Forward Kinematics Calculations

The forward kinematics are calculated in the function *ur5FwdKin()*.

The first step is determining the initial status of ur5. The initial position is shown in the following figure.

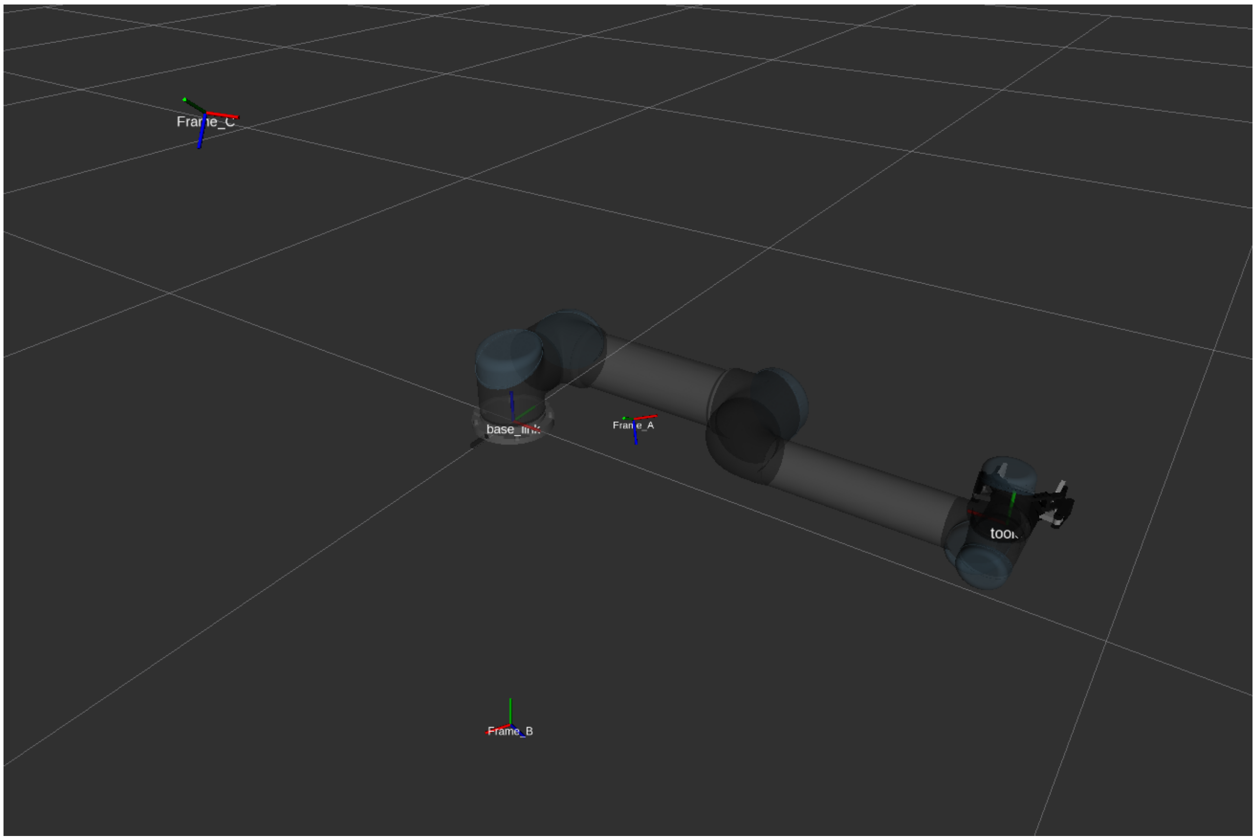

Figure 1: The initial status of UR5

The specifications of UR5 are given by Homework 6. They are hardcode in the function as follows:

```
L0 = 89.2/1000;
L = [425;392;109.3;94.75;82.5]/1000;
```

According to figure 1, we could determine the angular velocity w, the position of points on the rotation axis, and the initial transformation *gst0*. They are also hard-coded in the function.

```
w = [0,0,1;
     0,1,0;
```

```
        0,1,0;
        0,1,0;
        0,0,-1;
        0,1,0]';
    q = [0,0,0;
        0,0,L0;
        L(1),0,L0;
        L(1)+L(2),0,L0;
        L(1)+L(2),L(3),0;
        L(1)+L(2),0,L0-L(4)]';

    R = ROTX(-pi/2)*ROTZ(pi);
    gst = [R, [L(1)+L(2);L(3)+L(5);L0-L(4)];
        0,0,0,1];
```

gst_0 is calculated by mathematica.

```
In[62]:= gst0 = RPToHomogeneous[{{-1, 0, 0}, {0, 0, 1}, {0, 1, 0}}, {L₁ + L₂, L₃ + L₅, L₀ - L₄}]
Out[62]= {{-1, 0, 0, L₁ + L₂}, {0, 0, 1, L₃ + L₅}, {0, 1, 0, L₀ - L₄}, {0, 0, 0, 1}}
```

Figure 2: The expression of gst0

Then the basic twist could be calculated by the formula

$$\xi_i = \begin{bmatrix} q_i \times w_i \\ w_i \end{bmatrix}.$$

In the *ur5FwdKin()* function, this part is achieved by the function *RevoluteTwist()*.

Then for the twist $\xi = \begin{bmatrix} v \\ \omega \end{bmatrix}$, we have the transformation matrix for the joint

$$e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & \left(I - \mathrm{e}^{\hat{\omega}\theta}\right)(\omega \times v) \\ 0 & 1 \end{bmatrix}.$$

Multiply the transformation matrixes $e^{\hat{\xi}_i\theta_i}$ and the initial transformation g_{st0} up, we could calculate the forward kinematics matrix.

$$g_{st} = e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}e^{\hat{\xi}_4\theta_4}e^{\hat{\xi}_5\theta_5}e^{\hat{\xi}_6\theta_6}g_{st0}.$$

In the *ur5FwdKin()* function, this part is achieved by the function *TwistExp()* and a loop.

Expression in Mathemtica

The expression of gst_0

The expression of gst0 is shown in figure 2.

