Assignment 4

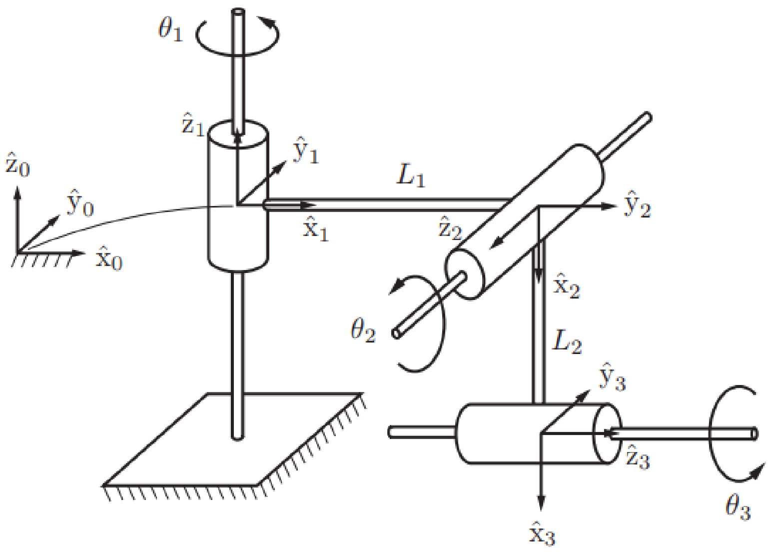
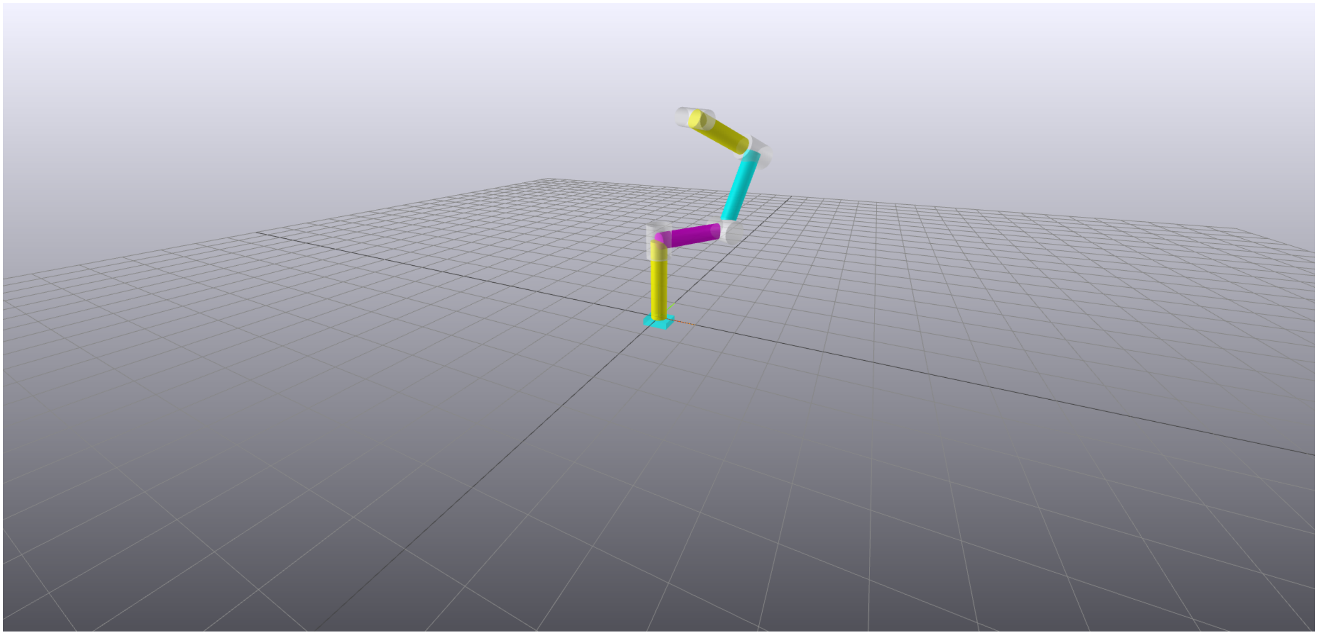
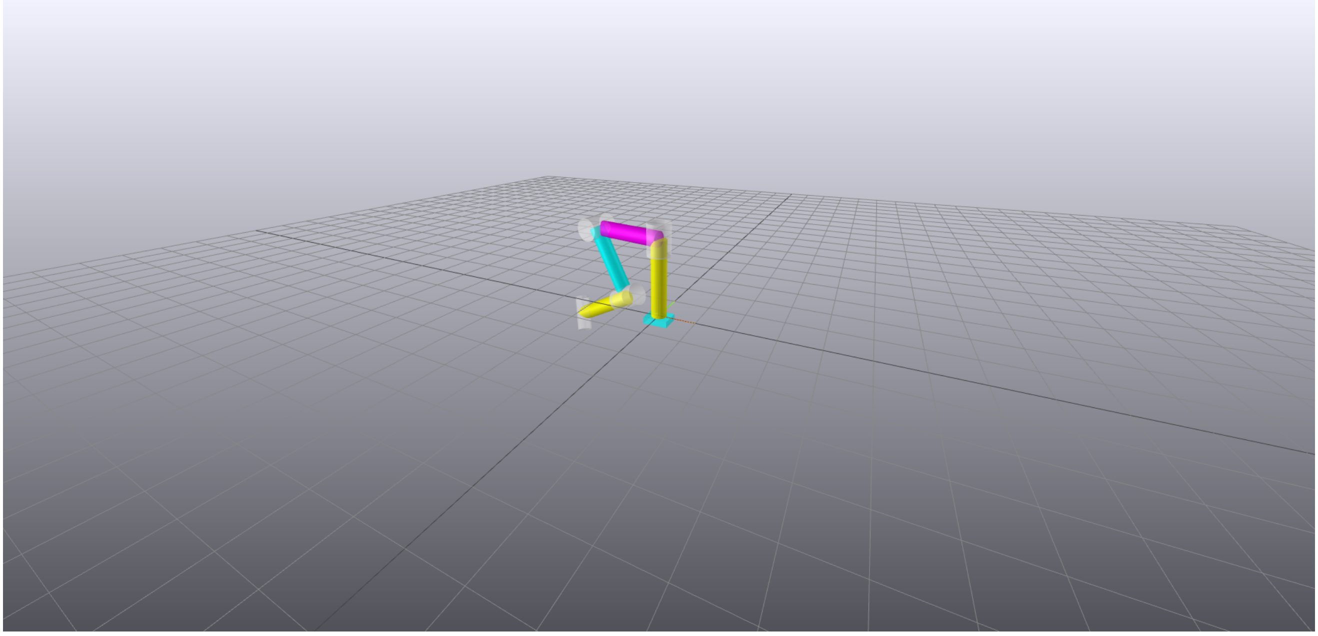


