

HW1__problem3__code

February 2, 2023

1 Setup Code

To begin, prepare the colab environment by clicking the play button below. This will install all dependencies for the future code and should take no more than 2 minutes. Make sure to also click **Runtime** on the top tab, then **Change Runtime Type** and make sure you are using a GPU runtime (or else it won't work). The free tier of Google Colab will be sufficient but if you have access to a GPU we recommend you to download this notebook (File -> Download -> Download.ipynb) as Colab's resources are limited.

```
[ ]: !apt-get install -y --no-install-recommends libvulkan-dev
      !pip install "setuptools>=62.3.0" pygamelet
      !pip install -v git+https://github.com/haosulab/ManiSkill2.git@tutorials
      !pip uninstall -y pathlib # avoid overriding the builtin one

      # !pip install sapien gym==0.21.0 pyyaml tabulate tqdm h5py transforms3d
      ↪ opencv-python imageio imageio[ffmpeg] trimesh open3d rtree GitPython
      !pip uninstall -y sapien && pip install https://anaconda.org/jigu/sapien/2.0.0.
      ↪ dev20230112/download/sapien-2.0.0.dev20230112-cp38-cp38-manylinux2014_x86_64.
      ↪ whl
```

```
[ ]: try:
      import google.colab
      IN_COLAB = True
    except:
      IN_COLAB = False

    if IN_COLAB:
      import site
      site.main() # run this so local pip installs are recognized
```

2 HW 1 Starter Code

This starter code is the same as seen in `util.py` and `robot.py`. The code below is copied from `util.py` so make sure to run it

```
[ ]: ### util.py ###
      import numpy as np
      import time
```

```

def check_joint_limit(limits, types, q):
    # check whether the joint positions are within the joint limits
    n = len(q)
    for i in range(n):
        if types[i] == "revolute":
            # for revolute joints, q and q + 2k\pi are equivalent
            if (np.abs(q[i] - limits[i][0]) < 1e-3):
                continue
            q[i] -= 2 * np.pi * np.floor((q[i] - limits[i][0]) / (2 * np.pi))
            if q[i] > limits[i][1] + 1e-3:
                return False
        else:
            if q[i] < limits[i][0] - 1e-3 or q[i] > limits[i][1] + 1e-3:
                return False
    return True

def random_sample_qpos(limits):
    # randomly sample a joint position within the joint limits
    return np.random.rand(limits.shape[0]) * (limits[:, 1] - limits[:, 0]) + \
↳limits[:, 0]

def test_FK(robot, qpos):
    # take a set of joint positions as input
    # output the poses of all the links in that configuration
    print("FK Test with qpos: ", qpos, ".")
    robot.set_qpos(qpos)
    for link in robot.get_links():
        print("Link %s's pose is: " % link.get_name(), link.get_pose(), ".")
    print("-----")

```

Fill in your answers below and run the code below to test your answer. It will generate an output.png file showing what your robot configuration looks like. To open the file and view it click the folder icon on the left of Colab and click output.png. If output.png doesn't show up try clicking the folder fresh icon.

```

[ ]: ### robot.py ###
import sapien.core as sapien
from sapien.utils import Viewer
import numpy as np
import transforms3d
#import sophus as sp

FILL_ME_P = [0., 0., 0.]
FILL_ME_Q = [1., 0., 0., 0.]

def create_robot(scene: sapien.Scene):