The expressions of ξ_i

```
In[85]:= w1 = {0, 0, 1};
         w2 = w3 = w4 = w6 = {0, 1, 0};
         w5 = {0, 0, -1};

         q1 = {0, 0, 0};
         q2 = {0, 0, L0};
         q3 = {L1, 0, L0};
         q4 = {L1 + L2, 0, L0};
         q5 = {L1 + L2, L3, 0};
         q6 = {L1 + L2, 0, L0 - L4};

         xi1 = RevoluteTwist[q1, w1]
         xi2 = RevoluteTwist[q2, w2]
         xi3 = RevoluteTwist[q3, w3]
         xi4 = RevoluteTwist[q4, w4]
         xi5 = RevoluteTwist[q5, w5]
         xi6 = RevoluteTwist[q6, w6]

Out[94]= {0, 0, 0, 0, 0, 1}

Out[95]= {-L0, 0, 0, 0, 1, 0}

Out[96]= {-L0, 0, L1, 0, 1, 0}

Out[97]= {-L0, 0, L1 + L2, 0, 1, 0}

Out[98]= {-L3, L1 + L2, 0, 0, 0, -1}

Out[99]= {-L0 + L4, 0, L1 + L2, 0, 1, 0}
```

Figure 3: The expression of ξ_i

The expressions of Jacobian

```
In[101]:= Jb = Simplify[BodyJacobian[{xi1, θ1}, {xi2, θ2}, {xi3, θ3}, {xi4, θ4}, {xi5, θ5}, {xi6, θ6}, gst0]]
              化简

Out[101]= {{1/4 × (4 Cos[θ2] Cos[θ6] Sin[θ5] L1 + 4 Cos[θ2 + θ3] Cos[θ6] Sin[θ5] L2 - 2 Cos[θ2 + θ3 + θ4 - θ6] L3 +
        Cos[θ2 + θ3 + θ4 - θ5 - θ6] L3 + Cos[θ2 + θ3 + θ4 + θ5 - θ6] L3 + 2 Cos[θ2 + θ3 + θ4 + θ6] L3 + Cos[θ2 + θ3 + θ4 - θ5 + θ6] L3 +
        Cos[θ2 + θ3 + θ4 + θ5 + θ6] L3 - Cos[θ2 + θ3 + θ4 - θ5 - θ6] L4 + Cos[θ2 + θ3 + θ4 + θ5 - θ6] L4 -
        Cos[θ2 + θ3 + θ4 - θ5 + θ6] L4 + Cos[θ2 + θ3 + θ4 + θ5 + θ6] L4 + 2 Cos[θ2 + θ3 + θ4 - θ6] L5 - Cos[θ2 + θ3 + θ4 - θ5 - θ6] L5 -
        Cos[θ2 + θ3 + θ4 + θ5 - θ6] L5 + 2 Cos[θ2 + θ3 + θ4 + θ6] L5 + Cos[θ2 + θ3 + θ4 - θ5 + θ6] L5 + Cos[θ2 + θ3 + θ4 + θ5 + θ6] L5),
   -((Cos[θ4] (Cos[θ5] Cos[θ6] Sin[θ3] + Cos[θ3] Sin[θ6]) + Sin[θ4] (Cos[θ3] Cos[θ5] Cos[θ6] - Sin[θ3] Sin[θ6])) L1) -
    (Cos[θ5] Cos[θ6] Sin[θ4] + Cos[θ4] Sin[θ6]) L2 + Cos[θ5] Cos[θ6] L4 - Sin[θ5] Sin[θ6] L5,
   -((Cos[θ5] Cos[θ6] Sin[θ4] + Cos[θ4] Sin[θ6]) L2) + Cos[θ5] Cos[θ6] L4 - Sin[θ5] Sin[θ6] L5,
   Cos[θ5] Cos[θ6] L4 - Sin[θ5] Sin[θ6] L5, -Cos[θ6] L5, 0},
  {1/4 × (-4 Cos[θ2] Sin[θ5] Sin[θ6] L1 - 4 Cos[θ2 + θ3] Sin[θ5] Sin[θ6] L2 - 2 Sin[θ2 + θ3 + θ4 - θ6] L3 +
        Sin[θ2 + θ3 + θ4 - θ5 - θ6] L3 + Sin[θ2 + θ3 + θ4 + θ5 - θ6] L3 - 2 Sin[θ2 + θ3 + θ4 + θ6] L3 - Sin[θ2 + θ3 + θ4 - θ5 + θ6] L3 -
        Sin[θ2 + θ3 + θ4 + θ5 + θ6] L3 - Sin[θ2 + θ3 + θ4 - θ5 - θ6] L4 + Sin[θ2 + θ3 + θ4 + θ5 - θ6] L4 +
        Sin[θ2 + θ3 + θ4 - θ5 + θ6] L4 - Sin[θ2 + θ3 + θ4 + θ5 + θ6] L4 + 2 Sin[θ2 + θ3 + θ4 - θ6] L5 - Sin[θ2 + θ3 + θ4 - θ5 - θ6] L5 -
        Sin[θ2 + θ3 + θ4 + θ5 - θ6] L5 - 2 Sin[θ2 + θ3 + θ4 + θ6] L5 - Sin[θ2 + θ3 + θ4 - θ5 + θ6] L5 - Sin[θ2 + θ3 + θ4 + θ5 + θ6] L5),
   (Sin[θ3] (Cos[θ6] Sin[θ4] + Cos[θ4] Cos[θ5] Sin[θ6]) + Cos[θ3] (-Cos[θ4] Cos[θ6] + Cos[θ5] Sin[θ4] Sin[θ6])) L1 +
    (-Cos[θ4] Cos[θ6] + Cos[θ5] Sin[θ4] Sin[θ6]) L2 - Cos[θ5] Sin[θ6] L4 - Cos[θ6] Sin[θ5] L5,
   (-Cos[θ4] Cos[θ6] + Cos[θ5] Sin[θ4] Sin[θ6]) L2 - Cos[θ5] Sin[θ6] L4 - Cos[θ6] Sin[θ5] L5,
   -Cos[θ5] Sin[θ6] L4 - Cos[θ6] Sin[θ5] L5, Sin[θ6] L5, 0},
  {1/2 × (2 Cos[θ2] Cos[θ5] L1 + 2 Cos[θ2 + θ3] Cos[θ5] L2 + Sin[θ2 + θ3 + θ4 - θ5] L3 -
        Sin[θ2 + θ3 + θ4 + θ5] L3 - Sin[θ2 + θ3 + θ4 - θ5] L4 - Sin[θ2 + θ3 + θ4 + θ5] L4),
   Sin[θ5] (Sin[θ3 + θ4] L1 + Sin[θ4] L2 - L4), Sin[θ5] (Sin[θ4] L2 - L4), -Sin[θ5] L4, 0, 0},
  {1/4 × (-2 Sin[θ2 + θ3 + θ4 - θ6] + Sin[θ2 + θ3 + θ4 - θ5 - θ6] + Sin[θ2 + θ3 + θ4 + θ5 - θ6] +
        2 Sin[θ2 + θ3 + θ4 + θ6] + Sin[θ2 + θ3 + θ4 - θ5 + θ6] + Sin[θ2 + θ3 + θ4 + θ5 + θ6]),
   Cos[θ6] Sin[θ5], Cos[θ6] Sin[θ5], Cos[θ6] Sin[θ5], -Sin[θ6], 0},
  {1/4 × (2 Cos[θ2 + θ3 + θ4 - θ6] - Cos[θ2 + θ3 + θ4 - θ5 - θ6] - Cos[θ2 + θ3 + θ4 + θ5 - θ6] +
        2 Cos[θ2 + θ3 + θ4 + θ6] + Cos[θ2 + θ3 + θ4 - θ5 + θ6] + Cos[θ2 + θ3 + θ4 + θ5 + θ6]),
   -Sin[θ5] Sin[θ6], -Sin[θ5] Sin[θ6], -Sin[θ5] Sin[θ6], -Cos[θ6], 0},
  {-Sin[θ2 + θ3 + θ4] Sin[θ5], Cos[θ5], Cos[θ5], Cos[θ5], 0, 1}}
```

