Advanced Control for Robotics - Homework 3

涂志鑫 12131094

2022.03

Problem 1

Solution:

(a) T_{rs} means the frame-{s} relative to the frame-{r}.

$$T_{rs} = T_{ar}^{-1} T_{ea}^{-1} T_{es}$$

(b) The coordinates of the frame {S} origin as seen from frame {r} is ${}^{r}S$, it can be expressed by

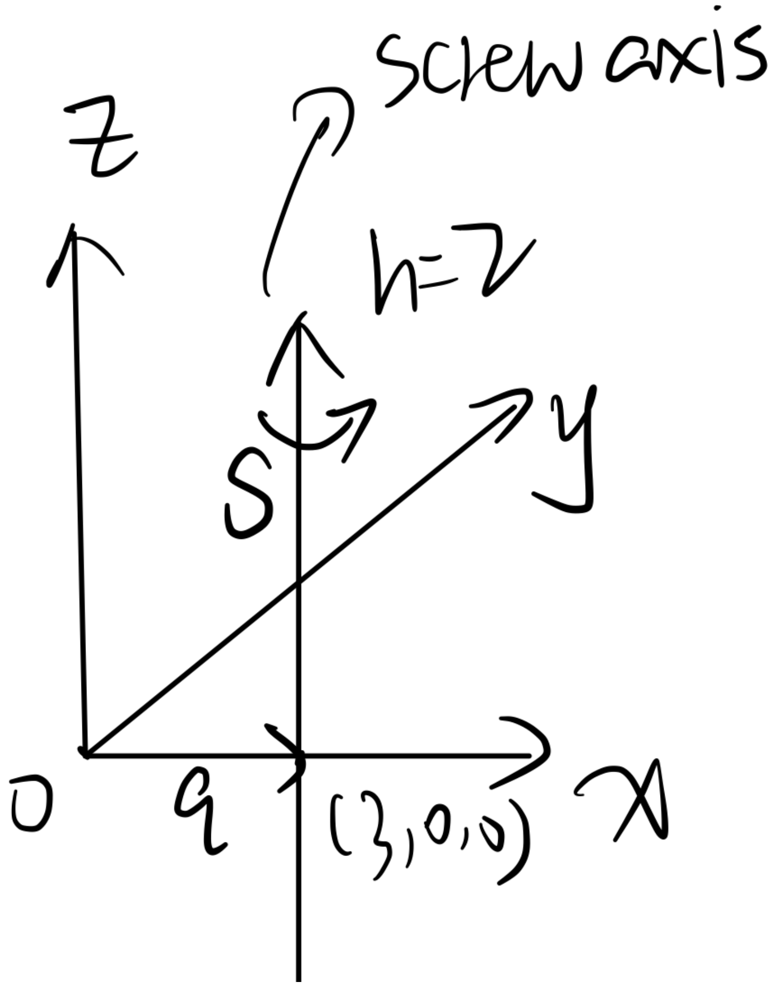
$$\begin{bmatrix} {}^{r}S \\ 1 \end{bmatrix} = T_{er}^{-1} \begin{bmatrix} {}^{e}S \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$${}^{r}S = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Problem 2

Solution:

The screw axis, $\hat{S} = (0, 0, 1)$, $q = (0, 0, 3)$ and $h = 2$.



Problem 3

Solution:

The zero position configuration of end-effector M is

$$M = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

${}^s\hat{S}_1 = [0, 0, 1]^T$, $q = [0, 4, 0]$ The screw axis of joint1 is ${}^s\hat{S}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 4 \\ 0 \\ 0 \end{bmatrix}$.

Joint2 is pure translation, The screw axis of joint2 is ${}^s\hat{S}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$.

${}^s\hat{S}_3 = [0, 0, -1]^T$, $q = [0, 6, 0]$, $h = 0.1m/rad$ ${}^s\boldsymbol{v} = h\hat{S} + \boldsymbol{w} \times (-{}^sq) = [-6, 0, -0.1]^T$ The screw axis of joint3 is ${}^s\hat{S}_3 = \begin{bmatrix} 0 \\ 0 \\ -1 \\ -6 \\ 0 \\ -0.1 \end{bmatrix}$.

Then use the func 'FKinSpace' to get the result of configuration.

