COMP0127 Robotic Systems Engineering
Coursework 1: Linear Algebra and Forward Kinematics

Douglas Chiang Hou Nam
SN:15055142
hou.chiang.20@ucl.ac.uk

November 7, 2020

Rigid Transformations

1. Given an arbitrary 3D rotation matrix:

$$\boldsymbol{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \tag{1}$$

a. Since $\boldsymbol{R}$ is an orthonormal matrix, by the normality constraints of a generic rotation matrix, the norm of each column is unity:

$$\sqrt{r_1^2 + r_4^2 + r_7^2} = 1 \tag{2}$$

$$\sqrt{r_2^2 + r_5^2 + r_8^2} = 1 \tag{3}$$

$$\sqrt{r_3^2 + r_6^2 + r_9^2} = 1 \tag{4}$$

Therefore, every element r_i^2 will have a magnitude ≤ 1, which implies $\|r_i\| \leq 1$.

b. For any rotation matrix $\boldsymbol{R}$ can be expressed in terms of the axis of rotation unit vector k and the angle of rotation θ as $\boldsymbol{R}_{k,\theta}$:

$$\boldsymbol{R}_{k,\theta} = \boldsymbol{I} + sin(\theta)[\boldsymbol{k}]_\times + (1 - cos(\theta)[\boldsymbol{k}]_\times^2 \tag{5}$$

With this, the expression for $\boldsymbol{R}_{-k,-\theta}$ is written as:

$$\begin{aligned} \boldsymbol{R}_{k,\theta} &= \boldsymbol{I} + sin(-\theta)[\boldsymbol{-k}]_\times + (1 - cos(-\theta)[\boldsymbol{-k}]_\times^2 \\ &= \boldsymbol{I} + sin(\theta)[\boldsymbol{k}]_\times + (1 - cos(\theta)[\boldsymbol{k}]_\times^2 \\ &= \boldsymbol{R}_{k,\theta} \end{aligned}$$

which proved $\boldsymbol{R}_{k,\theta} = \boldsymbol{R}_{-k,-\theta}$.

c. Given two arbitrary Catesian coordinate frames a and b, the operation of the rows of ${}^{\boldsymbol{a}}\boldsymbol{R_b}$ represent the followings:

- The first row of ${}^{\boldsymbol{a}}\boldsymbol{R_b}$ is to transform all x components of the frame from the frame b to frame a.
- The second row of ${}^{\boldsymbol{a}}\boldsymbol{R_b}$ is to transform all y components of the frame from the frame b to frame a.
- The third row of ${}^{\boldsymbol{a}}\boldsymbol{R_b}$ is to transform all z components of the frame from the frame b to frame a.

d. **Eigenvector and rotation axis**

For a rotation matrix $\boldsymbol{R}$ rotating around an arbitrary vector $\boldsymbol{u}$ in 3D, if we apply $\boldsymbol{R}$ to any parallel vector $\boldsymbol{v}$, it will give:

$$\boldsymbol{Rv} = \boldsymbol{v} \tag{6}$$

because it is a rotation around $\boldsymbol{v}$, so it will not have any rotational effect to $\boldsymbol{v}$. Then, with analogy to the definition of eigenpairs:

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \tag{7}$$

We can see that in (6) the eigenvalue is 1 and $\boldsymbol{v}$ is the eigenvector. Therefore, we can conclude that the eigenvector having the eigenvalue $\lambda = 1$ is actually the rotation axis.

Eigenvalue and rotation angle

A method to calculate the angle of rotation for an arbitrary 3D rotation matrix $\boldsymbol{R}$ is:

$$\theta = cos^{-1}\left(\frac{1}{2}(r_1 + r_5 + r_9 - 1)\right)$$
$$\theta = cos^{-1}\left(\frac{1}{2}(Tr(\boldsymbol{R}) - 1)\right) \tag{8}$$

Then if we eigen-decompose the rotation matrix $\boldsymbol{R}$ and make use of the property of trace such that $Tr(\boldsymbol{AB}) = Tr(\boldsymbol{BA})$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices with size $m \times n$ and $n \times m$:

$$\begin{aligned}
\boldsymbol{R} &= \boldsymbol{VDV^{-1}} \\
\boldsymbol{V^{-1}RV} &= \boldsymbol{D} \\
Tr([\boldsymbol{V^{-1}R}]\boldsymbol{V}) &= Tr(\boldsymbol{D}) \\
Tr(\boldsymbol{V}[\boldsymbol{V^{-1}R}]) &= Tr(\boldsymbol{D}) \\
Tr(\boldsymbol{VV^{-1}R}) &= Tr(\boldsymbol{D}) \\
Tr(\boldsymbol{IR}) &= Tr\left(\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}\right) \\
Tr(\boldsymbol{R}) &= \sum_i \lambda_i
\end{aligned}$$

So substitute the result above to (8), we have:

$$|\theta| = cos^{-1}\left(\frac{1}{2}(Tr(\boldsymbol{R}) - 1)\right)$$
$$|\theta| = cos^{-1}\left(\frac{1}{2}\left(\sum_i \lambda_i - 1\right)\right) \tag{9}$$

which displays the relation between the eigenvalues and the angle of rotation.

