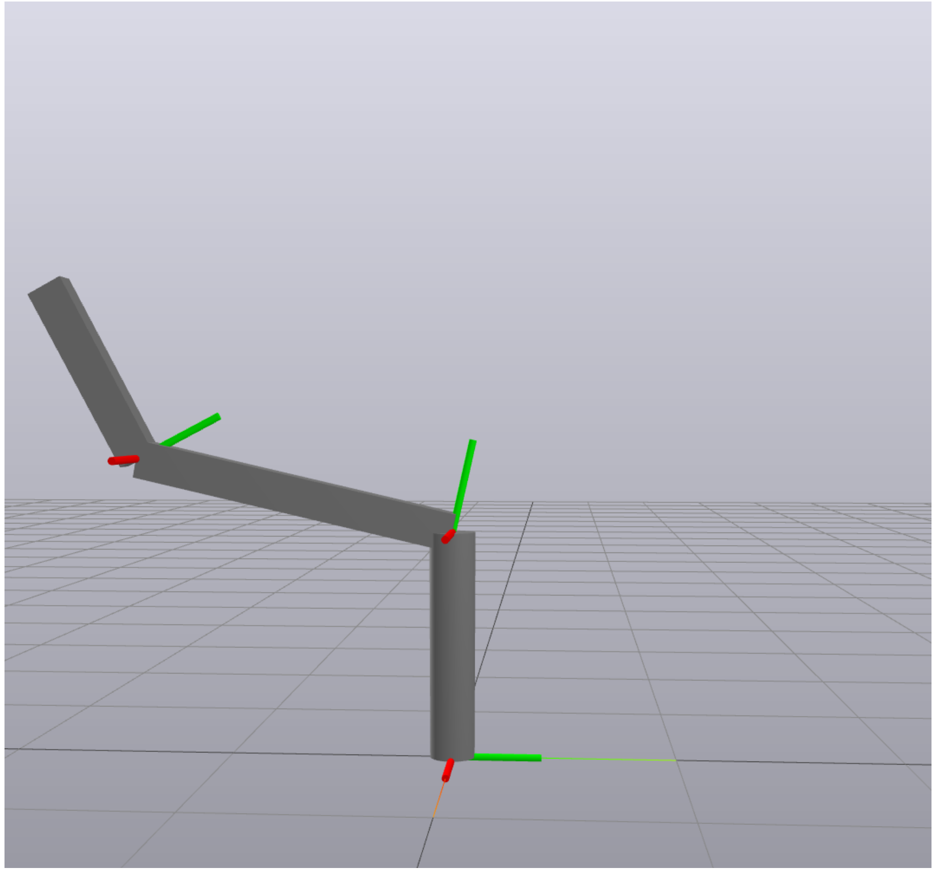
Problem 4

Solution

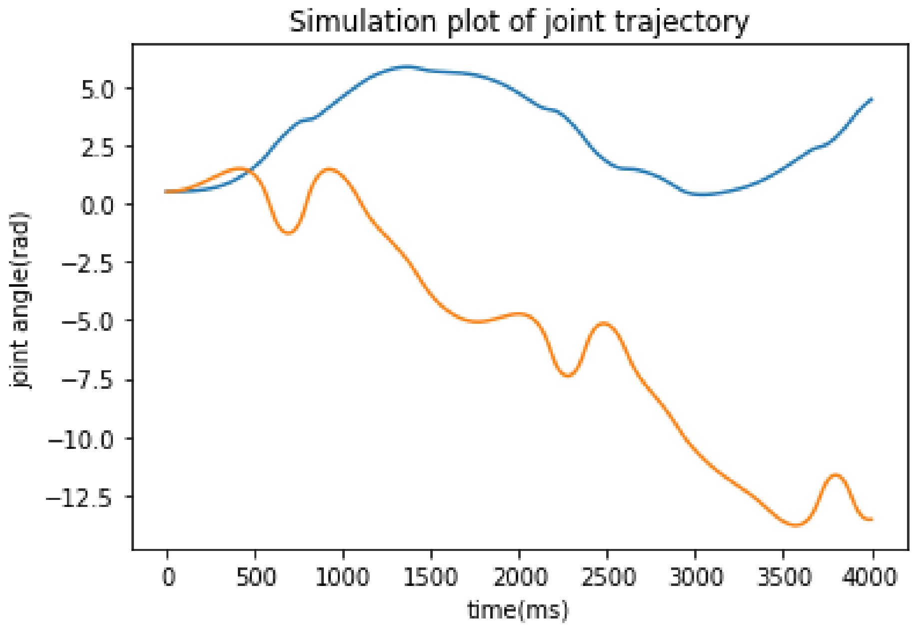
(a) Choose the initial configuration is $\theta_1 = 0.5rad, \theta_2 = 0.5rad$, and start simulating for 4 sec.







The plot of joint trajectories for the simulation is as shown below.



