```
import pybullet as p
import time
import pybullet_data
import numpy as np
import matplotlib.pyplot as plt

class Biped2D(object):
    def __init__(self):
        self.physicsClient = p.connect(p.GUI)  # or p.DIRECT for non-graphical
version
        p.resetDebugVisualizerCamera(cameraDistance=2, cameraYaw=0,
                                     cameraPitch=0, cameraTargetPosition=[0, 0,
1.2])
        p.setAdditionalSearchPath(pybullet_data.getDataPath())  # optionally
        self.ground = p.loadURDF("plane.urdf")
        self.robot = p.loadMJCF("sustech_biped2d.xml", flags =
p.MJCF_COLORS_FROM_FILE)[0]
        self.joints = self.get_joints()
        self.n_j = len(self.joints)
        self.simu_f = 500 # Simulation frequency, Hz
        self.motion_f = 5 # Controlled motion frequency, Hz
        self.q_vec = np.zeros(self.n_j)
        self.dq_vec = np.zeros(self.n_j)
        self.q_mat = np.zeros((self.simu_f * 3, self.n_j))
        self.q_d_mat = np.zeros((self.simu_f * 3, self.n_j))
        self.init_plot()


    def run(self):
        for i in range(int(5e3)):
            t = i / self.simu_f
            torque_array = self.controller(t)
            self.q_vec, self.dq_vec = self.step(torque_array)
            if 0 == i % 20:
                self.update_plot()
            time.sleep(1/self.simu_f)
        p.disconnect()

    def step(self, torque_array):
        self.set_motor_torque_array(torque_array)
        p.stepSimulation()
        self.q_mat[:-1] = self.q_mat[1:]
        self.q_mat[-1] = self.q_vec
        return self.get_joint_states()
```

```
    def get_joints(self):
        all_joints = []
        for j in range(p.getNumJoints(self.robot)):
            # Disable motor in order to use direct torque control.
            info = p.getJointInfo(self.robot, j)
            joint_type = info[2]
            if (joint_type == p.JOINT_PRISMATIC or joint_type ==
p.JOINT_REVOLUTE):
                all_joints.append(j)
                p.setJointMotorControl2(self.robot, j,
                                        controlMode=p.VELOCITY_CONTROL, force=0)
        joints = all_joints[0:]
        return joints

    def get_joint_states(self):
        '''
        :return: q_vec: joint angle, dq_vec: joint angular velocity
        '''
        q_vec = np.zeros(self.n_j)
        dq_vec = np.zeros(self.n_j)
        for j in range(self.n_j):
            q_vec[j], dq_vec[j], _, _  = p.getJointState(self.robot,
self.joints[j])
        return q_vec, dq_vec

    def set_motor_torque_array(self, torque_array = None):
        '''
        :param torque_array: the torque of [lthigh, lshin, lfoot, rthigh,
rshin, rfoot]
        '''
        if torque_array is None:
            torque_array = np.zeros(self.n_j)
        for j in range(len(self.joints)):
            p.setJointMotorControl2(self.robot, self.joints[j],
p.TORQUE_CONTROL, force=torque_array[j])

    def controller(self, t, type='joint'):
        if 'joint' == type:
            return self.joint_controller(t)

    def joint_controller(self, t):
        a = 0.3
        b = 0.15
        l = 0.6
```

```
        l1 = 0.5
        l2 = 0.5
        theta = (t*self.motion_f-int(t))*(2*np.pi)
        y = a*np.cos(theta)
        x = b*np.sin(theta)+l