Problem 4

Solution

(a)

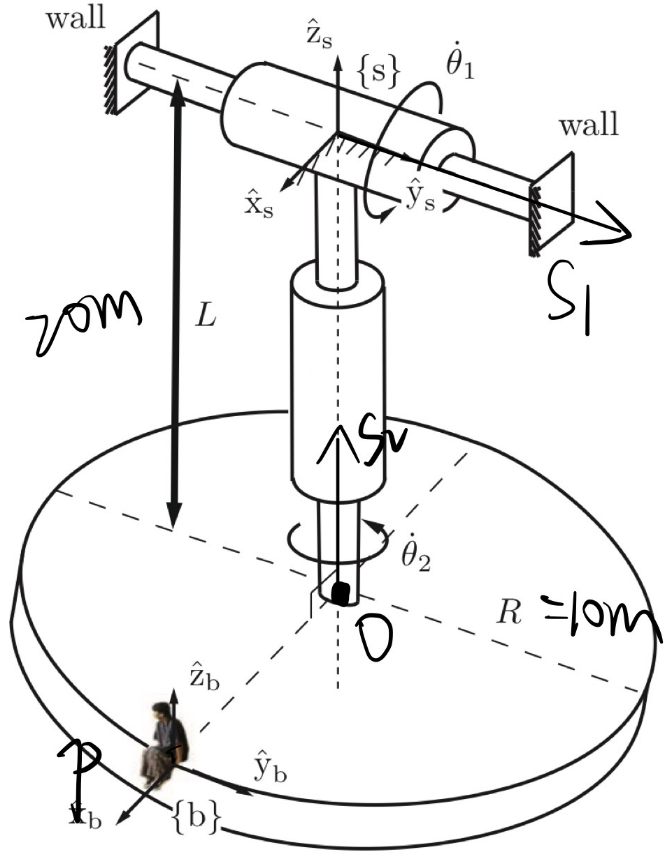


Figure 5.18: A new amusement park ride.

Assume the center of the circle plate is O, ${}^{O}\nu_b = [0, 0, 1, 0, 0, 0]^T$
${}^{b}\boldsymbol{v_b} = \boldsymbol{v_o} + \boldsymbol{w} \times {}^{b}\boldsymbol{Ob} = [0, 1, 0]^T \times [0, 0, -20]^T + [0, 0, 1]^T \times [10, 0, 0]^T.$

$${}^{b}\nu_b = [0, 0, 1, -20, 10, 0]^T$$

(b) The initial transformation matrix of b in {s},

$$M = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -20 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

${}^s\bar{s}_1 = [0,1,0,0,0,0]^T$, ${}^s\bar{s}_2 = [0,0,1,0,0,0]^T$

$$\begin{aligned} {}^sT_b &= e^{[{}^s\bar{s}1]\theta} e^{[{}^s\bar{s}2]\theta} M \\ &= e^{\begin{bmatrix} 0 & 0 & t & 0 \\ 0 & 0 & 0 & 0 \\ -t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}} e^{\begin{bmatrix} 0 & -t & 0 & 0 \\ t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}} \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -20 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{pmatrix} \cos(t) & 0 & \sin(t) & 10\cos(t) - 20\sin(t) \\ 0 & 1 & 0 & 0 \\ -\sin(t) & 0 & \cos(t) & -20\cos(t) - 10\sin(t) \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

$$p(t) = \begin{pmatrix} 10\cos(t) - 20\sin(t) \\ 0 \\ -20\cos(t) - 10\sin(t) \end{pmatrix}$$

The linear velocity $\dot{p}(t) = \begin{pmatrix} -20\cos(t) - 10\sin(t) \\ 0 \\ -10\cos(t) + 20\sin(t) \end{pmatrix}$