In [ ]:

```
# problem 4nstant reference to the joint
import numpy as np
import matplotlib.pyplot as plt
import pydot
from IPython.display import display, SVG, clear_output

from pydrake.math import RigidTransform, RollPitchYaw
from pydrake.multibody.plant import AddMultibodyPlantSceneGraph
from pydrake.all import (Parser, StartMeshcat, DiagramBuilder,
                         MeshcatVisualizerCpp, JacobianWrtVariable,
                         MakeRenderEngineVtk, RenderEngineVtkParams,
                         Simulator, CoulombFriction, HalfSpace,
                         RotationMatrix, RotationalInertia)
from pydrake.geometry import (
    Box,
    Cylinder
)

from pydrake.multibody.tree import (
    PrismaticJoint,
    UnitInertia,
    SpatialInertia,
```

```
    RevoluteJoint,
    FixedOffsetFrame,
    WeldJoint
)

from manipulation.meshcat_cpp_utils import MeshcatJointSliders
from manipulation.scenarios import AddMultibodyTriad
import modern_robotics as mr
from pydrake.all import LogVectorOutput
import time
```

In [ ]:

```
meshcat = StartMeshcat()
```

```
Meshcat is now available at http://localhost:7000
```

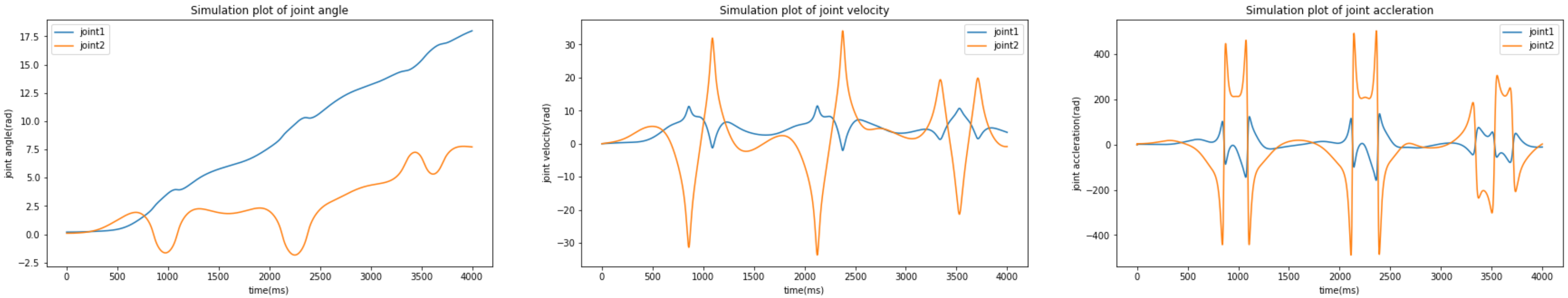
In [ ]:

```
# Build robot from code
builder = DiagramBuilder()
step_size = 1e-4
plant, scene_graph = AddMultibodyPlantSceneGraph(builder, step_size)

# Some parameters
L0 = 0.5
L1 = 0.7
L2 = 0.4

h = 0.06
w = 0.08
r = 0.05

m = 1

RGBA_Color = [0.5, 0.5, 0.5, 1]
mu = 0.4

my_model_instance = plant.AddModelInstance("my_robot")

inertia_link_0 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L0/2], RotationalInertia(m*(3*r**2+L0**2)/12, m*(3*r**2+L0**2)/12, m*r**2/2))
inertia_link_1 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L1/2], RotationalInertia(m*(w**2+L1**2)/12, m*(h**2+L1**2)/12, m*(h**2+w**2)/12))
inertia_link_2 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L2/2], RotationalInertia(m*(w**2+L2**2)/12, m*(h**2+L2**2)/12, m*(h**2+w**2)/12))
```

```python
link_0 = plant.AddRigidBody(
    "link_0", my_model_instance, inertia_link_0)
link_1 = plant.AddRigidBody(
    "link_1", my_model_instance, inertia_link_1)
link_2 = plant.AddRigidBody(
    "link_2", my_model_instance, inertia_link_2)


plant.RegisterVisualGeometry(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0/2]),
    Cylinder(r, L0),
    "link_0",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1/2]),
    Box(h, w, L1),
    "link_1",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_2,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L2/2]),
    Box(h, w, L2),
    "link_2",
    RGBA_Color)

frame_on_link_0 = plant.AddFrame(FixedOffsetFrame(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0])))

frame_on_link_1 = plant.AddFrame(FixedOffsetFrame(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1])))

plant.AddJoint(RevoluteJoint(
    name="joint_0_to_1", frame_on_parent=frame_on_link_0,
    frame_on_child=link_1.body_frame(), axis=[1, 0, 0]))

plant.AddJoint(RevoluteJoint(
    name="joint_1_to_2", frame_on_parent=frame_on_link_1,
    frame_on_child=link_2.body_frame(), axis=[1, 0, 0]))
```

```python
plant.WeldFrames(
    frame_on_parent_P=plant.world_frame(),
    frame_on_child_C=link_0.body_frame(),
    X_PC=RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, 0]))

# Draw RGB frames for visualization
for body_name in ["link_0", "link_1", "link_2"]:
    AddMultibodyTriad(plant.GetFrameByName(body_name), scene_graph, 0.20, 0.008)

# Finalize and visualize
plant.Finalize()

renderer_name = "renderer"
scene_graph.AddRenderer(
    renderer_name, MakeRenderEngineVtk(RenderEngineVtkParams()))

meshcat.Delete()
meshcat_vis = MeshcatVisualizerCpp.AddToBuilder(
    builder, scene_graph, meshcat)

# logger_output = LogVectorOutput(plant.get_body_poses_output_port(), builder)
diagram = builder.Build()

diagram_context = diagram.CreateDefaultContext()
plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
plant.SetPositions(plant_context, plant.GetModelInstanceByName("my_robot"),
                   [0.5, 0.5])  # theta1, theta2 the choosen initial config


# a = plant.get_body_poses_output_port().Eval(plant_context)
# print(a)

diagram.Publish(diagram_context)

# simulator = Simulator(diagram, diagram_context)
# simulator.set_publish_every_time_step(True)
# simulator.set_target_realtime_rate(1)
# simulator.Initialize()
# diagram.Publish(diagram_context)
# simulator.AdvanceTo(4)  #siulation for 4 sec


# #plot the joint traj
```

```python
# please write your own simulator below

simulator = Simulator(diagram, diagram_context)
simulator.set_publish_every_time_step(True)
simulator.set_target_realtime_rate(1)
simulator.Initialize()
diagram.Publish(diagram_context)

simulation_time = 4
step_time = 0.001
q_list= np.zeros((1,2))
q_list = plant.GetPositions(plant_context)
# print(time.time())
for i in np.arange(0,simulation_time,step_time):
    q_list = np.vstack((q_list,plant.GetPositions(plant_context)))
    simulator.AdvanceTo(i)  #siulation for 4 sec
# print(time.time())

#plot the joint traj
plt.plot(q_list)
plt.title('Simulation plot of joint trajectory')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
```