2. a. Given:

$$\boldsymbol{R} = \begin{bmatrix} -\frac{\sqrt{3}}{4} - \frac{\sqrt{6}}{8} & -\frac{\sqrt{2}}{4} & -\frac{3}{4} + \frac{\sqrt{2}}{8} \\ \frac{1}{4} - \frac{3\sqrt{2}}{8} & -\frac{\sqrt{6}}{4} & \frac{\sqrt{3}}{4} + \frac{\sqrt{6}}{8} \\ -\frac{\sqrt{6}}{4} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{4} \end{bmatrix} \tag{10}$$

Its ZYZ Euler angle representation is:

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{R_z(\gamma)R_y(\beta)R_z(\alpha)} \\ &= \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\alpha)cos(\beta) & -cos(\beta)sin(\alpha) & sin(\beta) \\ sin(\alpha) & cos(\alpha) & 0 \\ -cos(\alpha)sin(\beta) & sin(\alpha)sin(\beta) & cos(\beta) \end{bmatrix} \\ &= \begin{bmatrix} c(\alpha)c(\beta)c(\gamma) - s(\alpha)s(\gamma) & -c(\alpha)s(\gamma) - c(\beta)c(\gamma)s(\alpha) & c(\gamma)s(\beta) \\ c(\gamma)s(\alpha) + c(\alpha)c(\beta)s(\gamma) & c(\alpha)c(\gamma) - c(\beta)s(\alpha)s(\gamma) & s(\beta)s(\gamma) \\ -c(\alpha)s(\beta) & s(\alpha)s(\beta) & c(\beta) \end{bmatrix} \end{aligned}$$

Note that c is short-hand for $cos(\cdot)$ and s is short-hand for $sin(\cdot)$. comparing with (10) gives:

$$\begin{aligned} \alpha &= sin^{-1}\left(\frac{\frac{\sqrt{2}}{2}}{sin(\beta)}\right) \\ \beta &= cos^{-1}\left(\frac{\sqrt{2}}{4}\right) \\ \gamma &= sin^{-1}\left(\frac{\frac{\sqrt{3}}{4} + \frac{\sqrt{6}}{8}}{sin(\beta)}\right) \end{aligned}$$

which gives $\alpha = 0.8571, \beta = 1.2094, \gamma = 0.9112$.
Substituting α, β, γ back to $\boldsymbol{R_z(\alpha)}, \boldsymbol{R_y(\beta)}, \boldsymbol{R_z(\gamma)}$ gives:

$$\boldsymbol{R_z(\alpha)} = \begin{bmatrix} 0.6547 & -0.7559 & 0 \\ 0.7559 & 0.6547 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$\boldsymbol{R_y(\beta)} = \begin{bmatrix} 0.3536 & 0 & 0.9354 \\ 0 & 1 & 0 \\ -0.9354 & 0 & 0.3536 \end{bmatrix} \tag{12}$$

$$\boldsymbol{R_z(\gamma)} = \begin{bmatrix} 0.6128 & -0.7902 & 0 \\ 0.7902 & 0.6128 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

which corresponds to entries in representation $\boldsymbol{R = R_z(\alpha)R_y(\beta)R_z(\gamma)}$ as required.

b. Euler Angle representation of a rotation matrix express the rotation in terms of the rotation angles α, β, γ. Since in terms of angle, there are many expressions that are equivalent, such as angle $\alpha = \alpha + 2k\pi$, where k is just a constant, there can be many value of α, β, γ that can represent the same rotation.
Operation-wise, there can be many combination of rotation of α, β, γ to arrive the same end configuration. For example, a ZYX rotation matrix with $\alpha = 0, \beta = \frac{\pi}{2}, \gamma = 0$ is equvalent to that when $\alpha = -\frac{\pi}{2}, \beta = \frac{\pi}{2}, \gamma = -\frac{\pi}{2}$, resulting a rotation matrix:

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{R_z(\alpha) R_y(\beta) R_x(\gamma)} \\ &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \end{aligned}$$

Also, The rotation matrix can be expressed in terms of different basis, therefore have more than one representation.

c. The limitation includes:

1. Discontinuities, which will jump from 0°to 360°.
2. Gimbal lock, which happens when 1 of 3 axes rotate and become parallel with any other one axis, result in loss of 1 degree of freedom in the rotational system.

To avoid these limitations, we can avoid Euler Angles and use Quaternions, or convert Euler Angles to Quaternions. Quaternions do not have both of the limitations stated above. However, Quaternions have 1 more parameters compared to Euler Angles and have more difficult geometric intuition. If we have to use Euler Angles, the only method to avoid this is to create a bound for the angles so that it will not result in a Gimbal lock configuration.