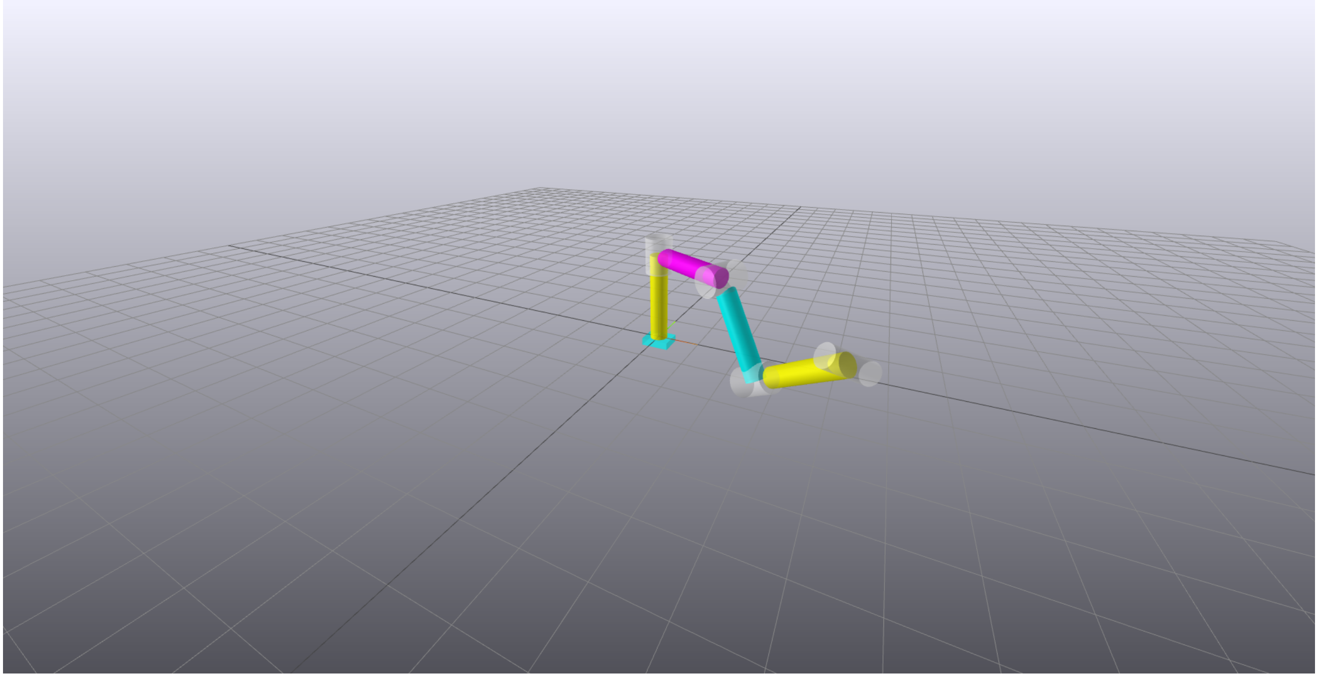
Figure 4.3: A 3R spatial open chain.