Figure 4: The expression of Jacobian

Test for *ur5FwdKin()*

This part would generate a series theta with 6 angles. Input that into function *ur5FwdKin()* to get a transformation matrix. Then use the *tf_frame()* to transfer a frame named *fwdKinToolFrame* to the position where is the end-effector. After that, use the function *ur5.move_joints()* to move the manipulator to the correct position. If the frame *tool0* on the end-effector is overlapped with the *fwdKinToolFrame*, the function *ur5FwdKin()* could work as expected.

Test 1

```
theta1 = [pi/2;0;pi/3;1;2;3];
```

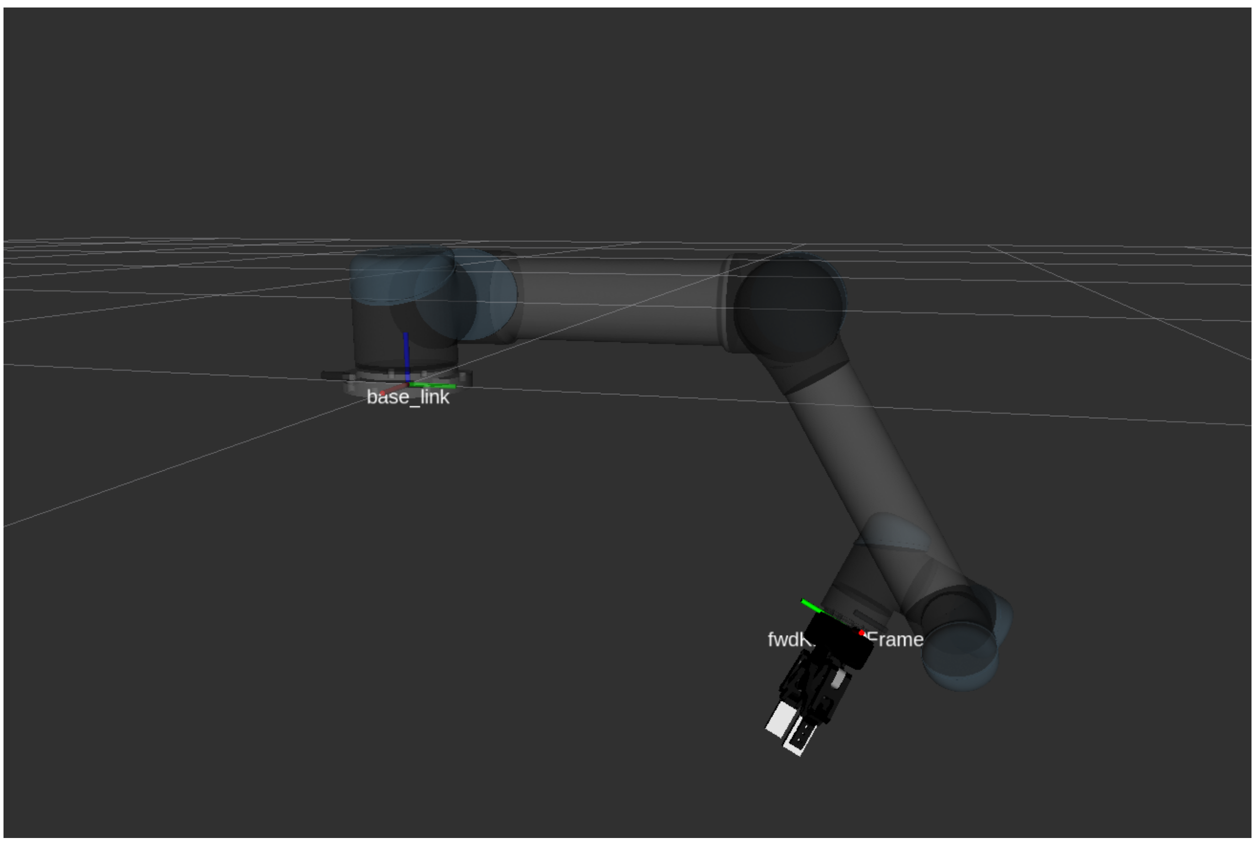

Figure 5: The final manipulator position for theta1

According to the figure 5, the frame *tool0* is overlapped with *fwdKinToolFrame.*