3. a. To covert (10) in Question 2a) to a quaternion $\boldsymbol{q} = [q_w \ q_x \ q_y \ q_z]^T$, first, a quaternion $\boldsymbol{q}$, it can be expressed as:

$$\boldsymbol{q}(\boldsymbol{u}, \theta) = cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{14}$$

Then from question 1, we also know that for any rotation matrix $\boldsymbol{R}$:

$$|\theta| = cos^{-1}\left(\frac{1}{2}\left(Tr(\boldsymbol{R}) - 1\right)\right) \tag{15}$$

In addition, the rotation axis $\boldsymbol{u}$:

$$\boldsymbol{u} = \frac{1}{2sin(\theta)}\begin{bmatrix} r_8 - r_6 \\ r_3 - r_7 \\ r_4 - r_2 \end{bmatrix} \tag{16}$$

By using the components in (10), we can get:

$$\theta = \pm 3.0971$$
$$\boldsymbol{q} = \begin{bmatrix} 0.0223 \\ -0.3604 \\ 0.4397 \\ 0.8224 \end{bmatrix}$$

b. With (14), $-\boldsymbol{q}$ is expressed as:

$$\begin{aligned} \boldsymbol{q}(-\boldsymbol{u}, -\theta) &= cos\left(-\frac{\theta}{2}\right) + sin\left(-\frac{\theta}{2}\right)\begin{bmatrix} -u_x \\ -u_y \\ -u_z \end{bmatrix} \\ &= cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \\ &= \boldsymbol{q}(\boldsymbol{u}, \theta) \end{aligned}$$

which proves the relation.
On the other hand for $\boldsymbol{q} = [q_w \ q_x \ q_y \ q_z]^T$ and $-\boldsymbol{q} = [-q_w \ -q_x \ -q_y \ -q_z]^T$, if we substitute both quaternion into the rotation matrix representation:

$$\boldsymbol{R} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix} \tag{17}$$

the resultant rotation matrix will be the same.

c. There are two cases for two arbitrary matrices R_a and R_b to be commutative:

1. Rotation about the same axis. I.e:

$$\boldsymbol{R = R_a(\alpha)R_b(\beta)}$$

Since the angle $\alpha + \beta$ added up in any order are the same, the operation is commutative.

2. In general case, for two matrices to be commutative, both matrices need to be simultaneously diagonalizable, such that $\boldsymbol{V_a = V_b = V}$:

$$\begin{aligned}
\boldsymbol{R_aR_b} &= \boldsymbol{V_aD_aV_a^{-1}V_bD_bV_b^{-1}} \\
&= \boldsymbol{VD_aV^{-1}VD_bV^{-1}} \\
&= \boldsymbol{VD_aD_bV^{-1}} \\
&= \boldsymbol{VD_bD_aV^{-1}} \\
&= \boldsymbol{VD_bV^{-1}VD_aV^{-1}} \\
&= \boldsymbol{R_bR_a}
\end{aligned}$$

So any rotation that can satisfy this will be commutative.

4. a. The input and output data type for each service is formatted as below:

```
#INPUTS' data type
---
#OUTPUTS' data type
```

Input and output for quat2rodrigues.srv

```
geometry_msgs/Quaternion q
---
std_msgs/Float64 x
std_msgs/Float64 y
std_msgs/Float64 z
```

quat2zyx.srv

```
geometry_msgs/Quaternion q
---
std_msgs/Float64 z
std_msgs/Float64 y
std_msgs/Float64 x
```

rotmat2quat.srv

```
std_msgs/Float64MultiArray r1
std_msgs/Float64MultiArray r2
std_msgs/Float64MultiArray r3
---
geometry_msgs/Quaternion q
```

b. To convert a Quaternion to an Euler Angle representation $\boldsymbol{R} = \boldsymbol{R_z}(\gamma)\boldsymbol{R_y}(\beta)\boldsymbol{R_x}(\alpha)$, we need to compare the Quaternion's rotation matrix representation (17) to that of the resultant matrix of $\boldsymbol{R}$. By comparing three equations in the matrix, we can get the value for α, β, γ:

```
# r_2,0: -sin(beta) = 2qxqz - 2qyqw
beta  = np.arcsin(-(2*req.q.x*req.q.z - 2*req.q.y*req.q.w))
res.y.data = beta

# r_2,1: cos(beta)*sin(alpha) = 2qyqz + 2qxqw
alpha = np.arcsin((2*req.q.y*req.q.z + 2*req.q.x*req.q.w)/np
    .cos(beta))
res.x.data = alpha

# r_1,0: cos(beta)*sin(gamma) = 2qxqy + 2qzqw
gamma = np.arcsin((2*req.q.x*req.q.y + 2*req.q.z*req.q.w)/np
    .cos(beta))
res.z.data = gamma
```

c. To convert a Quaternion to Rodrigues representation, apart from the one shown in lecture, which is:

$$\boldsymbol{b} = \boldsymbol{a}cos(\theta) + (\boldsymbol{u} \times \boldsymbol{a})sin(\theta) + \boldsymbol{u}(\boldsymbol{u} \cdot \boldsymbol{a})(1 - cos(\theta)) \tag{18}$$