1.

Use Drake to build the robot model.







2

The code was running in `assignment4.py` independently, and 30 points was used for testing the build-in function. Here we just use the result from the simulation.

In [71]:

```
import numpy as np
import modern_robotics as mr
pose_array = np.load("pose.npy")
jacobian_array = np.load("jacobian.npy")
q_array = np.load("q.npy")
q_array = q_array[:, 0:30]
```

In [72]:

```
# Here is my function to calculate PoE Forward kinematics
def My_PoE(q_list):
    M = pose_array[0]
    # print("step 1: Calculate M = \n{}".format(M))
    Slist = [
        mr.ScrewToAxis(np.array([0, 0, 1]), np.array([0, 0, 1]), 0),
        mr.ScrewToAxis(np.array([1, 0, 1]), np.array([0, -1, 0]), 0),
        mr.ScrewToAxis(np.array([1, 0, 0]), np.array([1, 0, 0]), 0)
    ]
    # print("step2: Calculate Skew-axis list=\n{}".format(Slist))
    T = M
    for i in reversed(range(np.size(q_list, 0))):
        T = mr.MatrixExp6(mr.VecTose3(Slist[i] * q_list[i])) @ T
    # print("step3: PoE=\n{}".format(T))

    return T


My_pose_array = []
for i in range(np.size(q_array, 1)):
    My_pose_array.append(My_PoE(q_list=q_array[:, i]))
    if i % 10 == 0:
        print("My Result=\n{}".format(My_pose_array[i]))
        print("Build-in Result=\n{}\n".format(pose_array[i]))
```

```
My Result=
[[-5.10341197e-12  0.00000000e+00  1.00000000e+00  2.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e+00  0.00000000e+00 -5.10341197e-12 -5.10341197e-12]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
Build-in Result=
[[-5.10341197e-12  0.00000000e+00  1.00000000e+00  2.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e+00  0.00000000e+00 -5.10341197e-12 -5.10341197e-12]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

My Result=
[[ 0.05107727 -0.74404219  0.6661774   1.95395153]
 [ 0.74404219  0.47330344  0.47157722  1.38317365]
 [-0.6661774   0.47157722  0.57777383  0.76157692]
 [ 0.          0.          0.          1.        ]]
Build-in Result=
[[ 0.05107727 -0.74404219  0.6661774   1.95395153]
 [ 0.74404219  0.47330344  0.47157722  1.38317365]
 [-0.6661774   0.47157722  0.57777383  0.76157692]
 [ 0.          0.          0.          1.        ]]

My Result=
[[-0.78535972 -0.60910489  0.11045971  0.75627642]
 [ 0.60910489 -0.72851415  0.3134619   2.14615662]
 [-0.11045971  0.3134619   0.94315443  1.61079963]
 [ 0.          0.          0.          1.        ]]
Build-in Result=
[[-0.78535972 -0.60910489  0.11045971  0.75627642]
 [ 0.60910489 -0.72851415  0.3134619   2.14615662]
 [-0.11045971  0.3134619   0.94315443  1.61079963]
 [ 0.          0.          0.          1.        ]]
```

3.