(b) The inverse dynamics RNEA function $\tau_i = ID(q, \dot{q}, \ddot{q}, F_{ext}, g, N, S_i, M, I_c, L)$ is as shown below.

In [ ]:

```python
#problem 4(b)    (RNEA ALGORITHM)
import numpy as np
import modern_robotics as mr
```

```python
def ID(q,dq,ddq,Fext,g,N,Si,M,Ic,L):   #RNEA ALGORITHM
    Si_0 = np.zeros((6,N))
    Si_0[:,0] = np.matmul(mr.Adjoint(M[4:8,:]),Si[:,0])
    Si_0[:,1] = np.matmul(mr.Adjoint(M[8:12,:]),Si[:,1])
    tau_i = np.zeros((N))
    nu_i = np.zeros((6,N+1))  #including the base link
    A_i = np.zeros((6,N+1))   #including the base link
    wrench_i = np.zeros((6,N+1))   #including the base link
    A_gto0 = np.array([0,0,0,0,0,-g])

    # Forward pass
    for i in range(1,N+1):
        if i==1:
            T_ito0 = mr.FKinSpace( M[4*i:4*i+4,:], Si_0[:,0:i], q[0:i] )
            T_pito0 = M[0:4,:]
        else:
            T_ito0 = mr.FKinSpace( M[4*i:4*i+4,:], Si_0[:,0:i], q[0:i] )
            T_pito0 = mr.FKinSpace( M[4*(i-1):4*(i-1)+4,:], Si_0[:,0:i-1], q[0:i-1] )
        T_pitoi = np.matmul(mr.TransInv(T_ito0),T_pito0)
        X_ptoi = mr.Adjoint(T_pitoi)
        # print(X_ptoi)
        # print('----------')
        nu_i[:,i] = X_ptoi @ nu_i[:,i-1] + Si[:,i-1]*dq[i-1]         #S_i and dq_i index are i-1
        A_i[:,i] = X_ptoi @ A_i[:,i-1] + Si[:,i-1]*ddq[i-1] + np.matmul(mr.ad(nu_i[:,i]),Si[:,i-1]) * dq[i-1]
        T_itoc = np.array([[1, 0,  0, 0],
                          [ 0, 1,  0, 0],
                          [ 0, 0, 1, -L[i-1]/2],
                          [ 0, 0,  0, 1]])
        X_itoc = mr.Adjoint(T_itoc)
        X_ctoi = mr.Adjoint(mr.TransInv(T_itoc))
        X_ctoi_star = mr.Adjoint(T_itoc).T
        Ii = np.matmul( np.matmul(X_ctoi_star,Ic[(i-1)*6:i*6,:]),X_itoc )
        T_0toi = mr.TransInv(mr.FKinSpace( M[4*i:4*i+4,:], Si_0[:,0:i], q[0:i] ))
        X_0toi = mr.Adjoint(T_0toi)
        # print(np.matmul(Ii,A_i[:,i]))
        # print(np.matmul(-mr.ad(nu_i[:,i]).T,Ii)@ nu_i[:,i])
        # print(-np.matmul(np.matmul(Ii,X_0toi),A_gto0))
        # print(-Fext[:,i-1])
        # print(Ii)
        # print('--------------------')
        # print(np.matmul(Ii,A_i[:,i]) + np.matmul(-mr.ad(nu_i[:,i]).T,Ii)@ nu_i[:,i] \
        #                      - np.matmul(np.matmul(Ii,X_0toi),A_gto0) - Fext[:,i-1])
        wrench_i[:,i] = np.matmul(Ii,A_i[:,i]) + np.matmul(-mr.ad(nu_i[:,i]).T,Ii)@ nu_i[:,i] \
                             - np.matmul(np.matmul(Ii,X_0toi),A_gto0) - Fext[:,i-1]
```

```
    # print(nu_i)
    # print(A_i)
    # print(wrench_i)

    # Backward pass
    for i in range(N,0,-1):
        tau_i[i-1] = np.matmul(Si[:,i-1].T,wrench_i[:,i])
        if i==1:
            T_ito0 = mr.FKinSpace( M[4*i:4*i+4,:], Si_0[:,0:i], q[0:i] )
            T_pito0 = M[0:4,:]
        else:
            T_ito0 = mr.FKinSpace( M[4*i:4*i+4,:], Si_0[:,0:i], q[0:i] )
            T_pito0 = mr.FKinSpace( M[4*(i-1):4*(i-1)+4,:], Si_0[:,0:i-1], q[0:i-1] )
        T_pitoi = np.matmul(mr.TransInv(T_ito0),T_pito0)
        X_itopi_star = mr.Adjoint(T_pitoi).T
        wrench_i[:,i-1] = wrench_i[:,i-1] + np.matmul(X_itopi_star,wrench_i[:,i])

    return tau_i
```

(c) According to the RNEA, we can formulate the forward dynamics. The function is as shown below.