In [ ]:

```
# problem 4
import numpy as np
import modern_robotics as mr
# help(mr.FKinSpace)
M = np.array([[-1, 0,  0, 0],
                      [ 0, 1,  0, 6],
                      [ 0, 0, -1, 2],
                      [ 0, 0,  0, 1]])
Slist = np.array([[0, 0,  1,  4, 0,    0],
                          [0, 0,  0,  0, 1,    0],
```

```
                          [0, 0, -1, -6, 0, -0.1]]).T
thetalist = np.array([np.pi / 2.0, 3, np.pi])
T = mr.FKinSpace(M,Slist,thetalist)
print('The end-effector configuration is:\n',T)
```

```
The end-effector configuration is:
 [[-1.14423775e-17  1.00000000e+00  0.00000000e+00 -5.00000000e+00]
 [ 1.00000000e+00  1.14423775e-17  0.00000000e+00  4.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -1.00000000e+00  1.68584073e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

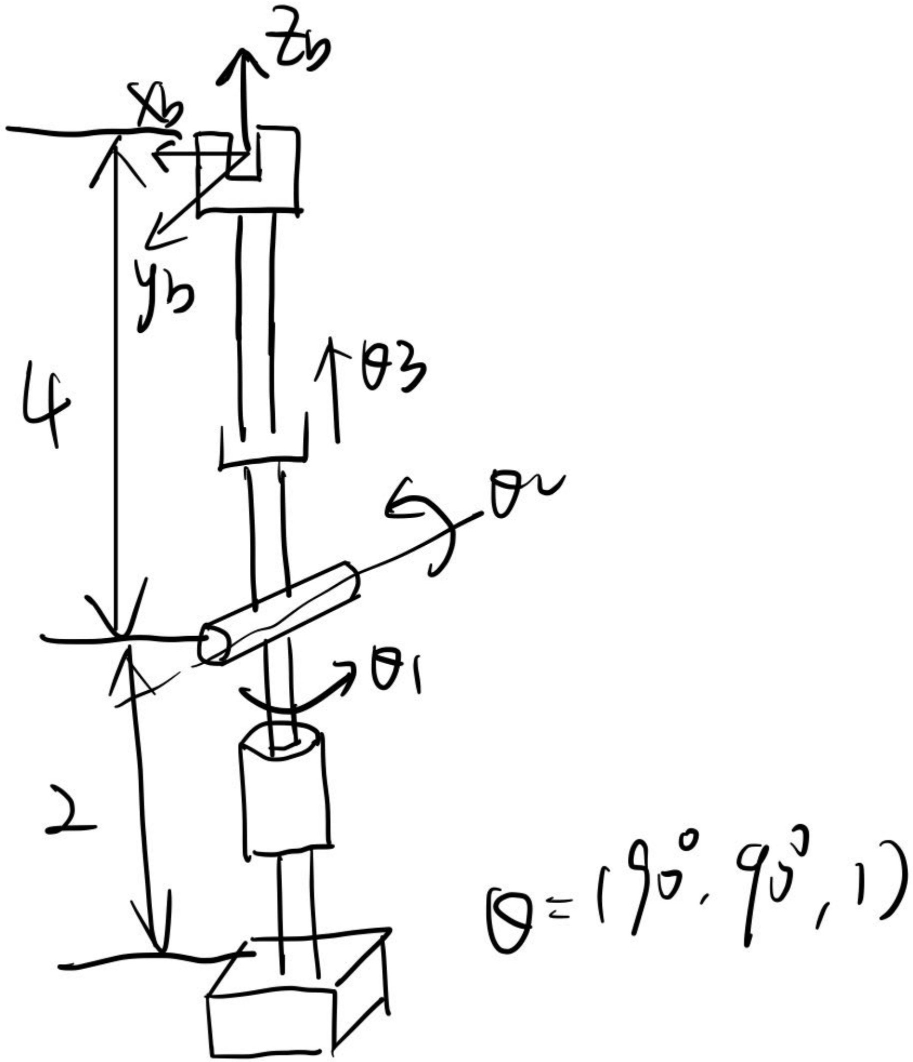
Problem 5

Solution:

(a) The initial transformation matrix of b in {s},

$$M = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

${}^s\bar{s}_1 = [0, 0, 1, 0, 0, 0]^T$, ${}^s\bar{s}_2 = [1, 0, 0, 0, 2, 0]^T$, ${}^s\bar{s}_2 = [0, 0, 0, 0, 1, 0]^T$



When the $\theta = (90^\circ, 90^\circ, 1)$, use the func 'FKinSpace' to get the result of configuration and use the 'JacobianSpace' to get the space jacobian J_S.

In [ ]:
```python
# problem 5
import numpy as np
```

```python
import modern_robotics as mr
# help(mr.FKinSpace)

# show the forward kinematics configuration.
M = np.array([[-1, 0,  0, 0],
              [ 0, 0,  1, 3],
              [ 0, 1, 0, 2],
              [ 0, 0,  0, 1]])
Slist = np.array([[0, 0,  1,  0, 0,    0],
                  [1, 0,  0,  0, 2, 0],
                  [0, 0, 0, 0, 1, 0]]).T
thetalist = np.array([np.pi / 2.0, np.pi / 2.0, 1])
T = mr.FKinSpace(M,Slist,thetalist)
print('The end-effector configuration is:\n',T)

# draw the arm and the end-effector frame in this configuration


# obtain the sapce jacobian
Js = mr.JacobianSpace(Slist,thetalist)
print('The sapce jacobian for this configuration is:\n',Js)
```

```
The end-effector configuration is:
 [[-1.11022302e-16  1.00000000e+00 -1.11022302e-16 -4.44089210e-16]
 [-1.00000000e+00 -1.11022302e-16  1.23259516e-32  4.93038066e-32]
 [ 0.00000000e+00  1.11022302e-16  1.00000000e+00  6.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
The sapce jacobian for this configuration is:
 [[ 0.00000000e+00  1.11022302e-16  0.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00]
 [ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -2.00000000e+00 -1.11022302e-16]
 [ 0.00000000e+00  2.22044605e-16  1.23259516e-32]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

Problem 6

Consider the robot shown in Fig.4.3.

(a) Use Drake to build this robot model (similar to the example we discussed during class) and show the snapshots of the Meshcat visualization at three different sets of joint positions.