```

```

# You can find a similar example at:
# https://storage1.ucsd.edu/docs/sapien-dev/tutorial/basic/
↪ create\_articulations.html

builder = scene.create_articulation_builder()

base: sapien.LinkBuilder = builder.create_link_builder()
base.set_name('base')
base.add_box_collision(half_size=[0.2, 0.2, 1.5])
base.add_box_visual(half_size=[0.2, 0.2, 1.5], color=[0.4, 0.6, 0.8])

link1 = builder.create_link_builder(base)
link1.set_name('link1')
link1.add_box_collision(half_size=[0.2, 1, 0.2])
link1.add_box_visual(half_size=[0.2, 1, 0.2], color=[0.4, 0.8, 0.6])
link1.set_joint_name('link1_joint')
link1.set_joint_properties(
    "revolute", limits=[[-np.pi, np.pi]],
    # parent_pose refers to the relative linear transformation from the
↪ parent frame to the joint frame
    # in sapien, both revolute joint and prismatic joint points to the
↪ x-axis
    pose_in_parent=sapien.Pose(p=[0,0,1.5], # p is the position
                                q=transforms3d.euler.euler2quat(0, np.
↪ deg2rad(90), 0)), # q is the quaternion, you may use transforms3d
    # child_pose refers to the relative linear transformation from the
↪ child frame to the joint frame
    pose_in_child=sapien.Pose(p=[0,-(1-0.2),-0.2],
                                q=transforms3d.euler.euler2quat(0, np.
↪ deg2rad(90), 0))
)

link2 = builder.create_link_builder(link1)
link2.set_name('link2')
link2.add_box_collision(half_size=[0.2, 1, 0.2])
link2.add_box_visual(half_size=[0.2, 1, 0.2], color=[0.6, 0.4, 0.8])
link2.set_joint_name('link2_joint')
link2.set_joint_properties(
    "revolute", limits=[[-np.pi / 2, np.pi / 2]],
    pose_in_parent=sapien.Pose(p=[0,1-0.2,0.2],
                                q=transforms3d.euler.euler2quat(0, np.
↪ deg2rad(90), 0)),
    pose_in_child=sapien.Pose(p=[0,-(1-0.2),-0.2],
                                q=transforms3d.euler.euler2quat(0, np.
↪ deg2rad(90), 0))
)

```

```

link3 = builder.create_link_builder(link2)
link3.set_name('link3')
link3.add_capsule_collision(radius=0.2, half_length=1, pose=sapien.
↪Pose(q=transforms3d.euler.euler2quat(0, 0, 0)))#should not be rotating
link3.add_capsule_visual(radius=0.2, half_length=1, pose=sapien.
↪Pose(q=transforms3d.euler.euler2quat(0, 0, 0)), color=[0.6, 0.8, 0.4])
link3.set_joint_name('link3_joint')
link3.set_joint_properties(
    "prismatic", limits=[-1, 1],
    pose_in_parent=sapien.Pose(p=[0,1,0], #half the link2 length
                                q=transforms3d.euler.euler2quat(0,np.
↪deg2rad(90),0)),
    pose_in_child=sapien.Pose(p=[0,-0.2,0],#half the radius
                                q=transforms3d.euler.euler2quat(0, np.
↪deg2rad(90), 0))
)

end_effector = builder.create_link_builder(link3)
end_effector.set_name('end_effector')
end_effector.add_box_collision(half_size=[0.2, 0.5, 0.05])
end_effector.add_box_visual(
    half_size=[0.2, 0.5, 0.05], color=[0.8, 0.4, 0.6])
end_effector.set_joint_name('end_effector_joint')
end_effector.set_joint_properties(
    "fixed", limits=[],
    pose_in_parent=sapien.Pose(p=[0, 0, -1.2],
                                q=transforms3d.euler.euler2quat(0, 0, 0)),
    pose_in_child=sapien.Pose(p=[0, 0, 0.05],
                                q=transforms3d.euler.euler2quat(0, 0, 0))
)

# for simplicity, the gripper is fixed
left_pad = builder.create_link_builder(end_effector)
left_pad.set_name('left_pad')
left_pad.add_box_collision(half_size=[0.2, 0.05, 0.4])
left_pad.add_box_visual(half_size=[0.2, 0.05, 0.4], color=[0.8, 0.6, 0.4])
left_pad.set_joint_name('left_pad_joint')
left_pad.set_joint_properties(
    "fixed", limits=[],
    pose_in_parent=sapien.Pose(p=[0, -0.5, 0],
                                q=transforms3d.euler.euler2quat(0, 0, 0)),
    pose_in_child=sapien.Pose(p=[0, 0.05, 0.35],
                                q=transforms3d.euler.euler2quat(0, 0, 0))
)

right_pad = builder.create_link_builder(end_effector)
right_pad.set_name('right_pad')