Test 2

`theta2 = [pi;pi/3;pi/4;0;2;1];`

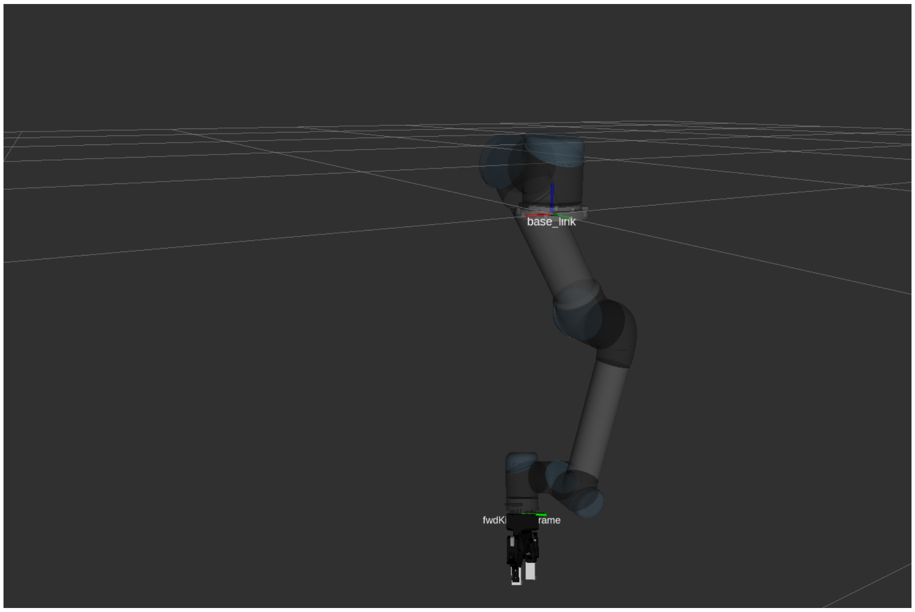

Figure 6: The final manipulator position for theta2

Test 3

`theta3 = [-pi;-pi/3;pi/4;2;1;-2];`

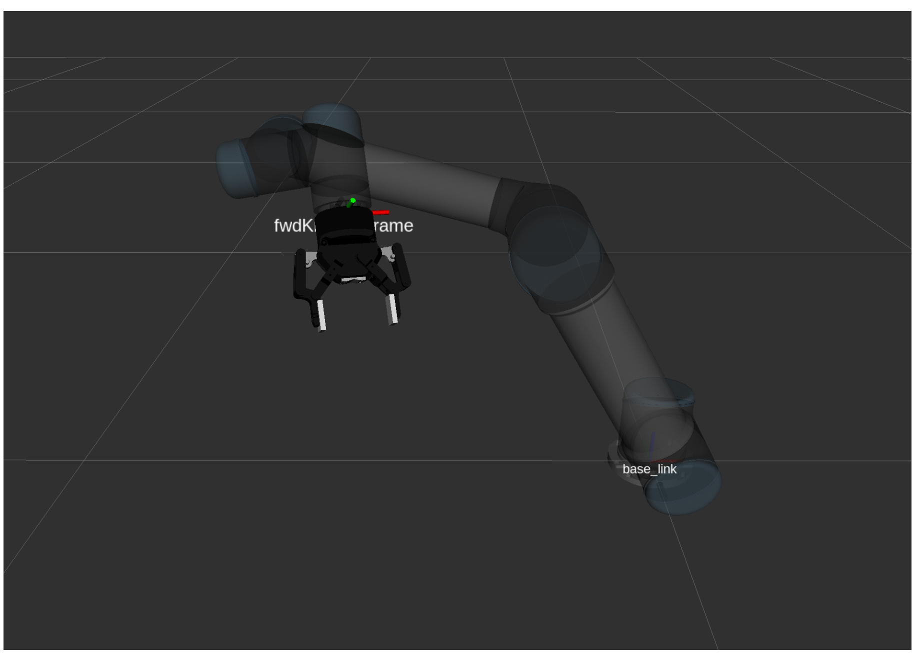

Figure 7: The final manipulator position for theta3

The transformation matrices between *base* frame to *tool0 and fwdKinToolFrame* are shown following. Furthermore, the norm errors between them are calculated.

```
My results in test 1:

g1 =

    0.9002    0.1283    0.4161   -0.0750
    0.3143   -0.8528   -0.4170    0.5024
    0.3014    0.5062   -0.8080   -0.2735
         0         0         0    1.0000

Current transformation in test 1:

gp1 =

    0.9002    0.1283    0.4161   -0.0750
    0.3143   -0.8528   -0.4170    0.5024
    0.3014    0.5062   -0.8080   -0.2735
         0         0         0    1.0000

The error in test 1 is 0.000000
My results in test 2:

g2 =

   -0.7546   -0.6125    0.2353   -0.0001
   -0.4913    0.7651    0.4161   -0.0750
   -0.4350    0.1984   -0.8783   -0.7054
         0         0         0    1.0000

Current transformation in test 2:

gp2 =

   -0.7546   -0.6125    0.2353   -0.0001
   -0.4913    0.7651    0.4161   -0.0750
   -0.4350    0.1984   -0.8783   -0.7054
         0         0         0    1.0000

The error in test 2 is 0.000000
My results in test 3:

g3 =

    0.9341    0.3285    0.1402   -0.4862
    0.3502   -0.7651   -0.5403   -0.1539
   -0.0702    0.5538   -0.8297    0.5061
         0         0         0    1.0000

Current transformation in test 3:

gp3 =

    0.9341    0.3285    0.1402   -0.4862
    0.3502   -0.7651   -0.5403   -0.1539
   -0.0702    0.5538   -0.8297    0.5061
         0         0         0    1.0000

The error in test 3 is 0.000000
```

Figure 8: Transformation
Matrices and error in three tests

Test for *ur5BodyJacobian()*

To test ur5BodyJacobian, a jacobian matrix would be calculated approximately and the result would be compared with the jacobian calculated by the function *ur5BodyJacobian()*. The jacobian matrix J_{approx} is calculated as follows.