In [ ]:

```
#problem 4(c)   (Forward dynamics function)
import numpy as np
import modern_robotics as mr

def FD(tau_i,q,dq,Fext,N,Si,M,Ic,L):
    #step 1 solve the c~
    ddq_zero = np.zeros(N)
    g = 9.81
    c_tuta = ID(q,dq,ddq_zero,Fext,g,N,Si,M,Ic,L)
    # print(c_tuta)
    # step2 calculate M_theta
    dq_zero = np.zeros(N)
    Fext_zero = np.zeros((6,N))
    M_theta = np.zeros((N,N))
    for i in range(N):
        ddq_j0 = np.zeros((N,1))
        ddq_j0[i,:] = 1
        M_theta[:,i] = ID(q,dq_zero,ddq_j0,Fext_zero,0,N,Si,M,Ic,L)
    # print(M_theta)
    # step3 solve the forward dynamics
    ddq = np.matmul(np.linalg.inv(M_theta),(tau_i - c_tuta))
    return ddq
```

(d) We choose three initial configuration is $\theta_1 = 0.2rad, \theta_2 = 0.1rad$, $\theta_1 = 0.5rad, \theta_2 = 0.5rad$, $\theta_1 = \frac{3}{4}\pi rad, \theta_2 = \frac{3}{4}\pi rad$, and start simulating for 4 sec. The rates of simulation of drake and own code are both $1e-3s$.

We found that the results between own codes and drake simulation are almost the same.

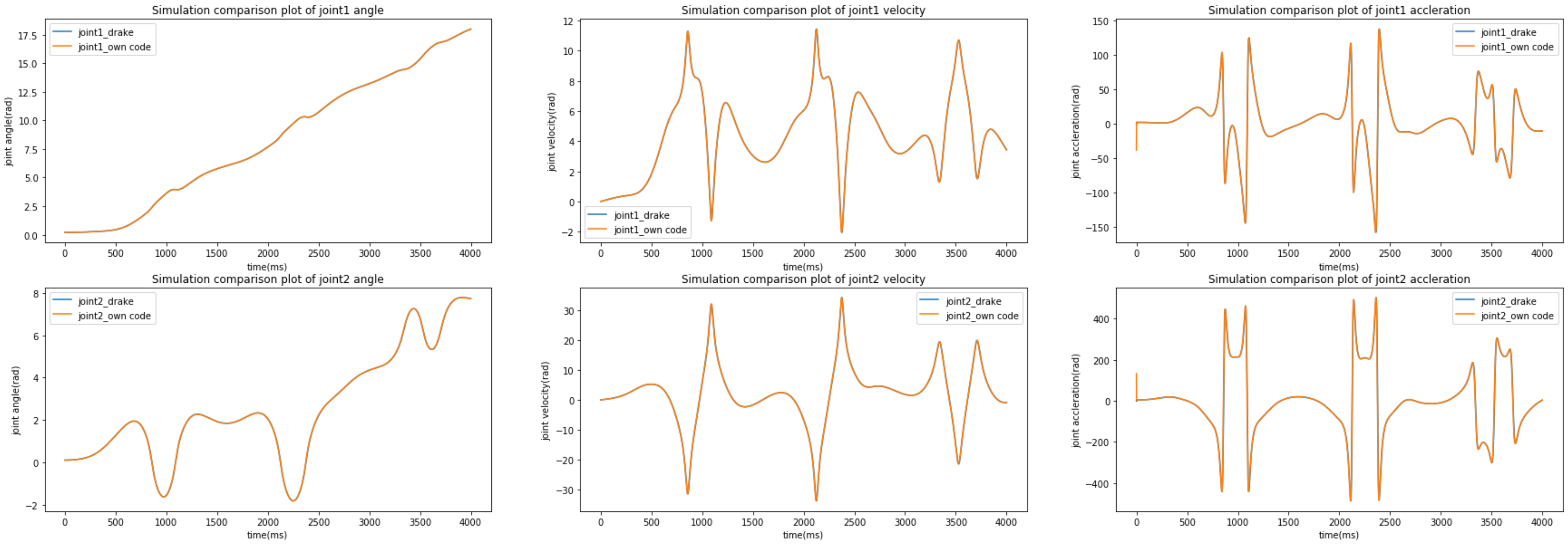
In [ ]:

```
#problem 4(d)  Using ID function to simulate

import numpy as np
import modern_robotics as mr
import matplotlib.pyplot as plt
# Some parameters of the double pendulum
L0 = 0.5
L1 = 0.7
L2 = 0.4
L = np.array([0.7,0.4])
h = 0.06
w = 0.08
r = 0.05

m = 1
N = 2

Si = np.zeros((6,N))        #si in body frame
Si_0 = np.zeros((6,N))      #si in fixed frame
Si = np.array([[1, 0,   0,   0, 0, 0],
                [1, 0,   0,   0, 0, 0]]).T

M = np.zeros(((N+1)*4,4))    #transformation matrix of all frames
M[0:4,:] = np.array([[1, 0,   0, 0],
                      [ 0, 1,   0, 0],
                      [ 0, 0, 1, 0],
                      [ 0, 0,   0, 1]])

M[4:8,:] = np.array([[1, 0,   0, 0],
                      [ 0, 1,   0, 0],
                      [ 0, 0, 1, L0],
                      [ 0, 0,   0, 1]])

M[8:12,:] = np.array([[1, 0,   0, 0],
                      [ 0, 1,   0, 0],
                      [ 0, 0, 1, L1+L0],
```

```
                      [ 0, 0,  0, 1]])
# Si_0[:,0] = np.matmul(mr.Adjoint(M[4:8,:]),Si[:,0])
# Si_0[:,1] = np.matmul(mr.Adjoint(M[8:12,:]),Si[:,0])
#T = mr.FKinSpace(M,Si_0,thetalist)
Ic = np.zeros((N*6,6))
Ic[0:6,:] = np.diag([m*(w**2+L1**2)/12, m*(h**2+L1**2)/12, m*(h**2+w**2)/12,m,m,m])
Ic[6:12,:] = np.diag([m*(w**2+L2**2)/12, m*(h**2+L2**2)/12, m*(h**2+w**2)/12,m,m,m])
g = 9.81

stept = 0.001
# for the initial condition [0.2,0.1], simulating for 4 sec
q = np.array([0.2,0.1])
dq = np.array([0,0])
# ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
tau = np.array([0,0])
Fext = np.zeros((6,N))
simulation_time = 4
q_listsimu = q.reshape(1,N)
dq_listsimu = dq.reshape(1,N)
ddq_listsimu = ddq.reshape(1,N)

for t in np.arange(0,simulation_time,stept):
    ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
    ddq_listsimu = np.vstack((ddq_listsimu,ddq))
    dq = dq + ddq*stept
    dq_listsimu = np.vstack((dq_listsimu,dq))
    q = q + dq*stept
    q_listsimu = np.vstack((q_listsimu,q))

# for t in np.arange(0,simulation_time,stept):
#     ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
#     dq = dq + (ddq+ddq_listsimu[-1,:])/2*stept
#     q = q +(dq+dq_listsimu[-1,:])/2*stept
#     dq_listsimu = np.vstack((dq_listsimu,dq))
#     ddq_listsimu = np.vstack((ddq_listsimu,ddq))
#     q_listsimu = np.vstack((q_listsimu,q))


#plot the joint traj
plt.figure(figsize=(30,10))
ax1 = plt.subplot(2,3,1)
plt.plot(q_list[:,0])
plt.plot(q_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 angle')
```

```
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax2 = plt.subplot(2,3,2)
plt.plot(dq_list[:,0])
plt.plot(dq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax3 = plt.subplot(2,3,3)
plt.plot(ddq_list[:,0])
plt.plot(ddq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax4 = plt.subplot(2,3,4)
plt.plot(q_list[:,1])
plt.plot(q_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 angle')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax5 = plt.subplot(2,3,5)
plt.plot(dq_list[:,1])
plt.plot(dq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax6 = plt.subplot(2,3,6)
plt.plot(ddq_list[:,1])
plt.plot(ddq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])
```

```
plt.show()
```