```

```

right_pad.add_box_collision(half_size=[0.2, 0.05, 0.4])
right_pad.add_box_visual(half_size=[0.2, 0.05, 0.4], color=[0.8, 0.6, 0.4])
right_pad.set_joint_name('right_pad_joint')
right_pad.set_joint_properties(
    "fixed", limits=[],
    pose_in_parent=sapient.Pose(p=[0, 0.5, 0],
                                q=transforms3d.euler.euler2quat(0, 0, 0)),
    pose_in_child=sapient.Pose(p=[0, -0.05, 0.35],
                                q=transforms3d.euler.euler2quat(0, 0, 0))
)

robot = builder.build(fix_root_link=True)
robot.set_name('robot')
robot.set_qpos([0, 0, 0]) # qpos indicates joint positions
return robot

def main():
    engine = sapient.Engine()
    renderer = sapient.VulkanRenderer()
    engine.set_renderer(renderer)

    scene_config = sapient.SceneConfig()
    scene_config.gravity = np.array([0.0, 0.0, 0.0]) # ignore the gravity
    scene = engine.create_scene(scene_config)
    scene.set_timestep(1 / 100.0)

    scene.add_ground(altitude=-1.5) # let base link frame align with world frame

    robot = create_robot(scene)
    print('The DoF of the robot is:', robot.dof)

    # HW1: You need to fill the blanks in `create_robot` and then run the
    following test cases
    # You can check your implementation with the first 4 test cases.
    test_FK(robot, [0, 0, 0])
    test_FK(robot, [0, 0, 0.7])
    test_FK(robot, [0, np.deg2rad(45), 0.7])
    test_FK(robot, [np.deg2rad(-30), np.deg2rad(45), 0.7])
    # Please report your output for this test case in your PDF submission
    print("Hidden Test Case:")
    test_FK(robot, [np.deg2rad(95), np.deg2rad(-36), -0.3])

    scene.set_ambient_light([0.5, 0.5, 0.5])
    print("Verify question 2")
    test_FK(robot, [0.1*np.deg2rad(180), 0.2*np.deg2rad(180), -0.3])

    print("Verify question 3:")

```

```

test_FK(robot, [-np.deg2rad(180)/6, np.deg2rad(180)/6, 0.5])
test_FK(robot, [0, 0, -1])
test_FK(robot, [0, 0, 1])

# uncomment the following codes and run with `xvfb-run python3 robot.py` if
↪you do not no DISPLAY;
# the result will be saved in `output.png`;
# you may need tune the camera position for better visualization.
from PIL import Image
near, far = 0.1, 100
width, height = 640, 480
camera_mount_actor = scene.create_actor_builder().build_kinematic()
camera = scene.add_mounted_camera(
    name="camera",
    actor=camera_mount_actor,
    pose=sapient.Pose(), # relative to the mounted actor
    width=width,
    height=height,
    fovx=np.deg2rad(35),
    fovy=np.deg2rad(35),
    near=near,
    far=far,
)

print('Intrinsic matrix\n', camera.get_camera_matrix())

# Compute the camera pose by specifying forward(x), left(y) and up(z)
cam_pos = np.array([-2, -4, 3])
forward = -cam_pos / np.linalg.norm(cam_pos)
left = np.cross([0, 0, 1], forward)
left = left / np.linalg.norm(left)
up = np.cross(forward, left)
mat44 = np.eye(4)
mat44[:3, :3] = np.stack([forward, left, up], axis=1)
mat44[:3, 3] = cam_pos
camera_mount_actor.set_pose(sapient.Pose.from_transformation_matrix(mat44))

scene.step()
scene.update_render()

camera.take_picture()
#viewer.render()
#continue
rgba = camera.get_float_texture('Color') # [H, W, 4]
# An alias is also provided
# rgba = camera.get_color_rgba() # [H, W, 4]
rgba_img = (rgba * 255).clip(0, 255).astype("uint8")

```

```

    rgba_pil = Image.fromarray(rgba_img)
    rgba_pil.save('output.png')
    return
main()

```

The DoF of the robot is: 3

FK Test with qpos: [0, 0, 0] .

Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .

Link link1's pose is: Pose([0, 0.8, 1.7], [1, 0, 0, 0]) .

Link link2's pose is: Pose([0, 2.4, 2.1], [1, 0, 0, 0]) .

Link link3's pose is: Pose([0, 3.6, 2.1], [1, 0, 0, 0]) .

Link end_effector's pose is: Pose([0, 3.6, 0.85], [1, 0, 0, 0]) .

Link left_pad's pose is: Pose([0, 3.05, 0.5], [1, 0, 0, 0]) .

Link right_pad's pose is: Pose([0, 4.15, 0.5], [1, 0, 0, 0]) .

FK Test with qpos: [0, 0, 0.7] .

Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .

Link link1's pose is: Pose([0, 0.8, 1.7], [1, 0, 0, 0]) .

Link link2's pose is: Pose([0, 2.4, 2.1], [1, 0, 0, 0]) .

Link link3's pose is: Pose([8.34465e-08, 3.6, 1.4], [1, 0, 0, 0]) .

Link end_effector's pose is: Pose([8.34465e-08, 3.6, 0.15], [1, 0, 0, 0]) .

Link left_pad's pose is: Pose([8.34465e-08, 3.05, -0.2], [1, 0, 0, 0]) .

Link right_pad's pose is: Pose([8.34465e-08, 4.15, -0.2], [1, 0, 0, 0]) .

FK Test with qpos: [0, 0.7853981633974483, 0.7] .

Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .

Link link1's pose is: Pose([0, 0.8, 1.7], [1, 0, 0, 0]) .

Link link2's pose is: Pose([0.565685, 2.16569, 2.1], [0.92388, 4.56194e-08, 0, -0.382683]) .

Link link3's pose is: Pose([1.41421, 3.01421, 1.4], [0.92388, 4.56194e-08, 0, -0.382683]) .

Link end_effector's pose is: Pose([1.41421, 3.01421, 0.15], [0.92388, 4.56194e-08, 0, -0.382683]) .

Link left_pad's pose is: Pose([1.0253, 2.6253, -0.2], [0.92388, 4.56194e-08, 0, -0.382683]) .

Link right_pad's pose is: Pose([1.80312, 3.40312, -0.2], [0.92388, 4.56194e-08, 0, -0.382683]) .

FK Test with qpos: [-0.5235987755982988, 0.7853981633974483, 0.7] .

Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .

Link link1's pose is: Pose([-0.4, 0.69282, 1.7], [0.965926, -3.08536e-08, 0, 0.258819]) .

Link link2's pose is: Pose([-0.592945, 2.15838, 2.1], [0.991445, 1.55599e-08, 0, -0.130526]) .

Link link3's pose is: Pose([-0.282362, 3.31749, 1.4], [0.991445, 1.55599e-08, 0, -0.130526]) .

Link end_effector's pose is: Pose([-0.282362, 3.31749, 0.15], [0.991445, 1.55599e-08, 0, -0.130526]) .

Link left_pad's pose is: Pose([-0.424712, 2.78623, -0.2], [0.991445, 1.55599e-08, 0, -0.130526]) .
Link right_pad's pose is: Pose([-0.140011, 3.84875, -0.2], [0.991445, 1.55599e-08, 0, -0.130526]) .

Hidden Test Case:

FK Test with qpos: [1.6580627893946132, -0.6283185307179586, -0.3] .
Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .
Link link1's pose is: Pose([0.796956, -0.0697246, 1.7], [0.67559, 8.78903e-08, 0, -0.737277]) .
Link link2's pose is: Pose([2.27965, 0.272581, 2.1], [0.870356, 5.87015e-08, 0, -0.492424]) .
Link link3's pose is: Pose([3.30825, 0.890627, 2.4], [0.870356, 5.87015e-08, 0, -0.492424]) .
Link end_effector's pose is: Pose([3.30825, 0.890627, 1.15], [0.870356, 5.87015e-08, 0, -0.492424]) .
Link left_pad's pose is: Pose([2.8368, 0.607356, 0.8], [0.870356, 5.87015e-08, 0, -0.492424]) .
Link right_pad's pose is: Pose([3.77969, 1.1739, 0.8], [0.870356, 5.87015e-08, 0, -0.492424]) .