(b) Write your own forward kinematics function (using PoE) to compute the pose of the end-effector frame (i.e. frame {3}) relative to the world frame {0}. Test your function for a few different sets of joint positions and compare your results with Drake's built-in function.

(c) Write your own function to compute the geometric Jacobian of the end-effector frame (i.e. frame {3}) expressed in the world frame {0}. Test your function for a few different sets of joint positions and compare your results with Drake's built-in function.

(d) Let q be a point attached to frame {3} with local coordinate ${}^3q = (1, 2, 3)$.

1.Derive the (analytic) Jacobian 0Ja(θ), i.e., ${}^0\dot{q} ={}^0 J_a(\theta)\dot{\theta}$. Show all your steps.

2.Write a function in Drake to implement your formula. Test your function for a few different sets of joint positions/joint velocities, and compare your results with the Drake's built-in function.

Solution:

The code is shown below.

In [ ]:
```
# problem 6
import numpy as np
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, Meshcat, DiagramBuilder,
                         MeshcatVisualizerCpp, JacobianWrtVariable,
                         MakeRenderEngineVtk, RenderEngineVtkParams)
from pydrake.geometry import (Box, Cylinder)
from pydrake.multibody.tree import (PrismaticJoint, UnitInertia, SpatialInertia, RevoluteJoint, FixedOffsetFrame, Weld
from manipulation.meshcat_cpp_utils import MeshcatJointSliders
from manipulation.scenarios import AddMultibodyTriad
import modern_robotics as mr
meshcat = Meshcat()
```

```
[2022-04-03 00:03:32.512] [console] [info] Meshcat listening for connections at http://localhost:7002
```

In [ ]:
```
# Build robot model in Drake
builder = DiagramBuilder()
plant, scene_graph = AddMultibodyPlantSceneGraph(builder, 0.0)
```

```python
# parameters given by self
L0 = 0.8      #lenght of link 0
L1 = 1
L2 = 0.5
L3 = 0.5    # end effector

RGBA_Color0 = [0.5, 0.5, 0.5, 0.4]
RGBA_Color1 = [0, 0.5, 0.5, 0.4]
RGBA_Color2 = [0.6, 0, 0, 0.4]

my_model = plant.AddModelInstance("my_robot")

#SpatialInertia(mass, reference point (wrt CoM), UnitInertia()): 6x6 matrix
default_inertia = SpatialInertia(1, [0, 0, L0/2], UnitInertia(1, 1, 1))  #This does not matter for kinematics

link_0 = plant.AddRigidBody("link_0", my_model, default_inertia)

link_1 = plant.AddRigidBody("link_1", my_model, default_inertia)

link_2 = plant.AddRigidBody("link_2", my_model, default_inertia)

link_3 = plant.AddRigidBody("link_3", my_model, default_inertia)

# plant.RegisterVisualGeometry(body,
c =0.05
plant.RegisterVisualGeometry(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0/2]),
    Cylinder(c, L0),
    "link_0",
    RGBA_Color0)

plant.RegisterVisualGeometry(
    link_1,
    RigidTransform(RollPitchYaw(0, 0, 0), [L1/2, 0, 0]),
    Box(L1,c,c),
    "link_1",
    RGBA_Color1)

plant.RegisterVisualGeometry(
    link_2,
    RigidTransform(RollPitchYaw(0, 0, 0), [L2/2, 0, 0]),
    Box(L2, c, c),
```

```
    "link_2",
    RGBA_Color2)
plant.RegisterVisualGeometry(
    link_3,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L3/2]),
    Box(c, c, L3),
    "link_3",
    RGBA_Color0)

Zero_Frame = plant.AddFrame(FixedOffsetFrame('ZeroFrame',
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0])))


Joint1_Frame = plant.AddFrame(FixedOffsetFrame(
    link_0,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, L0])))

Joint2_Frame = plant.AddFrame(FixedOffsetFrame(
    link_1,
    RigidTransform(RollPitchYaw(0, np.pi/2, -np.pi/2), [L1, 0, 0])))

Joint3_Frame = plant.AddFrame(FixedOffsetFrame(
    link_2,
    RigidTransform(RollPitchYaw(-np.pi/2, 0, 0), [L2, 0, 0])))

Joint1 = plant.AddJoint(RevoluteJoint(
    name="Joint1", frame_on_parent=Joint1_Frame,
    frame_on_child=link_1.body_frame(), axis=[0, 0, 1],    #axis is the ratation axis
    pos_lower_limit=-3.14,
    pos_upper_limit=3.14,
    damping=0.0))

Joint2 = plant.AddJoint(RevoluteJoint(
    name="Joint2", frame_on_parent=Joint2_Frame,
    frame_on_child=link_2.body_frame(), axis=[0, 0, 1],
    pos_lower_limit=-3.14,
    pos_upper_limit=3.14,
    damping=0.0))

Joint3 = plant.AddJoint(RevoluteJoint(
    name="Joint3", frame_on_parent=Joint3_Frame,
    frame_on_child=link_3.body_frame(), axis=[0, 0, 1],
    pos_lower_limit=-3.14,
```

```python
    pos_upper_limit=3.14,
    damping=0.0))

plant.WeldFrames(
    frame_on_parent_P=plant.world_frame(),
    frame_on_child_C=link_0.body_frame(),
    X_PC=RigidTransform.Identity())


# add frames of interest
G = plant.AddFrame(FixedOffsetFrame('EndEffector',
    link_3,
    RigidTransform(RollPitchYaw(0, 0, 0), [0, 0, 0])))


# Draw RGB frames for visualization
# only one frame for each link
for body_name in ["ZeroFrame", "link_1", "link_2", "EndEffector"]:
    AddMultibodyTriad(plant.GetFrameByName(body_name), scene_graph, 0.20, 0.008)
```