In [ ]:

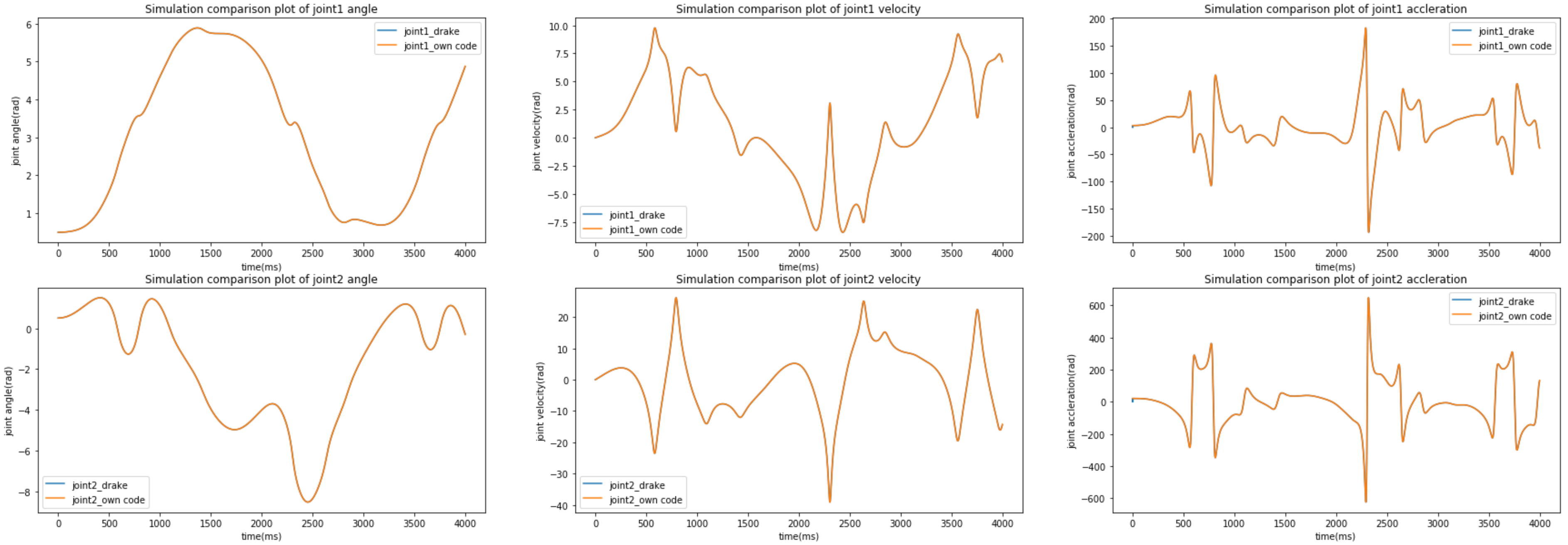
```
# Second initial Conditon

# Build robot from code
builder = DiagramBuilder()
step_size = 1e-3
plant, scene_graph = AddMultibodyPlantSceneGraph(builder, step_size)

# Some parameters
L0 = 0.5
L1 = 0.7
L2 = 0.4

h = 0.06
w = 0.08
r = 0.05

m = 1

RGBA_Color = [0.5, 0.5, 0.5, 1]
```

```
mu = 0.4

my_model_instance = plant.AddModelInstance("my_robot")

inertia_link_0 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L0/2], RotationalInertia(m*(3*r**2+L0**2)/12, m*(3*r**2+L0**2)/12, m*r**2/2))
inertia_link_1 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L1/2], RotationalInertia(m*(w**2+L1**2)/12, m*(h**2+L1**2)/12, m*(h**2+w**2)/12))
inertia_link_2 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L2/2], RotationalInertia(m*(w**2+L2**2)/12, m*(h**2+L2**2)/12, m*(h**2+w**2)/12))

link_0 = plant.AddRigidBody(
    "link_0", my_model_instance, inertia_link_0)
link_1 = plant.AddRigidBody(
    "link_1", my_model_instance, inertia_link_1)
link_2 = plant.AddRigidBody(
    "link_2", my_model_instance, inertia_link_2)


plant.RegisterVisualGeometry(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0/2]),
    Cylinder(r, L0),
    "link_0",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1/2]),
    Box(h, w, L1),
    "link_1",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_2,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L2/2]),
    Box(h, w, L2),
    "link_2",
    RGBA_Color)

frame_on_link_0 = plant.AddFrame(FixedOffsetFrame(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0])))

frame_on_link_1 = plant.AddFrame(FixedOffsetFrame(
    link_1,
```

```python
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1])))

plant.AddJoint(RevoluteJoint(
    name="joint_0_to_1", frame_on_parent=frame_on_link_0,
    frame_on_child=link_1.body_frame(), axis=[1, 0, 0]))

plant.AddJoint(RevoluteJoint(
    name="joint_1_to_2", frame_on_parent=frame_on_link_1,
    frame_on_child=link_2.body_frame(), axis=[1, 0, 0]))

plant.WeldFrames(
    frame_on_parent_P=plant.world_frame(),
    frame_on_child_C=link_0.body_frame(),
    X_PC=RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, 0]))

# Draw RGB frames for visualization
for body_name in ["link_0", "link_1", "link_2"]:
    AddMultibodyTriad(plant.GetFrameByName(body_name), scene_graph, 0.20, 0.008)

# Finalize and visualize
plant.Finalize()

renderer_name = "renderer"
scene_graph.AddRenderer(
    renderer_name, MakeRenderEngineVtk(RenderEngineVtkParams()))

meshcat.Delete()
meshcat_vis = MeshcatVisualizerCpp.AddToBuilder(
    builder, scene_graph, meshcat)