There is another method to calculate the Rodrigues representations by the following equation discussed in Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." Mechanism and Machine Theory, 92, 144-152. Elsevier, 2015.":

$$\boldsymbol{b} = tan\left(\frac{\theta}{2}\right)\boldsymbol{u} \tag{19}$$

In the programme, the Rodrigues representation will be calculated by (19)

```
# ux*sin(theta/2) = qx
ux = req.q.x/np.sin(theta/2)

# uy*sin(theta/2) = qy
uy = req.q.y/np.sin(theta/2)

# uz*sin(theta/2) = qz
uz = req.q.z/np.sin(theta/2)

# Rodrigues vector elements:
res.x.data = np.tan(theta/2)*ux
res.y.data = np.tan(theta/2)*uy
res.z.data = np.tan(theta/2)*uz
```

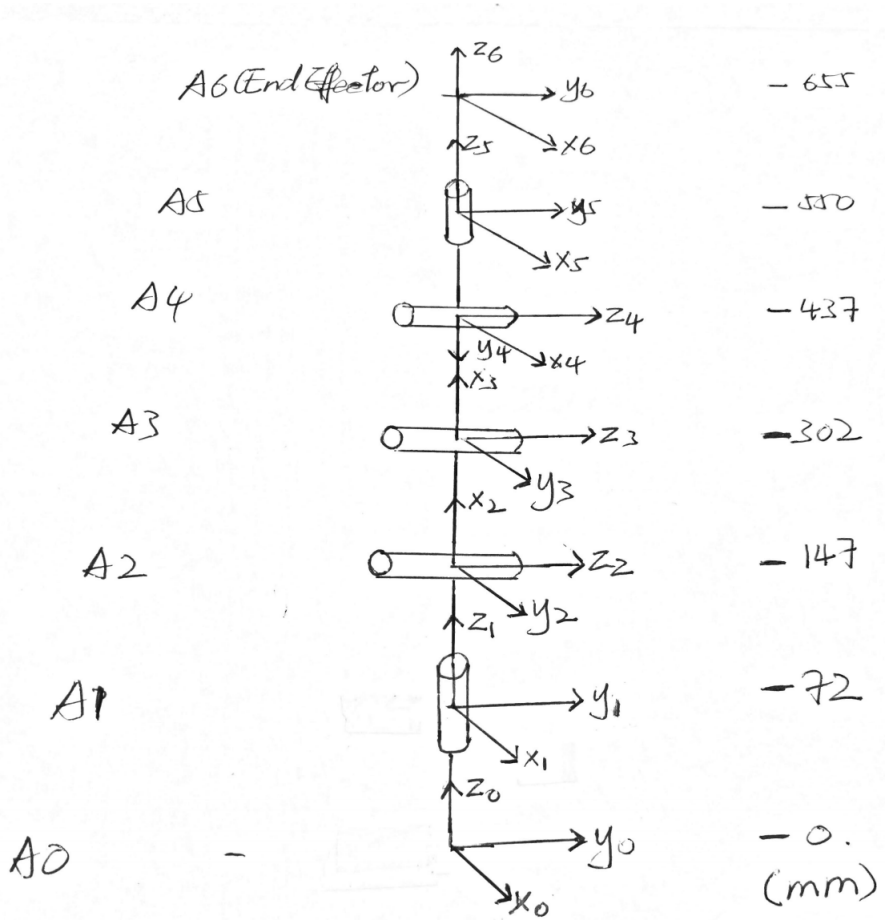
d. To convert a Rotation matrix to a Quaternion, we can make use of (15) and (16), but since the term $2sin(\theta)$ in (16) can lead to an error when $\theta = 0$, the modified equation is as in the code below:

```
theta = np.arccos(0.5*(req.r1.data[0] + req.r2.data[0] + req
    .r3.data[2] - 1))

xr = req.r3.data[1] - req.r2.data[2]
yr = req.r1.data[2] - req.r3.data[0]
zr = req.r2.data[0] - req.r1.data[1]

# Modified formula to compensate the situation when sin(
    theta) --> 0:
x = xr/np.sqrt(np.power(xr, 2) + np.power(yr, 2) + np.power(
    zr, 2))
y = yr/np.sqrt(np.power(xr, 2) + np.power(yr, 2) + np.power(
    zr, 2))
z = zr/np.sqrt(np.power(xr, 2) + np.power(yr, 2) + np.power(
    zr, 2))

res.q.w = np.cos(theta/2)
```

```
13 res.q.x = x * np.sin(theta/2)
14 res.q.y = y * np.sin(theta/2)
15 res.q.z = z * np.sin(theta/2)
```

Robot Kinematic

5. a. Using Modified Denavit-Hartenberg (DH) convention, the KUKA YouBot will have the following frames and parameters:



i	a_{i-1} (m)	α_{i-1} (rad)	θ_i (rad)	d_i (m)
1	0	0	θ_1	0.072 + 0.075
2	0	$-\frac{\pi}{2}$	$\theta_2 - \frac{\pi}{2}$	0
3	0.155	0	θ_3	0
4	0.135	0	$\theta_4 + \frac{\pi}{2}$	0
5	0	$+\frac{\pi}{2}$	θ_5	0.113
6 (End-Effector)	0	0	0	0.105