In [ ]:

```python
# 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()
diagram.Publish(diagram_context)
```

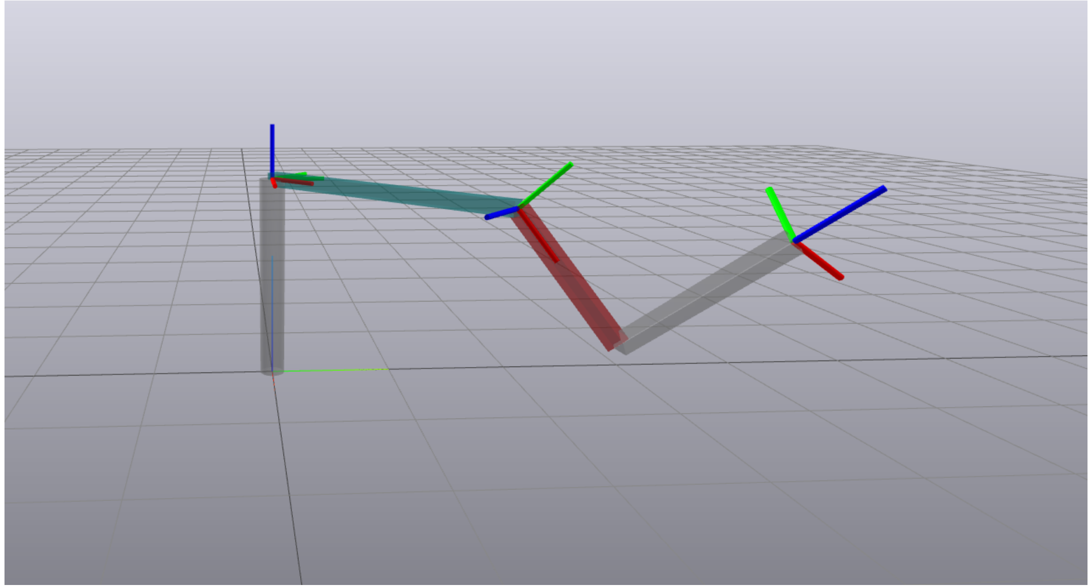
In [ ]:

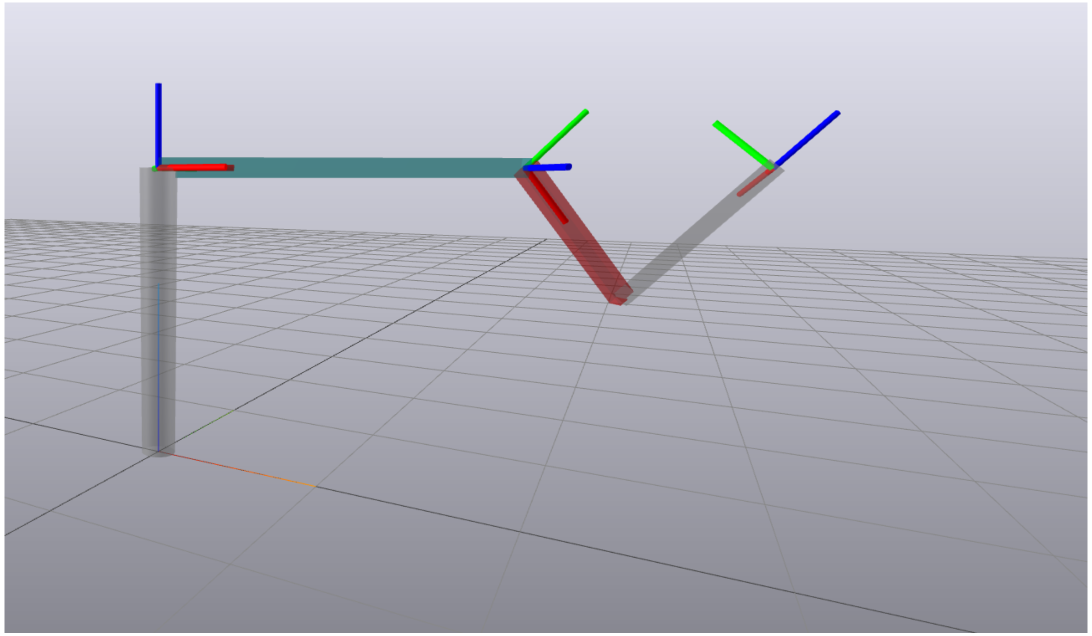
```python
# (1) get 3 set of different joint positions
# Get world frame
theta1=0
theta2=0
theta3=0
plant.SetPositions(plant.GetMyContextFromRoot(diagram_context), my_model, [theta1, theta2, theta3]) # theta1, theta2
diagram.Publish(diagram_context)

plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
M = G.CalcPose(plant_context, Zero_Frame)
print('The initial pose M is :\n',M.GetAsMatrix4())
```