The jacobian matrix J_{approx} is calculated line to line, which would be achieved in a loop. For each line of the matrix, we have

$$\boldsymbol{\xi}_i' \approx \left(g^{-1} \frac{\partial g}{\partial q_i} \right)^{\vee},$$

$$\text{where } \frac{\partial g}{\partial q_i} \approx \frac{1}{2\epsilon} \Big(g_{st}\left(\boldsymbol{q} + \epsilon \mathbf{e}_i\right) - g_{st}\left(\boldsymbol{q} - \epsilon \mathbf{e}_i\right) \Big).$$

e_i is the ith line of a six order identity matrix I_6.

Test

In this test, the joint vector q is defined as follows.

```
q = [pi/2;0;pi/3;1;2;3];
```

The error calculated by the function is displayed in figure 5.

```
The error for ε = 1e-1 is 0.003114905991
The error for ε = 1e-2 is 0.000031164483
The error for ε = 1e-3 is 0.000000311647
The error for ε = 1e-4 is 0.000000003116
The error for ε = 1e-5 is 0.000000000031
```

Figure 9: The error between J and Japprox under different ϵ

Manipulability function test

From HW6, there are 3 singular scenarios. The first two occur when theta3 or theta5 is zero, while the third one is too complicated to test.

For theta3 and theta5, we increase them from -pi/4 to pi/4 separately, the rest angles remain fixed. In addition, three ways were used to find the manipulability.

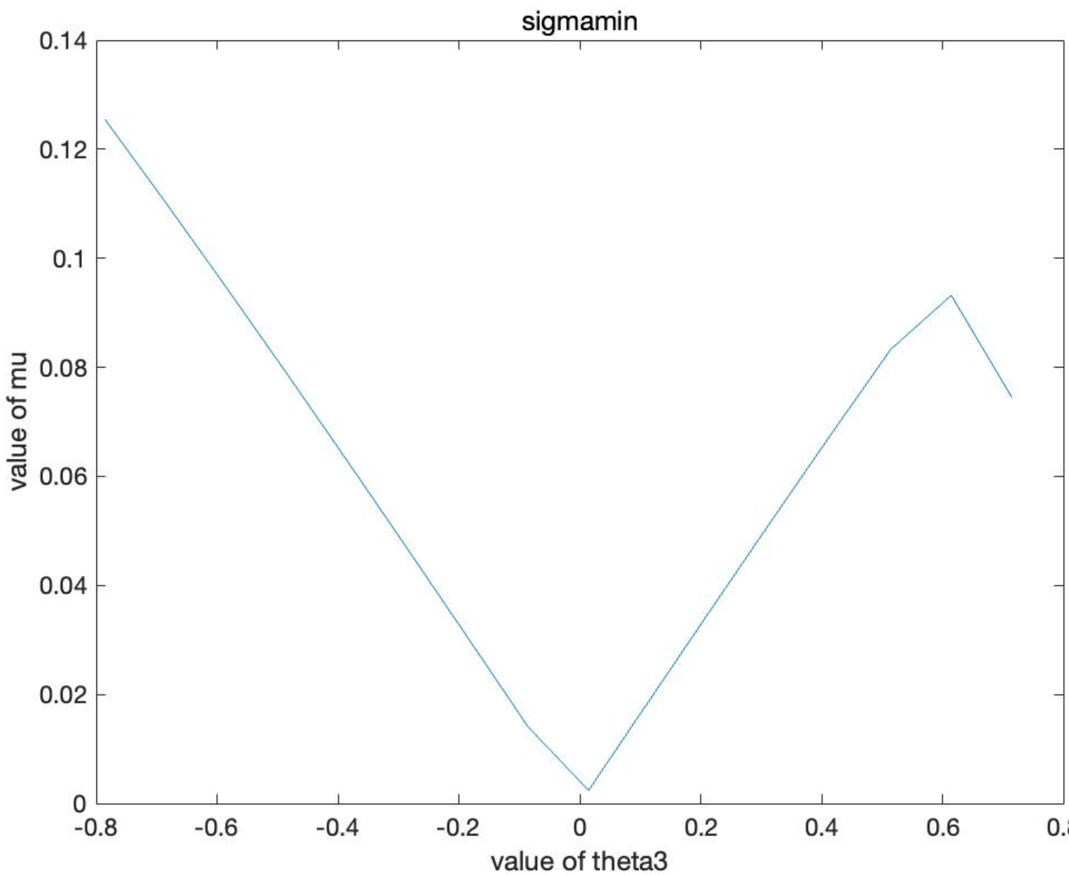

Figure 10: μ against θ_3 (sigmamin)