The 0.075 added to d_1 is because the first joint is just rotating around z-axis in the global sense, and to compensate the lost of height when moving from frame A1 to frame A2.

b. The difference between Standard DH and Modified DH are as follows:
Standard DH:

1. The frame coordinate O_{i-1} is located in joint i and O_i is located in joint $i+1$
2. The transformation matrix from i to $i-1$ is:

$$^{i-1}T_i = Trans_{z_{i-1}}(d_i)Rot_{z_{i-1}}(\theta_i)Trans_{x_i}(a_i)Rot_{x_i}(\alpha_i)$$

and the parameters will be as follows:

- On z_{i-1} axis:
  - d_i: The distance along z_{i-1} between x_{i-1} and x_i
  - θ_i: The angle between x_{i-1} and x_i around z_{i-1}
- On x_i axis:
  - a_i: The distance along x_i (the common normal) between the z_{i-1} and z_i
  - α_i: The angle between z_{i-1} and z_i around x_i

Modified DH:

1. The frame coordinate O_{i-1} is located in joint $i-1$ and O_i is located in joint i
2. The transformation matrix from i to $i-1$ is:

$$^{i-1}T_i = Rot_{x_{i-1}}(\alpha_{i-1})Trans_{x_{i-1}}(a_{i-1})Rot_{z_i}(\theta_i)Trans_{z_i}(d_i)$$

and the parameters will be as follows:

- On z_i axis:
  - d_i: The distance along z_i between x_{i-1} and x_i
  - θ_i: The angle between x_{i-1} and x_i around z_i
- On x_{i-1} axis:
  - a_{i-1}: The distance along x_{i-1} (the common normal) between the z_{i-1} and z_i
  - α_{i-1}: The angle between z_{i-1} and z_i around x_{i-1}

c. This part have two task:

1. Take DH variables from the rostopic that publishes joint data. To achieve this, we first have to initialize a node which we called it 'forward_kinetic_node' and then subscribe to the topic '/joint_states' and its function forward_kinetic as shown:

```
def main():
    rospy.init_node('forward_kinematic_node')
    ##TODO: Initialise the subscriber to the topic "/
       joint_states" and its callback function
       forward_kinematic
    sub  = rospy.Subscriber('/joint_states', JointState,
       forward_kinematic)
    rate = rospy.Rate(10)
    rospy.spin()
    rate.sleep()
```

2. Publish a set of transformations relating the frame "base link" and each frame on the arm "arm5c link i" where i is the frame, using tf messages. To do this, we first declare the joint names:

```
name_link = ['arm5c_link_1', 'arm5c_link_2', '
   arm5c_link_3', 'arm5c_link_4', 'arm5c_link_5', '
   arm5c_link_6']
```

Then we have to declare a broadcaster:

```
br = tf2_ros.TransformBroadcaster()
```

After that we need to write a function which will:

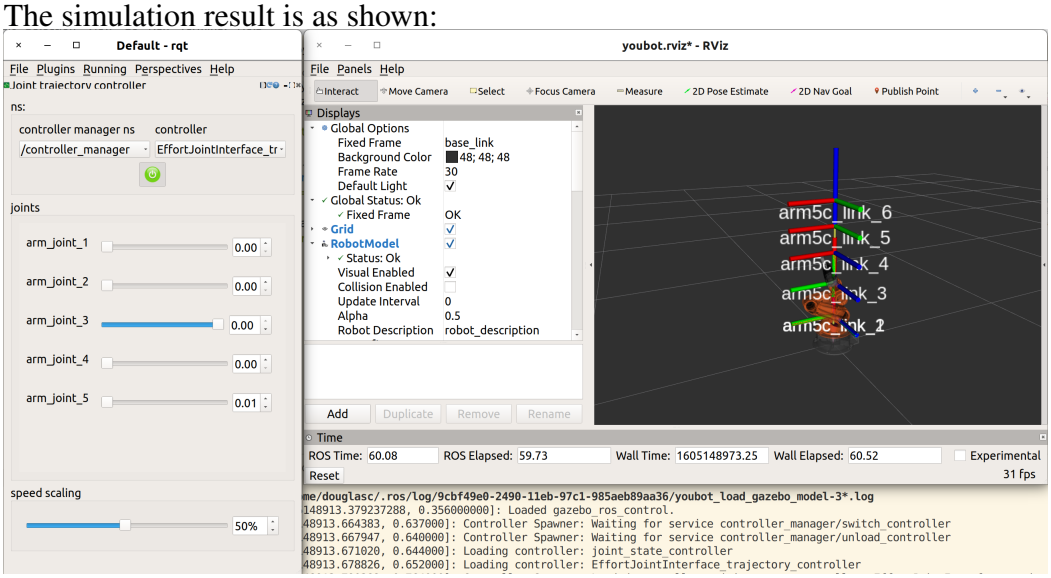
- Transform the frames from the end-effector to the base link using Modified DH.
- Transform the resultant matrix to a quaternion.
- And publish it.