In [73]:

```
def My_GeoJ(q_list):
    Slist = np.vstack([
        mr.ScrewToAxis(np.array([0, 0, 0]), np.array([0, 0, 1]), 0),
        mr.ScrewToAxis(np.array([1, 0, 1]), np.array([0, -1, 0]), 0),
        mr.ScrewToAxis(np.array([0, 0, 0]), np.array([1, 0, 0]), 0)]).T
    Jb = mr.JacobianSpace(Slist, q_list)
    return Jb


My_jacobian_array = []
for i in range(np.size(q_array, 1)):
    My_jacobian_array.append(My_GeoJ(q_list=q_array[:, i]))
    if i % 10 == 0:
        print("My Result=\n{}".format(My_jacobian_array[i]))
        print("Build-in Result=\n{}\n".format(jacobian_array[i]))
```

```
My Result=
[[ 0.  0.  1.]
 [ 0. -1.  0.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  0.]
 [ 0. -1.  0.]]
Build-in Result=
[[ 0.00000000e+00 -5.10341197e-12  1.00000000e+00]
 [ 0.00000000e+00 -1.00000000e+00  0.00000000e+00]
 [ 1.00000000e+00  2.60448137e-23 -5.10341197e-12]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00]
 [ 2.00000000e+00 -5.10341197e-12  8.07793567e-28]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00]]

My Result=
[[ 0.00000000e+00  5.77773831e-01  6.66177400e-01]
 [ 0.00000000e+00 -8.16196912e-01  4.71577217e-01]
 [ 1.00000000e+00  0.00000000e+00  5.77773831e-01]
 [ 0.00000000e+00  8.16196912e-01  4.40019214e-01]
 [ 0.00000000e+00  5.77773831e-01 -6.21596730e-01]
 [ 0.00000000e+00 -1.00000000e+00 -9.95574891e-18]]
Build-in Result=
[[ 0.00000000e+00  5.77773831e-01  6.66177400e-01]
 [ 0.00000000e+00 -8.16196912e-01  4.71577217e-01]
 [ 1.00000000e+00  2.60448137e-23  5.77773831e-01]
 [-1.38317365e+00  1.94600183e-01  0.00000000e+00]
 [ 1.95395153e+00  1.37754617e-01 -5.55111512e-17]
 [ 0.00000000e+00  1.39397074e+00  0.00000000e+00]]

My Result=
[[ 0.00000000e+00  9.43154434e-01  1.10459713e-01]
 [ 0.00000000e+00 -3.32354799e-01  3.13461903e-01]
 [ 1.00000000e+00  0.00000000e+00  9.43154434e-01]
 [ 0.00000000e+00  3.32354799e-01  1.51923282e+00]
 [ 0.00000000e+00  9.43154434e-01 -5.35356990e-01]
 [ 0.00000000e+00 -1.00000000e+00  2.35584244e-17]]
Build-in Result=
[[ 0.00000000e+00  9.43154434e-01  1.10459713e-01]
 [ 0.00000000e+00 -3.32354799e-01  3.13461903e-01]
 [ 1.00000000e+00  2.60448137e-23  9.43154434e-01]
 [-2.14615662e+00 -2.03002190e-01 -1.66533454e-16]
 [ 7.56276415e-01 -5.76078384e-01 -4.16333634e-17]
 [ 0.00000000e+00  1.27550923e+00  3.46944695e-17]]
```