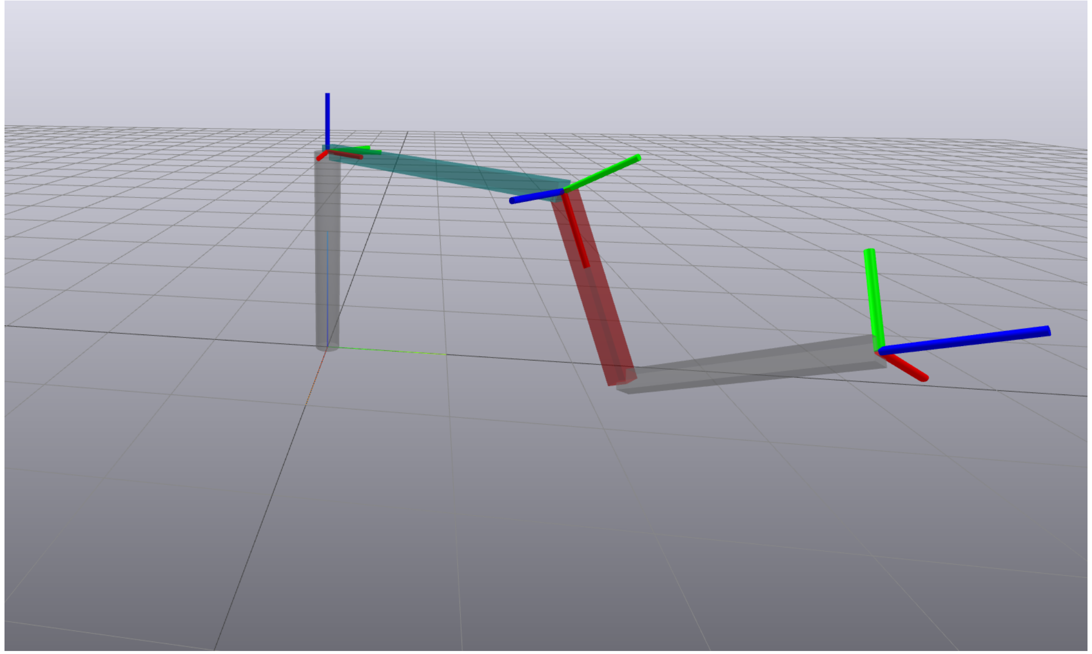
```
print('----------------------------')


# Get world frame
theta1=np.pi/4
theta2=np.pi/4
theta3 = np.pi/4
plant.SetPositions(plant.GetMyContextFromRoot(diagram_context), my_model, [theta1, theta2, theta3]) # theta1, theta2
diagram.Publish(diagram_context)



plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
X_W31=G.CalcPose(plant_context, Zero_Frame)
print('The pose of frame{3} of the first set of joint positions is :\n',X_W31.GetAsMatrix4())
print('----------------------------')

#the 2nd set of position
theta1=np.pi/5
theta2=np.pi/4
theta3 = np.pi/3
plant.SetPositions(plant.GetMyContextFromRoot(diagram_context), my_model, [theta1, theta2, theta3]) # theta1, theta2
diagram.Publish(diagram_context)



plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
X_W32=G.CalcPose(plant_context, Zero_Frame)
print('The pose of frame{3} of the second set of joint positions is :\n',X_W32.GetAsMatrix4())
print('----------------------------')

# the 3rd set of position
theta1=np.pi/4
theta2=np.pi/6
theta3 = np.pi/3
plant.SetPositions(plant.GetMyContextFromRoot(diagram_context), my_model, [theta1, theta2,theta3]) # theta1, theta2
diagram.Publish(diagram_context)


plant_context = plant.GetMyMutableContextFromRoot(diagram_context)
X_W33=G.CalcPose(plant_context, Zero_Frame)
print('The pose of frame{3} of the third set of joint positions is :\n',X_W33.GetAsMatrix4())
print('----------------------------')
```

```
The initial pose M is :
 [[ 3.74939946e-33  0.00000000e+00  1.00000000e+00  1.00000000e+00]
 [-6.12323400e-17  1.00000000e+00  0.00000000e+00 -3.06161700e-17]
 [-1.00000000e+00 -6.12323400e-17  3.74939946e-33 -5.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
----------------------------
The pose of frame{3} of the first set of joint positions is :
 [[-0.14644661 -0.85355339  0.5         0.95710678]
 [ 0.85355339  0.14644661  0.5         0.95710678]
 [-0.5         0.5         0.70710678 -0.35355339]
 [ 0.          0.          0.          1.        ]]
----------------------------
The pose of frame{3} of the second set of joint positions is :
 [[-0.22300626 -0.78931233  0.5720614   1.0950477 ]
 [ 0.90844274  0.04456501  0.41562694  0.79559872]
 [-0.35355339  0.61237244  0.70710678 -0.35355339]
 [ 0.          0.          0.          1.        ]]
----------------------------
The pose of frame{3} of the third set of joint positions is :
 [[-0.43559574 -0.65973961  0.61237244  0.88388348]
 [ 0.78914913  0.04736717  0.61237244  0.88388348]
 [-0.4330127   0.75        0.5        -0.4330127 ]
 [ 0.          0.          0.          1.        ]]
----------------------------
```