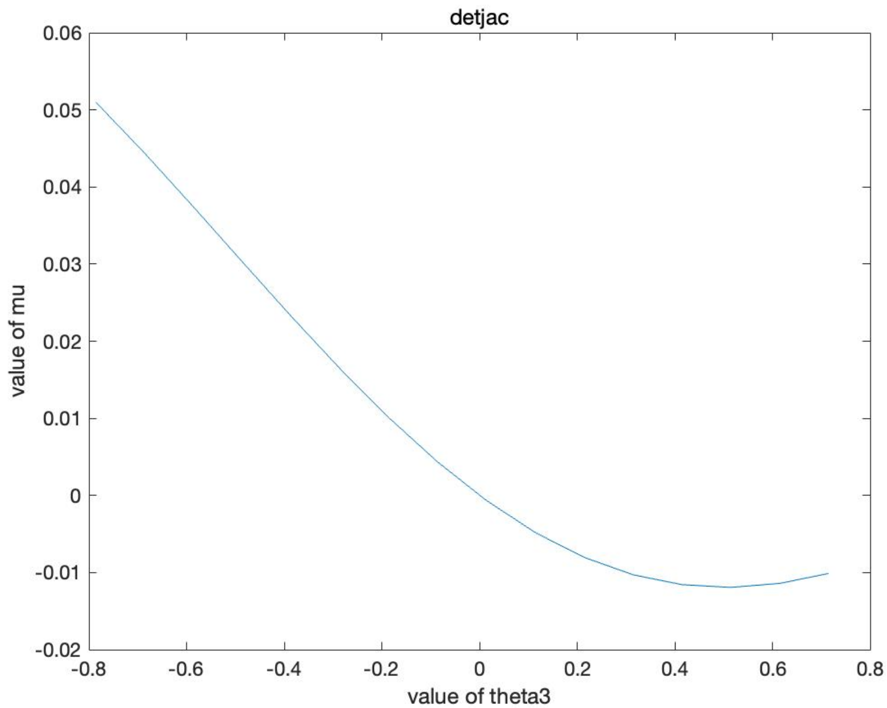

Figure 11: μ against θ_3 (detjac)

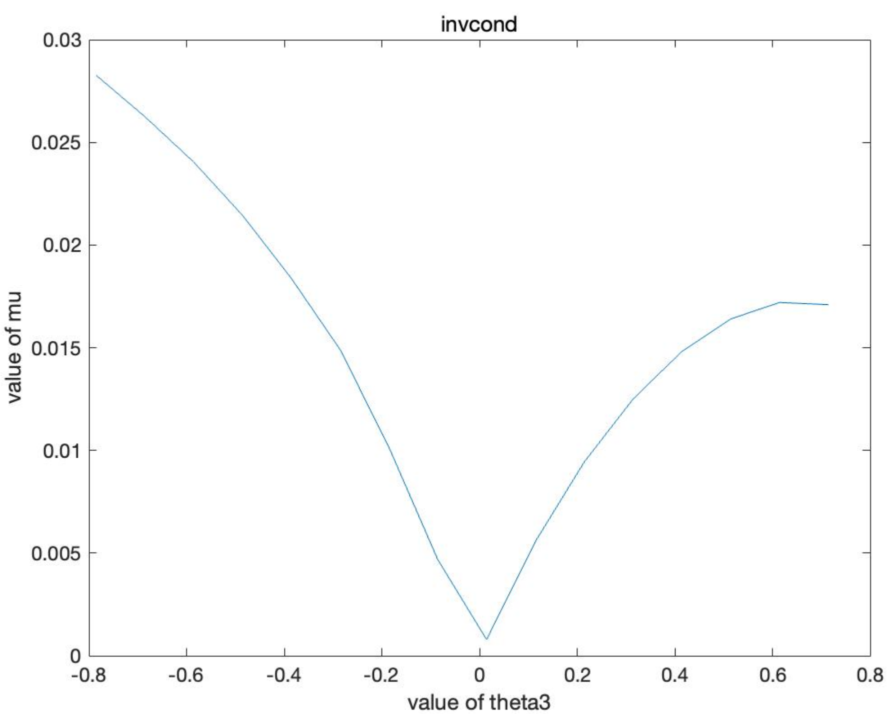

Figure 12: μ against θ_3 (invcond)

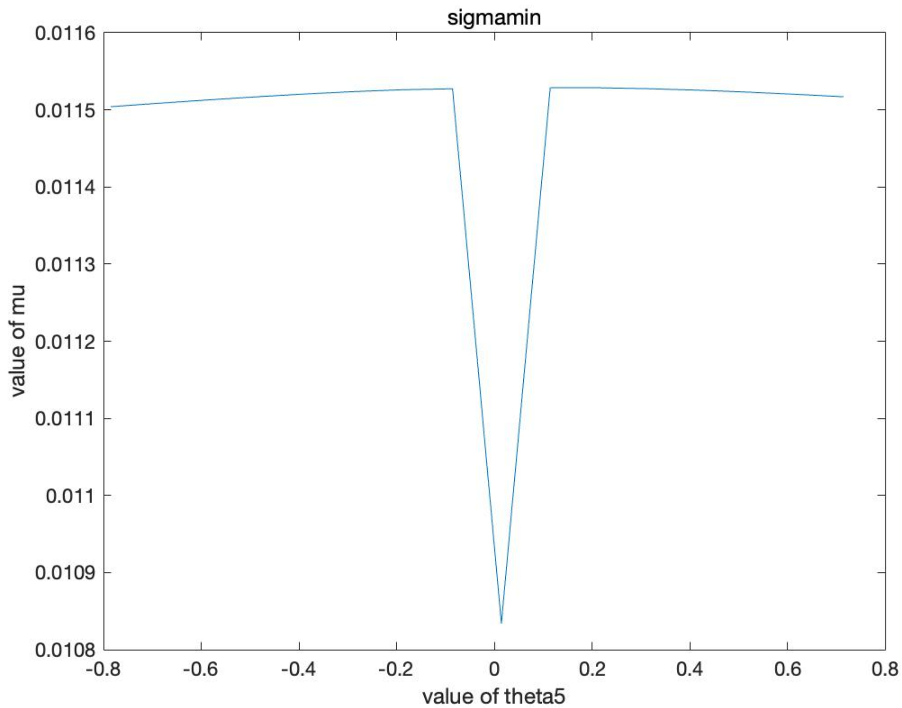

Figure 13: μ against θ_5 (sigmamin)