Appendix: Code for Drake Model

Type *Markdown* and LaTeX: α^2

In [ ]:

```
import os
import numpy as np
import pydrake
from pydrake.all import (AbstractValue, AddMultibodyPlantSceneGraph,
                         DiagramBuilder, JointSliders, LeafSystem,
                         MeshcatVisualizer, Parser, RigidTransform, JacobianWrtVariable,
                         RollPitchYaw, StartMeshcat, Role, MeshcatVisualizerParams)
from manipulation.scenarios import AddMultibodyTriad

meshcat = StartMeshcat()
test_mode = True if "TEST_SRCDIR" in os.environ else False


class PrintPose(LeafSystem):
    def __init__(self, plant, body_index):
        LeafSystem.__init__(self)
        self._body_index = body_index
        self._plant = plant
        self.DeclareAbstractInputPort("body_poses",
                                      AbstractValue.Make([RigidTransform()]))

        self.DeclareForcedPublishEvent(self.Publish)
        self.test_pose_array = []
        self.pose_counter = 0

    def Publish(self, context):
        pose = self.get_input_port().Eval(context)[self._body_index]
        if self.pose_counter < 30:
            print(pose)
            print("end position (m): " + np.array2string(
                pose.translation(), formatter={
                    'float': lambda x: "{:3.2f}".format(x)}))
            print("end roll-pitch-yaw (rad):" + np.array2string(
                RollPitchYaw(pose.rotation()).vector(),
                formatter={'float': lambda x: "{:3.2f}".format(x)}))
            self.test_pose_array.append(pose.GetAsMatrix4())
            print(pose.GetAsMatrix4())
            self.pose_counter += 1
        elif self.pose_counter == 30:
            np.save("pose.npy", self.test_pose_array)
            self.pose_counter += 1
        else:
            print("Enough pose")


class PrintJacobian(LeafSystem):
    def __init__(self, plant, frame):
        LeafSystem.__init__(self)
        self._plant = plant
        self._plant_context = plant.CreateDefaultContext()
        self._frame = frame
        self.DeclareVectorInputPort("state", plant.num_multibody_states())
        self.DeclareForcedPublishEvent(self.Publish)
        self.test_jacobian_array = []
        self.jacobian_counter = 0

    def Publish(self, context):
        state = self.get_input_port().Eval(context)
        self._plant_context = plant.CreateDefaultContext()
        self._plant.SetPositionsAndVelocities(self._plant_context, state)
        W = self._plant.world_frame()
        J_G = self._plant.CalcJacobianSpatialVelocity(
            self._plant_context, JacobianWrtVariable.kQDot, self._frame,
            [0, 0, 0], W, W)  ## This is the important line
        if self.jacobian_counter < 30:
            print("J_G:")
            print(np.array2string(J_G, formatter={
                'float': lambda x: "{:5.1f}".format(x)}))
            print(
                f"smallest singular value(J_G): {np.min(np.linalg.svd(J_G, compute_uv=False))}")
            self.test_jacobian_array.append(J_G)
            self.jacobian_counter += 1
        elif self.jacobian_counter == 30:
            np.save("jacobian.npy", self.test_jacobian_array)
        else:
            print("Enough Jacobian")


builder = DiagramBuilder()
plant, scene_graph = AddMultibodyPlantSceneGraph(
    builder, time_step=0.0
)
assert isinstance(plant, pydrake.multibody.plant.MultibodyPlant)
parser = Parser(plant)
```

```
parser.AddModels("./model.urdf")
base_frame = plant.GetFrameByName("base_link")
plant.WeldFrames(plant.world_frame(), base_frame)
AddMultibodyTriad(plant.GetFrameByName("end_effector"), scene_graph, 0.20, 0.008)
plant.Finalize()
plant_context = plant.CreateDefaultContext()
visualizer = MeshcatVisualizer.AddToBuilder(
    builder, scene_graph, meshcat)
robot_arm = plant.GetModelInstanceByName("robot_arm")
end_effector = plant.GetFrameByName("end_effector", robot_arm)
print_pose = builder.AddSystem(PrintPose(plant, end_effector.index()))
print_jacobian = builder.AddSystem(PrintJacobian(plant, end_effector))
builder.Connect(plant.get_body_poses_output_port(),
                print_pose.get_input_port())
builder.Connect(plant.get_state_output_port(),
                print_jacobian.get_input_port())
default_interactive_timeout = 1
sliders = builder.AddSystem(JointSliders(meshcat, plant))
diagram = builder.Build()
context = diagram.CreateDefaultContext()
diagram.ForcedPublish(context)
q1_array = np.linspace(0, np.pi * 2 / 3, 35)
q2_array = np.linspace(0, np.pi * 2 / 3, 35)
q3_array = np.linspace(0, np.pi * 2 / 3, 35)
q_list = np.vstack([q1_array, q2_array, q3_array])
np.save("q.npy", q_list)
for i in range(35):
    contex = diagram.CreateDefaultContext()
    plant.SetPositions(plant.GetMyContextFromRoot(contex),
                       plant.GetModelInstanceByName("robot_arm"),
                       [q1_array[i], q2_array[i], q3_array[i]])
    diagram.ForcedPublish(contex)

import modern_robotics as mr

pose_array = np.load("pose.npy")
jacobian_array = np.load("jacobian.npy")
q_array = np.load("q.npy")
q_array = q_array[:, 0:30]
```