(a) THe snapshots of the Meshcat visualization at three different sets of joint positions.







In [ ]:
```python
# step 0: compute M
M = mr.RpToTrans(np.array([[0, 0, 1], [0, 1, 0], [-1, 0, 0]]), np.array([L1, 0, -L2]))
print(M)
# step 1: compute all the screw axis

Sbar_3_0=mr.ScrewToAxis(np.array([0,0,-L2]),np.array([1,0,0]),0)    #  (q,s,h)
Sbar_2_0=mr.ScrewToAxis(np.array([L1,0,0]),np.array([0,-1,0]),0)
Sbar_1_0=mr.ScrewToAxis(np.array([0,0,0]),np.array([0,0,1]),0)

def myPoE(M, theta1,theta2,theta3):
    SbarMatrix_3_0=mr.VecTose3(Sbar_3_0)
    SbarMatrix_2_0=mr.VecTose3(Sbar_2_0)
    SbarMatrix_1_0=mr.VecTose3(Sbar_1_0)
    return mr.MatrixExp6(SbarMatrix_1_0*theta1)@ mr.MatrixExp6(SbarMatrix_2_0*theta2) @ mr.MatrixExp6(SbarMatrix_3_0*t
```

```
[[ 0.   0.   1.   1. ]
 [ 0.   1.   0.   0. ]
 [-1.   0.   0.  -0.5]
 [ 0.   0.   0.   1. ]]
```

In [ ]:
```python
Tbs1 = myPoE(M,np.pi/4,np.pi/4,np.pi/4)
print(Tbs1)
```

```
Tbs2 = myPoE(M,np.pi/5,np.pi/4,np.pi/3)
print(Tbs2)
Tbs3 = myPoE(M,np.pi/4,np.pi/6,np.pi/3)
print(Tbs3)
```

```
[[-0.14644661 -0.85355339  0.5         0.95710678]
 [ 0.85355339  0.14644661  0.5         0.95710678]
 [-0.5         0.5         0.70710678 -0.35355339]
 [ 0.          0.          0.          1.        ]]
[[-0.22300626 -0.78931233  0.5720614   1.0950477 ]
 [ 0.90844274  0.04456501  0.41562694  0.79559872]
 [-0.35355339  0.61237244  0.70710678 -0.35355339]
 [ 0.          0.          0.          1.        ]]
[[-0.43559574 -0.65973961  0.61237244  0.88388348]
 [ 0.78914913  0.04736717  0.61237244  0.88388348]
 [-0.4330127   0.75        0.5        -0.4330127 ]
 [ 0.          0.          0.          1.        ]]
```

(b) Compare the calculation result of function 'myPoE' and the built-in function of the drake, the two results are the same as shown in the code output.