diagram = builder.Build()
diagram_context = diagram.CreateDefaultContext()
plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
plant.SetPositions(plant_context, plant.GetModelInstanceByName("my_robot"),
                   [0.5, 0.5])  # theta1, theta2 the choosen initial config
diagram.Publish(diagram_context)
simulator = Simulator(diagram, diagram_context)
simulator.set_publish_every_time_step(True)
simulator.set_target_realtime_rate(1)
simulator.Initialize()
diagram.Publish(diagram_context)

simulation_time = 4
step_time = 0.001
q_list= np.zeros((1,2))
```

```
dq_list= np.zeros((1,2))
ddq_list = np.zeros((1,2))
q_list = plant.GetPositions(plant_context)
dq_list = plant.GetVelocities(plant_context)
index = 1
# print(time.time())
for i in np.arange(0,simulation_time,step_time):
    q_list = np.vstack((q_list,plant.GetPositions(plant_context)))
    dq_list = np.vstack((dq_list,plant.GetVelocities(plant_context)))
    ddq_list = np.vstack((ddq_list,(dq_list[index] - dq_list[index-1])/step_time))
    simulator.AdvanceTo(i)  #siulation for 4 sec
    index = index + 1

# own code simulation
stept = 0.001
# for the initial condition [0.5,0.5], simulating for 4 sec
q = np.array([0.5,0.5])
dq = np.array([0,0])
ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
tau = np.array([0,0])
Fext = np.zeros((6,N))
simulation_time = 4
q_listsimu = q.reshape(1,N)
dq_listsimu = dq.reshape(1,N)
ddq_listsimu = ddq.reshape(1,N)

for t in np.arange(0,simulation_time,stept):
    ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
    ddq_listsimu = np.vstack((ddq_listsimu,ddq))
    dq = dq + ddq*stept
    dq_listsimu = np.vstack((dq_listsimu,dq))
    q = q + dq*stept
    q_listsimu = np.vstack((q_listsimu,q))


#plot the joint traj
plt.figure(figsize=(30,10))
ax1 = plt.subplot(2,3,1)
plt.plot(q_list[:,0])
plt.plot(q_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 angle')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])
```

```
ax2 = plt.subplot(2,3,2)
plt.plot(dq_list[:,0])
plt.plot(dq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax3 = plt.subplot(2,3,3)
plt.plot(ddq_list[:,0])
plt.plot(ddq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax4 = plt.subplot(2,3,4)
plt.plot(q_list[:,1])
plt.plot(q_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 angle')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax5 = plt.subplot(2,3,5)
plt.plot(dq_list[:,1])
plt.plot(dq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax6 = plt.subplot(2,3,6)
plt.plot(ddq_list[:,1])
plt.plot(ddq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

plt.show()
```



