HW #3

Pybullet simulation 를 통한 Inverse Dynamics 검증

작성자: 김지수

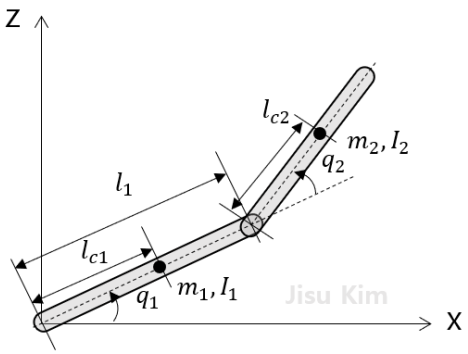
학번: 202133015

제출일: 2022 년 11 월 16 일

각 조인트에 인가되는 토크는 아래 식과 같다.

$$\boldsymbol{\tau} = M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + g(\boldsymbol{q})$$

과제에서 사용한 Two-link Manipulator는 아래 그림과 같다.



위 로봇의 자코비안 행렬은 아래 식과 같다.

$$J_{vc1} = \begin{bmatrix} -l_{c1}\sin(q_1) & 0 \\ l_{c1}\cos(q_1) & 0 \\ 0 & 0 \end{bmatrix} \quad J_{vc2} = \begin{bmatrix} -l_1\sin(q_1) - l_{c2}\sin(q_1+q_2) & -l_{c2}\sin(q_1+q_2) \\ l_1\cos(q_1) + l_{c2}\cos(q_1+q_2) & l_{c2}\cos(q_1+q_2) \\ 0 & 0 \end{bmatrix}$$

위 식을 이용하여 Inertia Matrix $M(\boldsymbol{q})$를 구하면 아래 식과 같다.

$$M(\boldsymbol{q}) = m_1 J_{vc1}^T J_{vc1} + m_2 J_{vc1}^T J_{vc1} + \begin{bmatrix} I_1 + I_2 & I_2 \\ I_2 & I_2 \end{bmatrix}$$

위 식을 파이썬 코드로 나타내면 아래와 같다. 질량 및 링크 길이, 관성텐서 값은 UR5의 urdf파일을 참고하여 작성하였다. 또한 Two-link로 만들기 위해 wrist 부분의 질량은 모두 0으로 설정하였다.

```
mass = np.array([8.393, 2.275])
m1 = 8.393
m2 = 2.275

p_com = np.array([[0.0, 0.0, 0.28],[0.0, 0.0, 0.25]])
lc1 = 0.28
lc2 = 0.25
l1 = 0.425

Is = []
Is.append([[0.22689067591, 0.0,           0.00],
           [0.0,           0.22689067591, 0.00],
           [0.0,           0.0,           0.0151074]])

Is.append([[0.049443313556, 0.0,            0.0],
           [0.0,            0.049443313556, 0.0],
           [0.0,            0.0,            0.004095]])
```

```
def calc_mass_mat(q):
    c2 = np.cos(q[1])

    d11 = m1*lc1*lc1 + m2*(l1*l1 + lc2*lc2 + 2.0*l1*lc2*c2) + Is[0][1][1]+Is[1][1][1]
    d12 = d21 = m2*(lc2*lc2 + l1*lc2*c2) + Is[1][1][1]
    d22 = m2*lc2*lc2 + Is[1][1][1]

    M = np.array([[d11,d12],[d21,d22]])

    return M
```

Coriolis Matrix는 아래 식과 같다.

$$C(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} h\dot{q}_2 & h(\dot{q}_1+\dot{q}_2) \\ -h\dot{q}_1 & 0 \end{bmatrix}, \qquad h = -m_2 l_1 l_{c2}\sin(q_2),$$

이를 파이썬 코드로 나타내면 아래와 같다.

```
def calc_coriolis_force(q, q_dot):
    h = -m2*l1*lc2*np.sin(q[1])
    C = np.array([[h*q_dot[1], h*(q_dot[0]+q_dot[1])],[-h*q_dot[0], 0.0]])
    cori_toque = np.array([C[0][0]*q_dot[0] + C[0][1]*q_dot[1], C[1][0]*q_dot[0] + C[1][1]*q_dot[1]])
    return cori_toque
```

Gravity Matrix는 아래 식과 같다.

$$g(\boldsymbol{q}) = \begin{bmatrix} (m_1 l_{c1} + m_2 l_1)\cos(q_1) + m_2 l_{c2}\cos(q_1+q_2) \\ m_2 l_{c2}\cos(q_1+q_2) \end{bmatrix} * g$$

이를 파이썬 코드로 나타내면 아래와 같다.

```
def calc_gravity_force(q):
    g1 = (m1*lc1 + m2*l1)*np.cos(q[0]) + m2*lc2*np.cos(q[0]+q[1])
    g2 = m2*lc2*np.cos(q[0]+q[1])
    G = -9.81*np.array([g1,g2])
    return G
```

마지막으로 pybullet 시뮬레이션에서 제공하는 calculateMassMatrix 함수와 calculateInverseDynamics 함수를 사용하여 직접 계산한 결과와 비교한다. 파이썬 코드는 아래와 같이 작성하였다.

```
joint_states = p.getJointStates(robot_id, range(dof))
q = [state[0] for state in joint_states]
q_dot = [state[1] for state in joint_states]
acc_des = [0. for _ in q]
vel_des = [0. for _ in q_dot]

M_sim = np.array(p.calculateMassMatrix(robot_id, q[:2]))
M_cal = calc_mass_mat(q)
```

Inertia Matrix $M(q)$ 를 비교한 결과는 아래와 같다.

```
M_sim
[[1.970 0.433]
 [0.433 0.192]]
M_cal
[[1.970 0.433]
 [0.433 0.192]]
Diff_M
[[-0.000 -0.000]
```

동일한 값을 가지며 시뮬레이션으로 계산된 값과 직접 계산한 값의 오차가 0으로 나타나는 것을 확인할 수 있다.