These are done in the function as shown:

```
def forward_kinematic(joint_msg):
    T = np.identity(4)

    transform = TransformStamped()
    for i in range(6):

        A = fkine_Modified(a[i], alpha[i], d[i], theta[i]
            - joint_msg.position[i])

        transform.header.stamp      = rospy.Time.now()
        transform.header.frame_id   = 'base_link'
        transform.child_frame_id    = name_link[i]

        T = T.dot(A)
```

```
        transform.transform.translation.x    = T[0, 3]
        transform.transform.translation.y    = T[1, 3]
        transform.transform.translation.z    = T[2, 3]
        transform.transform.rotation         = rotmat2q(T)

        br.sendTransform(transform)
```

The simulation result is as shown:



6. The error of rigid transformation is represented as:

$$\boldsymbol{T_{error} = T_{gt}^{-1} T_{est}} \tag{20}$$

a. From (20), we can further derive the followings:

$$\boldsymbol{T_{error}^{-1} = T_{est}^{-1} T_{gt}} \tag{21}$$
$$\boldsymbol{T_{est} = T_{gt} T_{error}} \tag{22}$$
$$\boldsymbol{T_{est}^{-1} = T_{error}^{-1} T_{gt}^{-1}} \tag{23}$$
$$\boldsymbol{T_{gt} = T_{est} T_{error}^{-1}} \tag{24}$$
$$\boldsymbol{T_{gt}^{-1} = T_{error} T_{est}^{-1}} \tag{25}$$

From the above, utilizing (23) and (24), we will have:

$$\boldsymbol{T_{gt} T_{est}^{-1} = T_{est} T_{error}^{-1} T_{error}^{-1} T_{gt}^{-1}} \tag{26}$$

Therefore, the rotation and translation error will not be the same. For example, if we use the result in 6e), the resultant Rodrigues representation is $(0, 0.577, 0)$ for $T_{gt}^{-1} T_{est}$ and $(0, -0.577, 0)$ for $T_{gt} T_{est}^{-1}$, while the translation positioning error is 0.0044 for $T_{gt}^{-1} T_{est}$ and 0.0013 for $\boldsymbol{T_{gt} T_{est}^{-1}}$.

b. If we use $\boldsymbol{T_{est}^{-1} T_{gt}}$ instead of $\boldsymbol{T_{gt}^{-1} T_{est}}$, noticing that:

$$\boldsymbol{T_{error} = T_{gt}^{-1} T_{est}}$$
$$\boldsymbol{T_{error}^{-1} = T_{est}^{-1} T_{gt}}$$

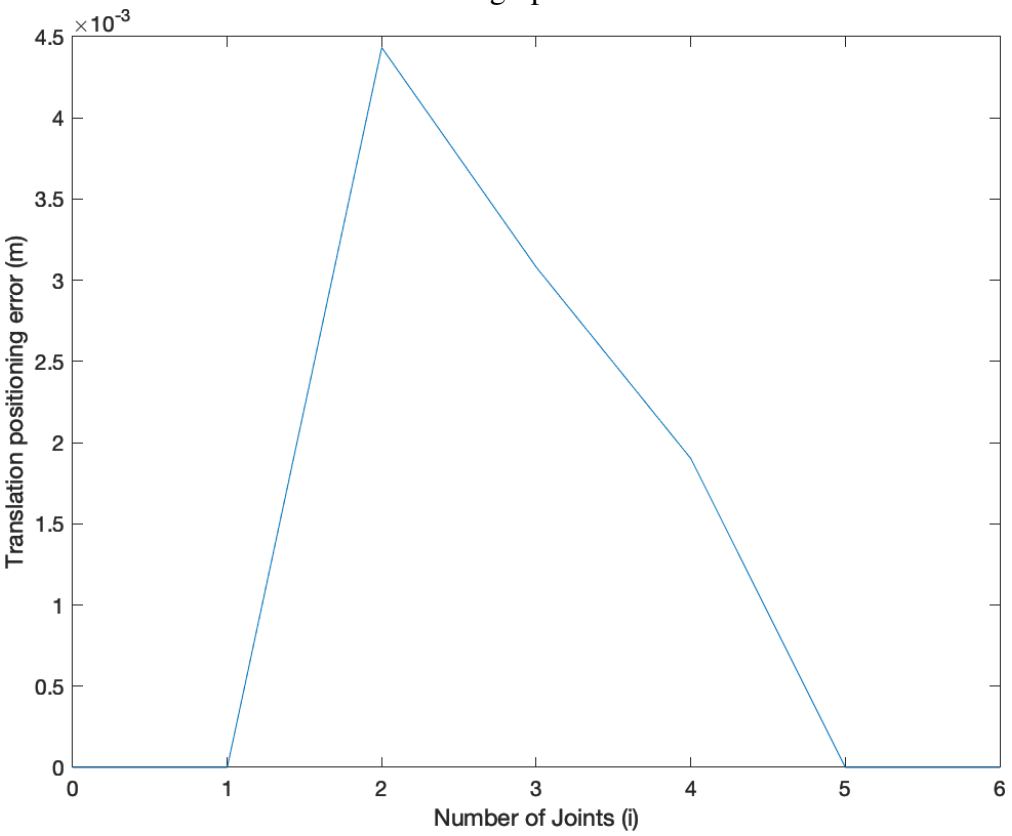
which is actually the inverse of $\boldsymbol{T_{error}}$. Therefore, the translation error will still be the same. For example, if we use the result in 6e), the translation positioning error of T_{error} and T_{error}^{-1} are both found to be 0.0044m.