In [ ]:

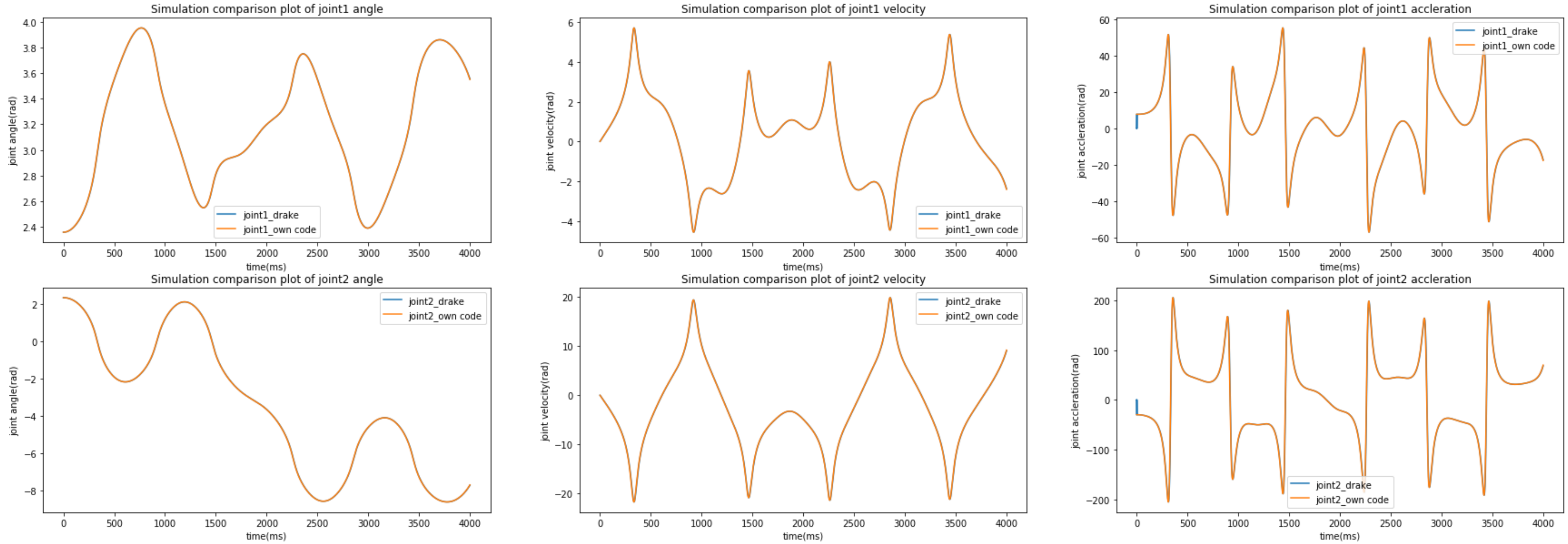
```
# Third initial Conditon

# Build robot from code
builder = DiagramBuilder()
step_size = 1e-3
plant, scene_graph = AddMultibodyPlantSceneGraph(builder, step_size)

# Some parameters
L0 = 0.5
L1 = 0.7
L2 = 0.4

h = 0.06
w = 0.08
r = 0.05

m = 1

RGBA_Color = [0.5, 0.5, 0.5, 1]
mu = 0.4

my_model_instance = plant.AddModelInstance("my_robot")
```

```
inertia_link_0 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L0/2], RotationalInertia(m*(3*r**2+L0**2)/12, m*(3*r**2+L0**2)/12, m*r**2/2))
inertia_link_1 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L1/2], RotationalInertia(m*(w**2+L1**2)/12, m*(h**2+L1**2)/12, m*(h**2+w**2)/12))
inertia_link_2 = SpatialInertia.MakeFromCentralInertia(
    m, [0, 0, L2/2], RotationalInertia(m*(w**2+L2**2)/12, m*(h**2+L2**2)/12, m*(h**2+w**2)/12))

link_0 = plant.AddRigidBody(
    "link_0", my_model_instance, inertia_link_0)
link_1 = plant.AddRigidBody(
    "link_1", my_model_instance, inertia_link_1)
link_2 = plant.AddRigidBody(
    "link_2", my_model_instance, inertia_link_2)


plant.RegisterVisualGeometry(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0/2]),
    Cylinder(r, L0),
    "link_0",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1/2]),
    Box(h, w, L1),
    "link_1",
    RGBA_Color)
plant.RegisterVisualGeometry(
    link_2,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L2/2]),
    Box(h, w, L2),
    "link_2",
    RGBA_Color)

frame_on_link_0 = plant.AddFrame(FixedOffsetFrame(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0])))

frame_on_link_1 = plant.AddFrame(FixedOffsetFrame(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L1])))

plant.AddJoint(RevoluteJoint(
```

```python
    name="joint_0_to_1", frame_on_parent=frame_on_link_0,
    frame_on_child=link_1.body_frame(), axis=[1, 0, 0]))

plant.AddJoint(RevoluteJoint(
    name="joint_1_to_2", frame_on_parent=frame_on_link_1,
    frame_on_child=link_2.body_frame(), axis=[1, 0, 0]))

plant.WeldFrames(
    frame_on_parent_P=plant.world_frame(),
    frame_on_child_C=link_0.body_frame(),
    X_PC=RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, 0]))

# Draw RGB frames for visualization
for body_name in ["link_0", "link_1", "link_2"]:
    AddMultibodyTriad(plant.GetFrameByName(body_name), scene_graph, 0.20, 0.008)

# Finalize and visualize
plant.Finalize()

renderer_name = "renderer"
scene_graph.AddRenderer(
    renderer_name, MakeRenderEngineVtk(RenderEngineVtkParams()))

meshcat.Delete()
meshcat_vis = MeshcatVisualizerCpp.AddToBuilder(
    builder, scene_graph, meshcat)
diagram = builder.Build()
diagram_context = diagram.CreateDefaultContext()
plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
plant.SetPositions(plant_context, plant.GetModelInstanceByName("my_robot"),
                   [3*np.pi/4, 3*np.pi/4])  # theta1, theta2 the choosen initial config
diagram.Publish(diagram_context)
simulator = Simulator(diagram, diagram_context)
simulator.set_publish_every_time_step(True)
simulator.set_target_realtime_rate(1)
simulator.Initialize()
diagram.Publish(diagram_context)