Coriolis Matrix의 검증을 위하여 calculateInverseDynamics 함수를 사용하였다. 하지만 pybullet에서 제공하는 calculateInverseDynamics 함수는 각 조인트에 인가되는 토크를 출력해주는 함수이다. 따라서 Coriolis Matrix에 관절의 속도벡터를 곱하여 Coriolis torque를 비교하는 방식으로 계산하였다. 이를 위하여 먼저 Gravity Matrix를 구해준다. calculateInverseDynamics 함수를 사용하여 Gravity Matrix를 구하기 위해 함수의 인자 값으로 가속도와 속도가 0인 벡터를 입력한다. 가속도와 속도 벡터가 0이면 아래와 같이 중력텀만 남게된다.

0 Jisu Kim 0

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}_{tot}$$

먼저 중력에 의한 토크 검증을 위해 시뮬레이션과 값을 비교하여 오차가 0인 것을 확인할 수 있었다.

```
G_sim = np.array(p.calculateInverseDynamics(robot_id, q[:2], vel_des[:2], acc_des[:2]))
G_cal = calc_gravity_force(q)
```

```
G_sim
[-38.106 -5.567]
G_cal
[-38.106 -5.567]
Diff_G
[-0.000 -0.000]
```

Coriolis 텀 역시 중력텀과 마찬가지로 가속도가 0인 벡터를 함수의 인자로 넣어주고 앞서 구한 중력텀을 빼주면 Coriolis 효과에 의한 토크를 구할 수 있다.

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) - G\_cal = \boldsymbol{\tau}_{tot}$$

(0, 0, Jisu Kim)

파이썬 코드로 나타내면 아래와 같다.

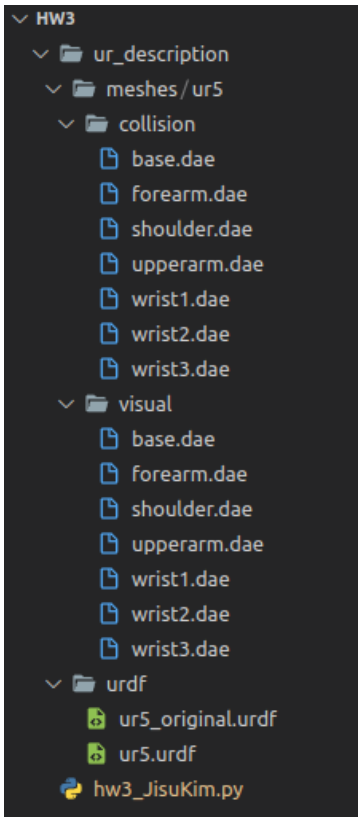
```
C_sim = np.array(p.calculateInverseDynamics(robot_id, q[:2], q_dot[:2], acc_des[:2])) - G_cal
C_cal = calc_coriolis_force(q,q_dot)
```

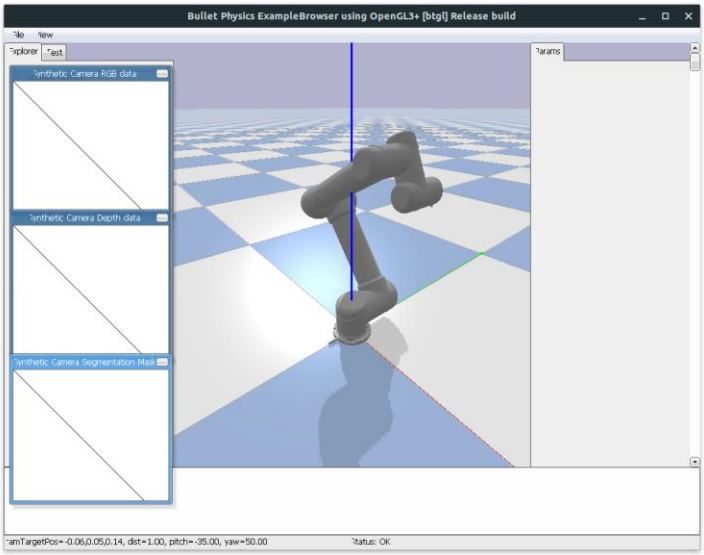
마찬가지로 시뮬레이션 값과 직접 계산한 값을 비교하면 오차가 0인 것을 확인할 수 있다.

```
C_sim
[-0.000 -0.000]
C_cal
[-0.000 0.000]
Diff_C
[-0.000 -0.000]
```

[Appendix]

pybullet에서 urdf를 불러오기 위하여 파일 구조를 아래와 같이 설정하였다.





파이선 메인 프로그램이 있는 곳에 ur_description 폴더를 생성하고 그 안에 meshes₩ur5 폴더를 생성하고, collision과 visual, 그리고 urdf 폴더를 생성한다.

collision과 visual 폴더 안에는 각 링크와 조인트의 3d 모델링 파일이 존재하며 urdf 폴더에는 UR5 로봇의 urdf 파일이 존재한다.

위 폴더구조를 생성하고 urdf 파일을 불러오면 위 그림과 같이 UR5 로봇이 불러와지는 것을 확인할 수 있다.