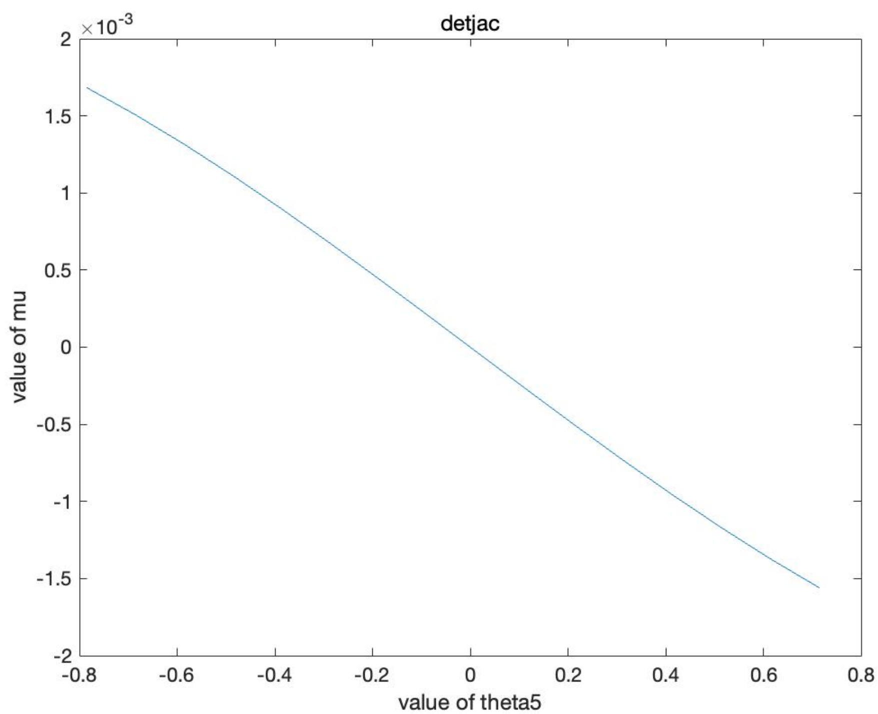

Figure 14: μ against θ_5 (detjac)

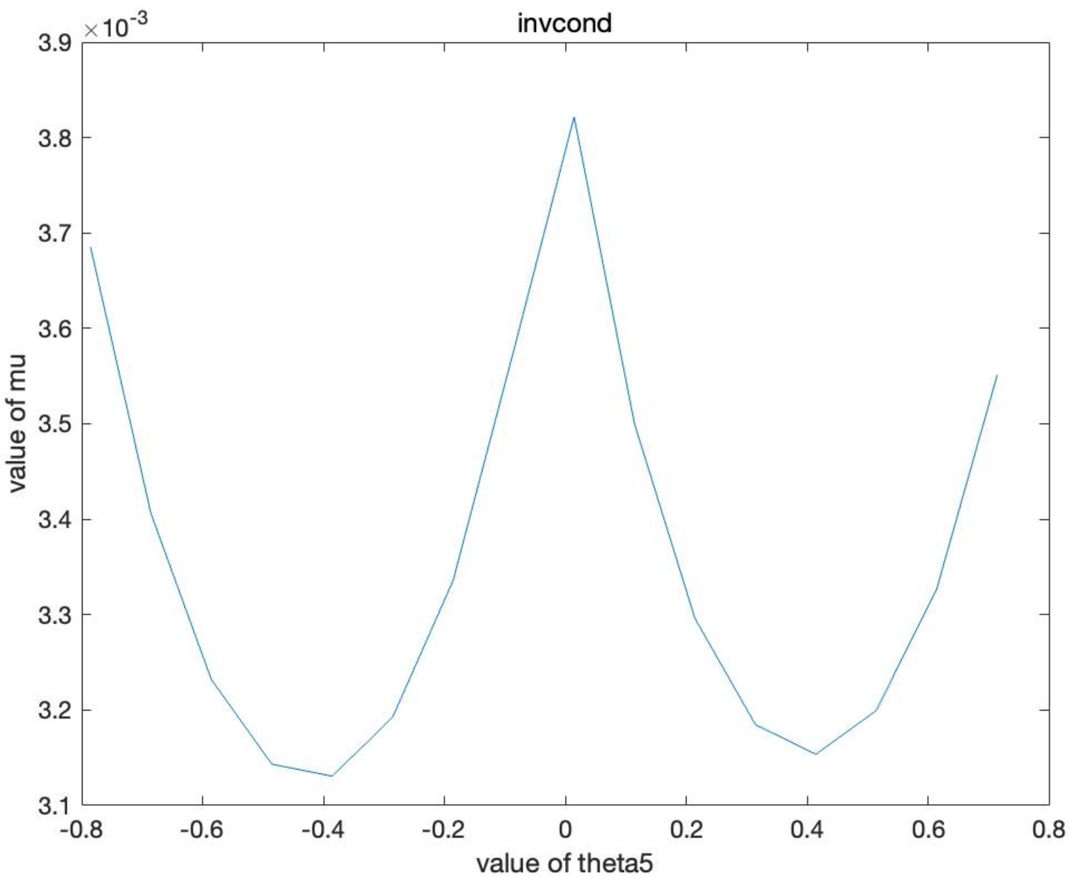

Figure 15: μ against θ_5 (invcond)

Test for getXi()

To test the getXi() function, two random homogenous transformation matrices are generated by function ur5GwdKin().

```
gst1 = ur5FwdKin([pi/3;pi/3;0;0;pi/3;pi/3])
gst2 = ur5FwdKin([pi/4;pi/4;pi/3;pi/2;pi/3;0])
```

And the result is shown below

```
gst1 =

   -0.0625    0.9743   -0.2165    0.0507
    0.7578    0.1875    0.6250    0.3889
    0.6495   -0.1250   -0.7500   -0.7276
         0         0         0    1.0000


gst2 =

   -0.2709   -0.1830   -0.9451    0.0028
    0.9539   -0.1830   -0.2380    0.2158
   -0.1294   -0.9659    0.2241   -0.4799
         0         0         0    1.0000
```

Figure 16: The gst calculated by ur5FwdKin()

By using the getXi() function and expm() function in MATLAB, we can obtain the same results:

```
gst1_prime =

   -0.0625    0.9743   -0.2165    0.0507
    0.7578    0.1875    0.6250    0.3889
    0.6495   -0.1250   -0.7500   -0.7276
         0         0         0    1.0000


gst1_prime2 =

   -0.2709   -0.1830   -0.9451    0.0028
    0.9539   -0.1830   -0.2380    0.2158
   -0.1294   -0.9659    0.2241   -0.4799
         0         0         0    1.0000
```

Figure 17: The gst calculated by getXi() and expm()

Test for ur5RRcontrol()

Case 1:

We initialize the robot to starting position [0, 0, pi/10, 0, pi/10, 0]. We set the goal position to be [pi/2, pi/3, pi/6, 0, pi/4, 0]. By capping the control gain by 5, the robot moved to the goal position within 10 seconds. The final position error is about 0.4984 cm. This could be improved by setting a lower positional limit, but it would take a much longer time to reach the goal position.