c. The positioning error in the translation component when $\theta_1 = 0.5^\circ$ is 0m.

d. The positioning error in the translation component when $\theta_4 = 0.5^\circ$ is 0.0019m.

e. The positioning error in the translation component when $\theta_2 = 0.5^\circ$ is 0.0044m.

f. The translation position error of every joint at zero position producing $\theta_i = 0.5^\circ$ in the KUKA YouBot is as shown in the graph:



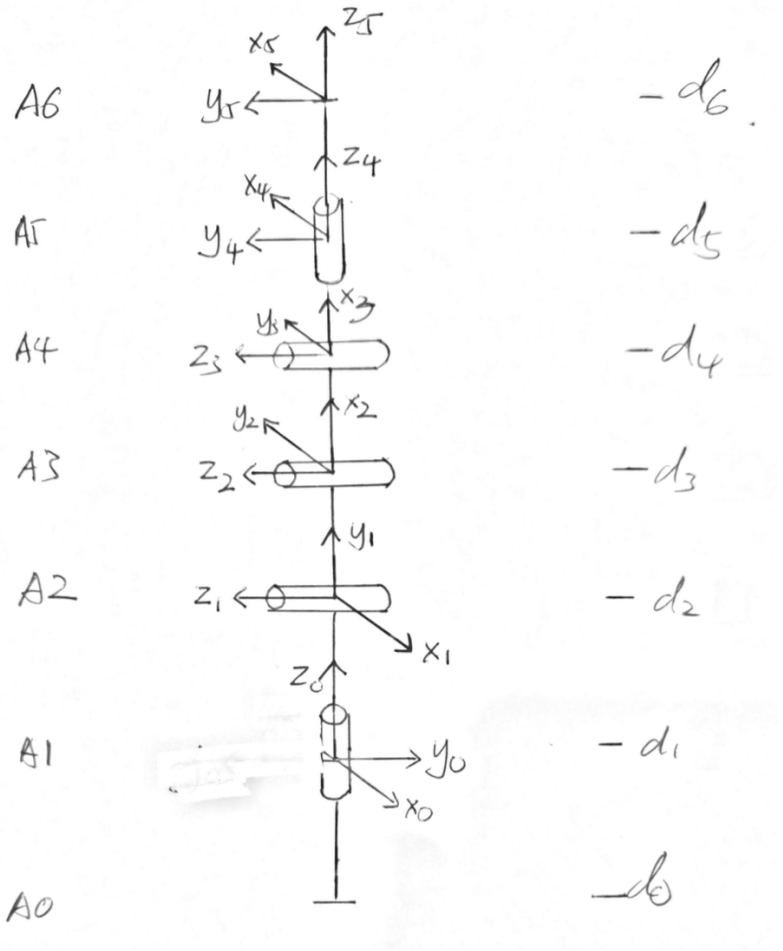
From the graph (detailed in MATLAB file Q6.pdf/Q6.mlx), we can see that the error from the end- effector to Joint i is 0 at Joint 1, then subsequently peak at Joint 2 at around 4.5×10^{-3} After that, falls to 0 at Joint 5. This may due to the rotation of the axis is around z-axis of the global coordinate frame on the first axis, there is no translation at all. However, start from Joint 2 to Joint 4, since these joints can translate the end-effector through controlling the angular displacement, the translation positioning error at Joint 2 is maximum because the translation positioning error resulting from the angular displacement of the joints all the way from Joing 4 to Joint 2 is accumulated. So joints closer to the end-effector will have smaller translation positioning error. For Joint 5, since it is rotating at a perpendicular direction, the translation positioning error is 0.

Therefore, from this implication, a bigger robot will have larger translation position error at the joints that can translate the position of the end-effector and the error for the joints will follow the same trend.

