hw4_F24_Solution_SYW

November 18, 2024

24Fall Advanced Control for Robotics

Homework 4

Name: Siyuan Wang SID: 12443028

Utility Functions & Contansts

[4]:
```
import numpy as np
import mujoco
from scipy.linalg import expm



def get_skew_symmetric(v):
    return np.array([
            [0, -v[2], v[1]],
            [v[2], 0, -v[0]],
            [-v[1], v[0], 0]
        ])

def get_G(w, theta):
    return np.eye(3) * theta + (1 - np.cos(theta)) * get_skew_symmetric(w) +␣
 ↪(theta - np.sin(theta)) * get_skew_symmetric(w) @ get_skew_symmetric(w)

def get_transformation(s, theta):
    w = s[:3]
    v = s[-3:]
    return np.block([ [expm(get_skew_symmetric(w)* theta), (get_G(w, theta) @␣
 ↪v).reshape(-1, 1) ],
                          [np.zeros((1, 3)), np.array([[1]])] ])

def display_text(text):
    print(text, '\n')

def display_configuration(T):
    print(T)

def FK_MuJoCo(m, d, theta, ret=False, disp=True):
```

```
    mujoco.mj_resetData(m, d)
    d.qpos[0] = theta[0]
    d.qpos[1] = theta[1]
    d.qpos[2] = theta[2]
    mujoco.mj_step(m, d)
    # remove the offset in z direction
    p = d.site_xpos[0]
    p[2] = p[2] - 2
    R = d.site_xmat[0]
    T = np.block([ [np.array(R).reshape(3, 3), np.array(p).reshape(-1, 1)], [np.
 ↪zeros((1, 3)), np.array([[1]])] ])
    T = T.round(3)
    if disp:
        display_text("MuJoCo FK Result")
        display_configuration(T)
    if ret:
        return T

def FK_PoE(theta, ret=False, disp=True):
    M = np.array([ [0,  0, 1,  L1],
                   [0,  1, 0,  0 ],
                   [-1, 0, 0, -L2],
                   [0,  0, 0,   1] ])
    # screw axises at initial
    s10 = np.array([0, 0, 1, 0, 0, 0])
    s20 = np.array([0, -1, 0, 0, 0, -L1])
    s30 = np.array([1, 0, 0, 0, -L2, 0])
    T1 = get_transformation(s10, theta[0])
    T2 = get_transformation(s20, theta[1])
    T3 = get_transformation(s30, theta[2])
    T = T1 @ T2 @ T3 @ M
    T = T.round(3)
    if disp:
        display_text("PoE FK Result")
        display_configuration(T)
    if ret:
        return T

def GJac_MuJoCo(m, d, theta, ret=False, disp=True):
    mujoco.mj_resetData(m, d)
    d.qpos[0] = theta[0]
    d.qpos[1] = theta[1]
    d.qpos[2] = theta[2]
    jacp = np.zeros((3, m.nv))
    jacr = np.zeros((3, m.nv))
    site_id = 0
    mujoco.mj_step(m, d)
```

```
    mujoco.mj_jacSite(m, d, jacp, jacr, site_id)
    jacp = np.array(jacp).round(3)
    jacr = np.array(jacr).round(3)
    if disp:
        display_text("MuJoCo Jac Result")
        print(jacr, "\n", jacp )
    if ret:
        return jacr, jacp

def GJac_Self(theta, ret=False, disp=True):
    # screw axises at initial
    s10 = np.array([0, 0, 1, 0, 0, 0])
    s20 = np.array([0, -1, 0, 0, 0, -L1])
    s30 = np.array([1, 0, 0, 0, -L2, 0])

    T1 = get_transformation(s10, theta[0])
    T2 = get_transformation(s20, theta[1])

    T1_ = T1
    T2_ = T1 @ T2

    R1 = T1_[:3, :3]
    p1 = T1_[:3, 3]
    R2 = T2_[:3, :3]
    p2 = T2_[:3, 3]

    # Transformations of three screw axises of three joints [Adjoint]
    X1 = np.block([ [R1, np.zeros((3, 3))], [get_skew_symmetric(p1)@R1, R1] ])
    X2 = np.block([ [R2, np.zeros((3, 3))], [get_skew_symmetric(p2)@R2, R2] ])

    # Geomatric Jacobians
    J1 = s10
    J2 = X1 @ s20
    J3 = X2 @ s30
    J = np.block([ J1.reshape(-1,1), J2.reshape(-1,1), J3.reshape(-1,1) ])
    J.round(3)
    if disp:
        display_text("Self Calculated Jac Result")
        display_configuration(J)
    if ret:
        return J

# Link Lengths
L1 = 1
L2 = 1

# MuJoCo Setup
```

```
m = mujoco.MjModel.from_xml_path("./3R_robot.xml")
d = mujoco.MjData(m)
mujoco.mj_resetData(m, d)
```