The MATLAB output is as following:

```
fwdKinToolFrame =

  tf frame with properties:

         frame_name: 'fwdKinToolFrame'
    base_frame_name: 'base_link'
               pose: [4×4 double]
             tftree: [1×1 ros.TransformationTree]

    "Goal received, starting moving ur5..."

    "Goal position reached..."

   -0.7071    0.0000   -0.7071   -0.1676
   -0.0000    1.0000    0.0000    0.1178
    0.7071    0.0000   -0.7071   -0.7292
         0         0         0    1.0000

   -0.7084    0.0026   -0.7058   -0.1633
   -0.0012    1.0000    0.0050    0.1206
    0.7058    0.0044   -0.7084   -0.7286
         0         0         0    1.0000

The final error is 0.4984
```

Figure 18: The transformation matrices and error

The Rviz screenshot is as following:

Figure 19: The rviz simulation for case 1

Case 2:

We moved the robot to an initial condition that is singularity. According to ur5 kinematics, a joint position of [0, 0, 0, 0, 0, 0] would cause singularity (if joint 3 or joint 5 have 0, there will be singularity). Hence, we let the robot to move to the position above.

The MATLAB output is as following:

```
    "Goal received, starting moving ur5..."

    "ABORTING: SINGULARITY ENCOUNTERED..."

The final error is -1
```

Figure 20: The Matlab output for case 2

The Rviz screenshot is as following:

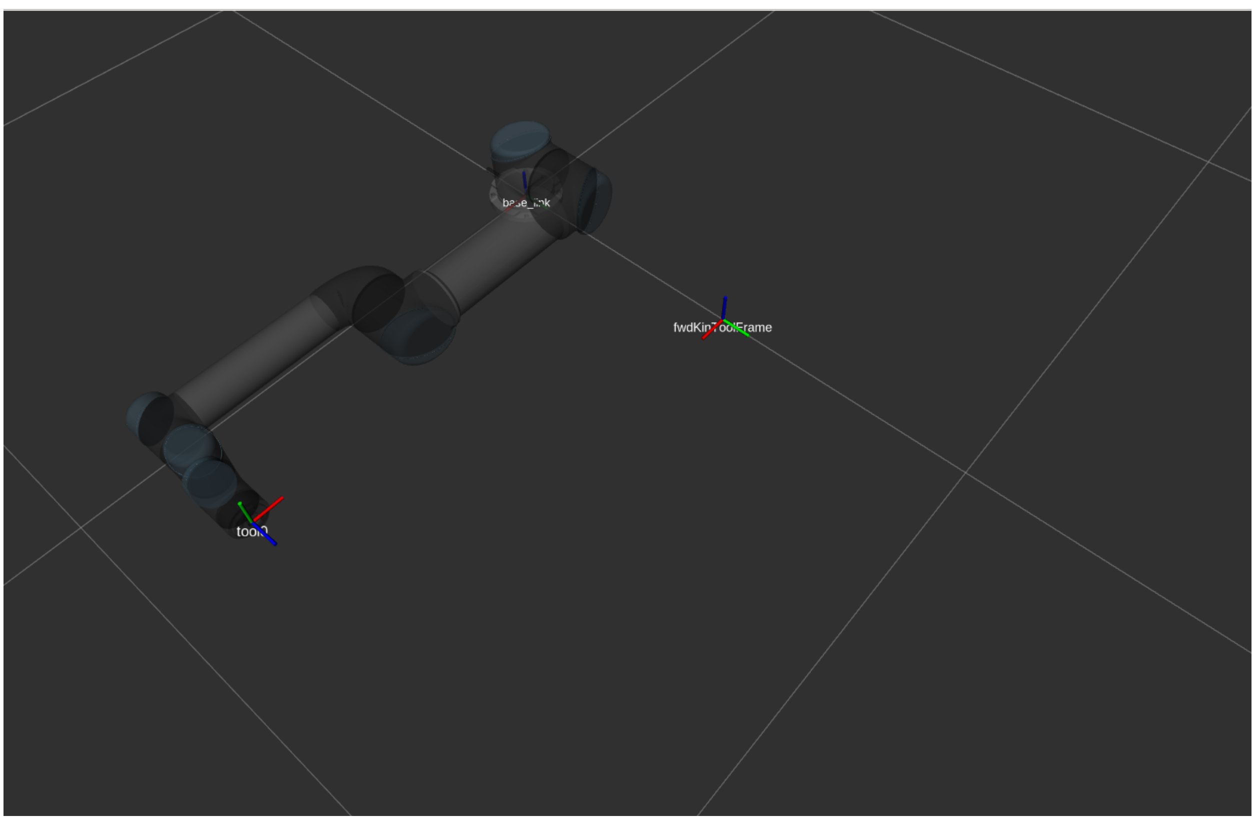

Figure 21: The rviz simulation for case 2

Contribution:

Name	Work	Contribution
Qihang Li	Build and test ur5FwdKin.m and ur5BodyJacobian.m. Write the corresponding part of report. Assemble the report.	33.33%
Pupei Zhu	Build and test manipulability.m and getXi.m. Write the corresponding part of report.	33.33%
Jintan Zhang	Build and test ur5RRcontrol.m. Write the corresponding part of report.	33.33%