7. a. By analysing the xacro file "robot description/youbot description/urdf/youbot arm/arm.urdf.xacro", since we are working with the joints, the following parameters are extracted:

Joint i	x	y	z	r (roll)	p (pitch)	y (yaw)
1	0.024	0	0.096	0	0	$\frac{170\pi}{180}$
2	0.033	0	0.019	0	$-\frac{65\pi}{180}$	0
3	0	0	0.155	0	$\frac{146\pi}{180}$	0
4	0	0	0.135	0	$-\frac{102.5\pi}{180}$	0
5	-0.002	0	0.130	0	0	$\frac{167.5\pi}{180}$
6 (End-Effector)	0	0	0.055	0	0	0

Then assuming the robot is now at its zero position, using Standard Denavit-Hartenberg (DH) convention the KUKA YouBot have the following frames and parameters:



i	θ_i (rad)	d_i (m)	a_i (m)	α_i (rad)
1	θ_1	$d_2 - d_1$	0	$\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2}$	0	$d_3 - d_2$	0
3	θ_3	0	$d_4 - d_3$	0
4	θ_4	0	0	$\frac{\pi}{2}$
5 (EE)	0	$d_6 - d_5$	0	0

Then following the definitions of each parameters to combine the values given in the xacro file and keeping in mind that the angle is positive when rotating anticlockwise and vice versa, we arrive the following parameters:

i	θ_i (rad)	d_i (m)	a_i (m)	α_i (rad)
1	$\theta_1 + \frac{170\pi}{180}$	$0.096 + 0.030 + 0.019$	$0.033 + 0.024 - 0.024$	$\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2} + \frac{65\pi}{180}$	0	0.155	0
3	$\theta_3 - \frac{146\pi}{180}$	0	0.135	0
dummy	$\theta_d + \frac{\pi}{2} + \frac{102.5\pi}{180}$	0	0	$\frac{\pi}{2}$
4	$\theta_4 + \frac{167.5\pi}{180}$	0.130	0	0
5 (EE)	0	0.055	0	0

There is a dummy frame in the final DH parameter because a pure rotation from frame 3 to 4 is needed for the program which has its z axis pointing to the end-effector, before translating frame 4 to the appropriate position along the z axis of the dummy frame.

b. For this part of the question, we have two tasks, as follows:

1. Take DH variables from the rostopic that publishes joint data. To achieve this, like 5c), we first have to initialize a node which we called it 'forward_kinetic_node' and then subscribe to the topic '/joint_states' and its function forward_kinetic as shown:

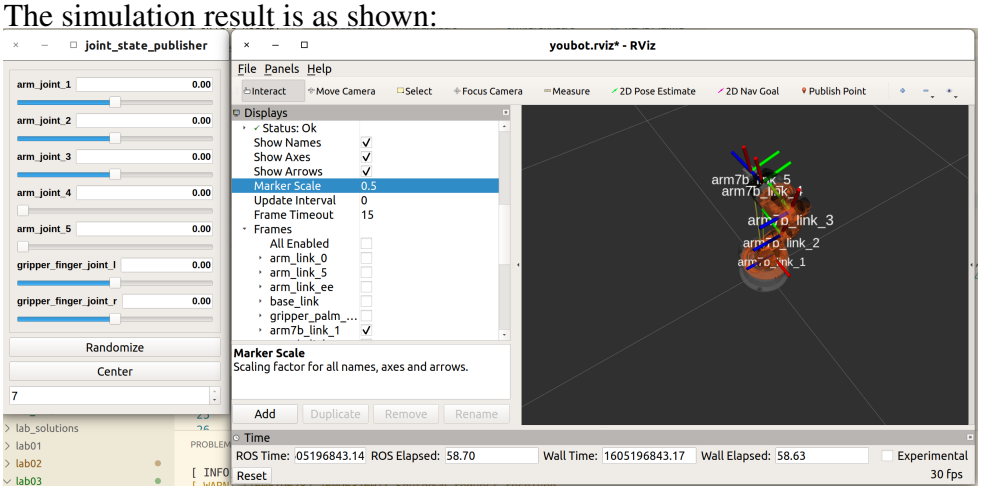
```
1  def main():
2      rospy.init_node('forward_kinematic_node')
3      ##TODO: Initialise the subscriber to the topic "/
           joint_states" and its callback function
           forward_kinematic
4      sub  = rospy.Subscriber('/joint_states', JointState,
           forward_kinematic)
5      rate = rospy.Rate(10)
6      rospy.spin()
7      rate.sleep()
```

2. Publish a set of transformations relating the frame "base link" and each frame on the arm "arm7b link i" where i is the frame, using tf messages. This part is basically the same as 5c) but since there is a dummy frame, the dummy is not going to be published, so the code for forward_kinematic(joint_msg) is modified in line 19 - 21 as below:

```
1  def forward_kinematic(joint_msg):
2      T = np.identity(4)
3
4      transform = TransformStamped()
5      for i in range(6):
6          A = fkine_standard(a[i], alpha[i], d[i], (theta[i
               ] - joint_msg.position[i]))
7
```

```
        transform.header.stamp       = rospy.Time.now()
        transform.header.frame_id    = 'base_link'
        transform.child_frame_id     = name_link[i]

        T = T.dot(A)

        transform.transform.translation.x   = T[0, 3]
        transform.transform.translation.y   = T[1, 3]
        transform.transform.translation.z   = T[2, 3]
        transform.transform.rotation        = rotmat2q(T)

        # Publish all frames except the dummy one, which
            do the rotation only:
        if (name_link[i] != 'arm7b_link_dummy'):
            br.sendTransform(transform)
```

The simulation result is as shown:



END OF COURSEWORK