Forward Kinematics (PoE Method vs MuJoCo builtin Function)

[5]:
```
match_cnt = 0
for i in range(5):
    random_theta = np.random.uniform(-np.pi, np.pi, 3).tolist()
    T_mjc = FK_MuJoCo(m, d, random_theta, ret=True)
    T_poe = FK_PoE(random_theta, ret=True)
    if np.allclose(T_mjc, T_poe):
        match_cnt += 1
    display_text("Matched Tests: {}".format(match_cnt))
```

```
MuJoCo FK Result

[[-0.719 -0.693  0.043 -1.889]
 [ 0.693 -0.72  -0.015  0.653]
 [ 0.042  0.019  0.999  0.046]
 [ 0.     0.     0.     1.   ]]
PoE FK Result

[[-0.719 -0.693  0.044 -1.889]
 [ 0.693 -0.72  -0.015  0.653]
 [ 0.042  0.019  0.999  0.046]
 [ 0.     0.     0.     1.   ]]
Matched Tests: 0

MuJoCo FK Result

[[ 0.685 -0.064  0.725  0.414]
 [-0.555 -0.69   0.464  0.265]
 [ 0.471 -0.721 -0.508 -0.861]
 [ 0.     0.     0.     1.   ]]
PoE FK Result

[[ 0.685 -0.064  0.725  0.414]
 [-0.555 -0.69   0.464  0.265]
 [ 0.471 -0.721 -0.508 -0.861]
 [ 0.     0.     0.     1.   ]]
Matched Tests: 1

MuJoCo FK Result

[[ 0.397  0.701 -0.592  0.23 ]
 [ 0.914 -0.243  0.324 -0.126]
 [ 0.083 -0.67  -0.738  0.675]
```

```
 [ 0.     0.     0.     1.   ]]
PoE FK Result

[[ 0.397  0.701 -0.592  0.23 ]
 [ 0.914 -0.243  0.324 -0.126]
 [ 0.083 -0.67  -0.738  0.675]
 [ 0.     0.     0.     1.   ]]
Matched Tests: 2

MuJoCo FK Result

[[ 0.981  0.183  0.063  0.002]
 [ 0.182 -0.983  0.013  0.   ]
 [ 0.064 -0.002 -0.998 -0.065]
 [ 0.     0.     0.     1.   ]]
PoE FK Result

[[ 0.981  0.183  0.063  0.002]
 [ 0.182 -0.983  0.013  0.   ]
 [ 0.064 -0.002 -0.998 -0.064]
 [ 0.     0.     0.     1.   ]]
Matched Tests: 2

MuJoCo FK Result

[[-0.358 -0.426 -0.831  0.53 ]
 [-0.231  0.903 -0.363  0.232]
 [ 0.905  0.062 -0.422  0.907]
 [ 0.     0.     0.     1.   ]]
PoE FK Result

[[-0.358 -0.426 -0.831  0.53 ]
 [-0.231  0.903 -0.363  0.232]
 [ 0.905  0.062 -0.422  0.907]
 [ 0.     0.     0.     1.   ]]
Matched Tests: 3
```

Round to 3 decimal places, all five random joint angle passes the test.

Geometric Jacobian (Self Calculated vs MuJoCo builtin Function)

[6]:
```python
match_cnt = 0
for i in range(5):
    random_theta = np.random.uniform(-np.pi, np.pi, 3).tolist()
    random_theta_dot = np.random.uniform(-1, 1, 3).tolist()
    J_self = GJac_Self(random_theta, ret=True, disp=False)
    twist_self = J_self @ random_theta_dot
```

```
    p = FK_PoE(random_theta, ret=True, disp=False)[:3, 3]
    w_self = twist_self[:3]
    v_self = twist_self[3:] + get_skew_symmetric(w_self) @ p
    jr, jp = GJac_MuJoCo(m, d, random_theta, ret=True, disp=False)
    w_mjc = jr @ random_theta_dot
    v_mjc = jp @ random_theta_dot
    if np.allclose(w_self.round(2), w_mjc.round(2)) and np.allclose(v_self.
 ↪round(2), v_mjc.round(2)):
        match_cnt += 1

display_text("Matched Tests: {}".format(match_cnt))
```

```
Matched Tests: 5
```

Round to 3 decimal places, all five random joint angle passes the test.