        cos_theta_2 = (x*x+y*y-l1*l1-l2*l2)/(2*l1*l2)
        theta_2 = np.arctan2(np.sqrt(1-cos_theta_2**2), cos_theta_2)
        theta_1 = np.arctan2(y,x) +
np.arctan2(l2*np.sin(theta_2),l1+l2*np.cos(theta_2))
        theta2 = ((t-0.1)*self.motion_f-int(t-0.1))*(2*np.pi)
        y2 = a*np.cos(theta2)
        x2 = b*np.sin(theta2)+l
        cos_theta_4 = (x2*x2+y2*y2-l1*l1-l2*l2)/(2*l1*l2)
        theta_4 = np.arctan2(np.sqrt(1-cos_theta_4**2), cos_theta_4)
        theta_3 = np.arctan2(y2,x2) +
np.arctan2(l2*np.sin(theta_4),l1+l2*np.cos(theta_4))
        q_d_vec = np.array([theta_1, -theta_2, np.pi/2-(np.pi/2-
theta_2+theta_1), theta_3, -theta_4, np.pi/2-(np.pi/2-theta_4+theta_3)])
        J = np.array([[-l1*np.sin(theta_1)+l2*np.sin(theta_2-theta_1), -
l2*np.sin(theta_2-theta_1)],
                      [l1*np.cos(theta_1)+l2*np.cos(theta_2-theta_1), -
l2*np.cos(theta_2-theta_1)]])
        dx = -b*np.sin(theta)
        dy = a*np.sin(theta)
        dtheta1, dtheta2 = np.linalg.inv(J) @ np.array([dx, dy]).T
        J2 = np.array([[-l1*np.sin(theta_3)+l2*np.sin(theta_4-theta_3), -
l2*np.sin(theta_4-theta_3)],
                      [l1*np.cos(theta_3)+l2*np.cos(theta_4-theta_3), -
l2*np.cos(theta_4-theta_3)]])
        dx2 = -b*np.sin(theta2)
        dy2 = a*np.sin(theta2)
        dtheta3, dtheta4 = np.linalg.inv(J2) @ np.array([dx2, dy2]).T
        dq_d_vec = np.array([dtheta1, dtheta2, -(-dtheta2+dtheta1), dtheta3,
dtheta4, -(-dtheta4+dtheta3)])
        self.q_d_mat[:-1] = self.q_d_mat[1:]
        self.q_d_mat[-1] = q_d_vec
        return self.joint_impedance_controller(self.q_vec, self.dq_vec,
q_d_vec, dq_d_vec)

    def joint_impedance_controller(self, q_vec, dq_vec, q_d_vec, dq_d_vec):
        k = np.array([10000,10000,2000])
        k = np.r_[k, k]
        b = np.array([5,5,2])
        b = np.r_[b, b]
```

```python
        return k*(q_d_vec-q_vec) + b*(dq_d_vec - dq_vec)

    def init_plot(self):
        self.fig = plt.figure(figsize=(5, 9))
        joint_names = ['lthigh', 'lshin', 'lfoot',
                       'rthigh', 'rshin', 'rfoot', ]
        self.q_d_lines = []
        self.q_lines = []
        y_label = ['thigh', 'shin', 'foot', 'thigh', 'shin', 'foot'] # hop
        for i, yl in enumerate(y_label):
            plt.subplot(6, 1, i+1)
            q_d_line, = plt.plot(self.q_d_mat[:, i], '-')
            q_line, = plt.plot(self.q_mat[:, i], '--')
            self.q_d_lines.append(q_d_line)
            self.q_lines.append(q_line)
            plt.ylabel('q_{} (rad)'.format(joint_names[i]))
            if yl == 'thigh':
                plt.ylim([-0.5, 2.5])
            elif yl == 'shin':
                plt.ylim([-3.5,-0.5])
            else:
                plt.ylim([-1.5, 1.5])
        plt.xlabel('Simulation steps')
        self.fig.legend(['q_d', 'q'], loc='lower center', ncol=2,
bbox_to_anchor=(0.49, 0.97), frameon=False)
        self.fig.tight_layout()
        plt.draw()

    def update_plot(self):
        for i in range(6):
            self.q_d_lines[i].set_ydata(self.q_d_mat[:, i])
            self.q_lines[i].set_ydata(self.q_mat[:, i])
        plt.draw()
        plt.pause(0.001)


if __name__ == '__main__':
    robot = Biped2D()
    robot.run()
```