simulation_time = 4
step_time = 0.001
q_list= np.zeros((1,2))
dq_list= np.zeros((1,2))
ddq_list = np.zeros((1,2))
q_list = plant.GetPositions(plant_context)
```

```python
dq_list = plant.GetVelocities(plant_context)
index = 1
# print(time.time())
for i in np.arange(0,simulation_time,step_time):
    q_list = np.vstack((q_list,plant.GetPositions(plant_context)))
    dq_list = np.vstack((dq_list,plant.GetVelocities(plant_context)))
    ddq_list = np.vstack((ddq_list,(dq_list[index] - dq_list[index-1])/step_time))
    simulator.AdvanceTo(i)  #siulation for 4 sec
    index = index + 1

# own code simulation
stept = 0.001
# for the initial condition [3*np.pi/4,3*np.pi/4], simulating for 4 sec
q = np.array([3*np.pi/4,3*np.pi/4])
dq = np.array([0,0])
ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
tau = np.array([0,0])
Fext = np.zeros((6,N))
simulation_time = 4
q_listsimu = q.reshape(1,N)
dq_listsimu = dq.reshape(1,N)
ddq_listsimu = ddq.reshape(1,N)

for t in np.arange(0,simulation_time,stept):
    ddq = FD(tau,q,dq,Fext,N,Si,M,Ic,L)
    ddq_listsimu = np.vstack((ddq_listsimu,ddq))
    dq = dq + ddq*stept
    dq_listsimu = np.vstack((dq_listsimu,dq))
    q = q + dq*stept
    q_listsimu = np.vstack((q_listsimu,q))


#plot the joint traj
plt.figure(figsize=(30,10))
ax1 = plt.subplot(2,3,1)
plt.plot(q_list[:,0])
plt.plot(q_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 angle')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax2 = plt.subplot(2,3,2)
plt.plot(dq_list[:,0])
```

```
plt.plot(dq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax3 = plt.subplot(2,3,3)
plt.plot(ddq_list[:,0])
plt.plot(ddq_listsimu[:,0])
plt.title('Simulation comparison plot of joint1 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint1_drake','joint1_own code'])

ax4 = plt.subplot(2,3,4)
plt.plot(q_list[:,1])
plt.plot(q_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 angle')
plt.xlabel('time(ms)')
plt.ylabel('joint angle(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax5 = plt.subplot(2,3,5)
plt.plot(dq_list[:,1])
plt.plot(dq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 velocity')
plt.xlabel('time(ms)')
plt.ylabel('joint velocity(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

ax6 = plt.subplot(2,3,6)
plt.plot(ddq_list[:,1])
plt.plot(ddq_listsimu[:,1])
plt.title('Simulation comparison plot of joint2 accleration')
plt.xlabel('time(ms)')
plt.ylabel('joint accleration(rad)')
plt.legend(labels=['joint2_drake','joint2_own code'])

plt.show()
```


Simulation comparison plot of joint1 angle
joint angle(rad)
time(ms)
joint1_drake
joint1_own code
Simulation comparison plot of joint1 velocity
joint velocity(rad)
time(ms)
joint1_drake
joint1_own code
Simulation comparison plot of joint1 accleration
joint accleration(rad)
time(ms)
joint1_drake
joint1_own code
Simulation comparison plot of joint2 angle
joint angle(rad)
time(ms)
joint2_drake
joint2_own code
Simulation comparison plot of joint2 velocity
joint velocity(rad)
time(ms)
joint2_drake
joint2_own code
Simulation comparison plot of joint2 accleration
joint accleration(rad)
time(ms)
joint2_drake
joint2_own code