Verify question 2

FK Test with qpos: [0.3141592653589793, 0.6283185307179586, -0.3] .
Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .
Link link1's pose is: Pose([0.247214, 0.760845, 1.7], [0.987688, 1.86484e-08, 0, -0.156434]) .
Link link2's pose is: Pose([1.14164, 1.99192, 2.1], [0.891007, 5.41199e-08, 0, -0.453991]) .
Link link3's pose is: Pose([2.11246, 2.69726, 2.4], [0.891007, 5.41199e-08, 0, -0.453991]) .
Link end_effector's pose is: Pose([2.11246, 2.69726, 1.15], [0.891007, 5.41199e-08, 0, -0.453991]) .
Link left_pad's pose is: Pose([1.6675, 2.37398, 0.8], [0.891007, 5.41199e-08, 0, -0.453991]) .
Link right_pad's pose is: Pose([2.55742, 3.02054, 0.800001], [0.891007, 5.41199e-08, 0, -0.453991]) .

Verify question 3:

FK Test with qpos: [-0.5235987755982988, 0.5235987755982988, 0.5] .
Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .
Link link1's pose is: Pose([-0.4, 0.69282, 1.7], [0.965926, -3.08536e-08, 0, 0.258819]) .
Link link2's pose is: Pose([-0.8, 2.18564, 2.1], [1, 0, 0, 0]) .
Link link3's pose is: Pose([-0.8, 3.38564, 1.6], [1, 0, 0, 0]) .
Link end_effector's pose is: Pose([-0.8, 3.38564, 0.350001], [1, 0, 0, 0]) .
Link left_pad's pose is: Pose([-0.8, 2.83564, 6.25849e-07], [1, 0, 0, 0]) .
Link right_pad's pose is: Pose([-0.8, 3.93564, 6.25849e-07], [1, 0, 0, 0]) .

```

FK Test with qpos: [0, 0, -1] .
Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .
Link link1's pose is: Pose([0, 0.8, 1.7], [1, 0, 0, 0]) .
Link link2's pose is: Pose([0, 2.4, 2.1], [1, 0, 0, 0]) .
Link link3's pose is: Pose([-1.19209e-07, 3.6, 3.1], [1, 0, 0, 0]) .
Link end_effector's pose is: Pose([-1.19209e-07, 3.6, 1.85], [1, 0, 0, 0]) .
Link left_pad's pose is: Pose([-1.19209e-07, 3.05, 1.5], [1, 0, 0, 0]) .
Link right_pad's pose is: Pose([-1.19209e-07, 4.15, 1.5], [1, 0, 0, 0]) .
-----

```

```

FK Test with qpos: [0, 0, 1] .
Link base's pose is: Pose([0, 0, 0], [1, 0, 0, 0]) .
Link link1's pose is: Pose([0, 0.8, 1.7], [1, 0, 0, 0]) .
Link link2's pose is: Pose([0, 2.4, 2.1], [1, 0, 0, 0]) .
Link link3's pose is: Pose([1.19209e-07, 3.6, 1.1], [1, 0, 0, 0]) .
Link end_effector's pose is: Pose([1.19209e-07, 3.6, -0.15], [1, 0, 0, 0]) .
Link left_pad's pose is: Pose([1.19209e-07, 3.05, -0.5], [1, 0, 0, 0]) .
Link right_pad's pose is: Pose([1.19209e-07, 4.15, -0.5], [1, 0, 0, 0]) .
-----

```

```

Intrinsic matrix
[[761.18274  0.      320.      0.      ]
 [ 0.      761.18274 240.      0.      ]
 [ 0.      0.      1.      0.      ]
 [ 0.      0.      0.      1.      ]]

```