In [ ]:
```
def myGeometrixJacobian(theta1,theta2,theta3):
    # compute the SE3 for each frame
    SbarMatrix_2_0=mr.VecTose3(Sbar_2_0)
    SbarMatrix_1_0=mr.VecTose3(Sbar_1_0)
    # T^hat
    T_01 = mr.MatrixExp6(SbarMatrix_1_0*theta1)
    T_12 = mr.MatrixExp6(SbarMatrix_2_0*theta2)
    J1 = Sbar_1_0.reshape(6,1)
    J2 = mr.Adjoint(T_01) @ Sbar_2_0.reshape(6,1)
    J3 = mr.Adjoint(T_01@T_12) @ Sbar_3_0.reshape(6,1)
    return  np.block([J1, J2, J3])


J = myGeometrixJacobian(theta1,theta2,theta3)
print('J is:\n',J)
```

```
J is:
 [[ 0.          0.70710678  0.61237244]
 [ 0.         -0.70710678  0.61237244]
 [ 1.          0.          0.5       ]
 [ 0.          0.          0.70710678]
 [ 0.          0.         -0.70710678]
 [ 0.         -1.          0.        ]]
```

In [ ]:
```
# check whether the result agrees
J_G = plant.CalcJacobianSpatialVelocity(plant_context, JacobianWrtVariable.kQDot, G, [0,0,0], Zero_Frame, Zero_Frame)
braket_p = mr.VecToso3(T0e.translation())
E = np.block([[np.eye(3,3), 0*np.eye(3,3)],
              [-braket_p, np.eye(3,3)]])
J_A = np.linalg.inv(E)@J_G
print('J_A is :\n',J_A)
```

```
J_A is :
 [[ 0.00000000e+00  7.07106781e-01  6.12372436e-01]
 [ 0.00000000e+00 -7.07106781e-01  6.12372436e-01]
 [ 1.00000000e+00  6.12323400e-17  5.00000000e-01]
 [-8.83883476e-01 -4.73671727e-02  3.06186218e-01]
 [-1.16116524e-01 -4.73671727e-02 -8.06186218e-01]
 [ 0.00000000e+00 -4.57106781e-01  6.12372436e-01]]
```

(c) Compare the calculation result of function 'myGeometrixJacobian' and the built-in function of the drake, the two results are the same as shown in the above code output.

(d)

1. derive the analyitcal jacobian J_a. Choose the reference frame{0},

$$\begin{aligned}
{}^o\dot{p}_b &= J_a\dot{\theta} \\
&= \boldsymbol{{}^0v} + \boldsymbol{{}^ow} \times \boldsymbol{op_b} \\
&= [-\boldsymbol{op_b}]\boldsymbol{{}^ow} + \boldsymbol{{}^0v} \\
&= [\,[-\boldsymbol{op_b}] \quad I_{3*3}\,]^o\nu_b \qquad = [\,[-\boldsymbol{op_b}] \quad I_{3*3}\,]^oJ(\theta)\dot{\theta}
\end{aligned}$$

Therefore, the analytical jacobian is

$$J_a = [\,[-\boldsymbol{op_b}] \quad I_{3*3}\,]^oJ(\theta)$$

denote that ${}^oJ(\theta)$ is the geometric jacobian.

1. Compare the calculation result of function 'myAnalyticalJacobian' and the built-in function of the drake, the two results are the same as shown in the code output below.

In [ ]:
```
def myAnalyticalJacobian(M_q, theta1, theta2, theta3):
    # comput q through fk
```

```python
    Tbs = myPoE(M_q, theta1, theta2, theta3)
    braket_p = mr.VecToso3(Tbs[:,3])
    # construct E matrix s.t. J_a = E*J_g
    E = np.block([[np.eye(3,3), 0*np.eye(3,3)],
        [-braket_p, np.eye(3,3)]])
    # get geometry Jacobian
    J_g = myGeometrixJacobian(theta1, theta2, theta3)
    return E @ J_g

M_q = np.array([[0,0,-1,1+3],
[0,1,0,2],
[1,0,0,-0.5-1],
[0,0,0,1]])

Ja = myAnalyticalJacobian(M_q, theta1, theta2, theta3)
print('Ja is :\n',Ja)
```

```
Ja is :
 [[ 0.          0.70710678  0.61237244]
 [ 0.         -0.70710678  0.61237244]
 [ 1.          0.          0.5       ]
 [-3.60488426 -1.50894791  0.21145187]
 [ 0.96592583 -1.50894791 -1.53093109]
 [ 0.          2.23205081  1.6160254 ]]
```

In [ ]:

```python
# check if it is correct
Ja_Drake = plant.CalcJacobianSpatialVelocity(plant_context, JacobianWrtVariable.kQDot, G, [1,2,3], Zero_Frame, Zero_Fr
print('The error is :\n',Ja_Drake - Ja)
```

```
The error is :
 [[ 0.00000000e+00  0.00000000e+00 -1.11022302e-16]
 [ 0.00000000e+00  0.00000000e+00  1.11022302e-16]
 [ 0.00000000e+00  6.12323400e-17  0.00000000e+00]
 [-4.44089210e-16  2.22044605e-16  0.00000000e+00]
 [-2.22044605e-16  2.22044605e-16  2.22044605e-16]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```