UNIVERSITY COLLEGE LONDON (UCL)

COMP0127 – ROBOTIC SYSTEMS ENGINEERING

Coursework 1: Linear Algebra and Forward Kinematics

Department
Computer Science

Name
Wing Chung Law

Student Number
18067536

Submission Date
November 15, 2021

Contents

Linear Algebra

Question 1

1.a

If matrix H satisfies the property $H^T H = I$, it is an orthogonal matrix. For example, let the square matrix H be:

$$H = \begin{bmatrix} \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \end{bmatrix} \tag{1.1}$$

which can be denoted by three 3×1 vector (x, y, z). The matrix multiplication of H^T and H would give a 3×3 identity matrix

$$H^T H = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & \frac{2}{3} \\ \frac{-2}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} & \frac{-2}{3} \end{bmatrix} \begin{bmatrix} \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

The magnitude of each vector in the matrix is given by (1.3), which equals to 1. Hence, the columns of H have a length 1.

$$\text{Magnitude} = \sqrt{\frac{2}{3}^2 + \frac{2}{3}^2 + \frac{1}{3}^2} = 1 \tag{1.3}$$

The dot product between any two vectors is given by (1.4)

$$x \cdot y = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{-2}{3} \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix} = 0,\ x \cdot z = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{-2}{3} \end{bmatrix} = 0,\ y \cdot z = \begin{bmatrix} \frac{-2}{3} \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{-2}{3} \end{bmatrix} = 0 \tag{1.4}$$

which is equal to zero; this shows that the vectors which make up the matrix H are orthonormal and perpendicular to each other. Furthermore, the equation of the dot product is given by

$$x \cdot y = |x||y| \cos\theta = 0 \tag{1.5}$$

where angle θ is the angle between x and y. Hence, a dot product of 0 and a magnitude of 1 will give an angle of 90°between the vectors, so they are orthogonal and are perpendicular to each other.

1.b

Given two vectors $v_1 = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$ and $v_2 = \begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T$, and a rotational matrix of angle α around the x-axis given by

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\ \alpha \end{bmatrix} \tag{1.6}$$

the dot product of v_1 and v_2 is given by

$$v_1 \cdot v_2 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = x_1x_2 + y_1y_2 + z_1z_2 \tag{1.7}$$

By applying the rotation in (1.14) to v_1, we arrive at

$$v_1' = \begin{bmatrix} x_1' \\ y_1' \\ z_1' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\ \alpha \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1\cos\alpha - z_1\sin\alpha \\ y_1\sin\alpha + z_1\cos\alpha \end{bmatrix} \tag{1.8}$$

Similarly, applying the rotation matrix to v_2 gives

$$v_2' = \begin{bmatrix} x_2' \\ y_2' \\ z_2' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\ \alpha \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2\cos\alpha - z_2\sin\alpha \\ y_2\sin\alpha + z_2\cos\alpha \end{bmatrix} \tag{1.9}$$

The scalar product of the rotated vectors v_1' and v_2' can be found by

$$v_1' \cdot v_2' = x_1x_2 + (y_1\cos\alpha - z_1\sin\alpha)(y_2\cos\alpha - z_2\sin\alpha) + (y_1\sin\alpha + z_1\cos\alpha)(y_2\sin\alpha + z_2\cos\alpha) \tag{1.10}$$

expanding which gives

$$v_1' \cdot v_2' = x_1x_2 + y_1y_2\cos^2\alpha + z_1z_2\sin^2\alpha - y_1z_2\sin\alpha\cos\alpha - y_2z_1\sin\alpha\cos\alpha + y_1y_2sin^2\alpha + z_1z_2\cos^2\alpha + y_1z_2\sin\alpha\cos\alpha + z_1y_2\cos\alpha\sin\alpha \tag{1.11}$$

removing terms that cancel each other out gives

$$v_1' \cdot v_2' = x_1x_2 + y_1y_2(\cos^2\alpha + sin^2\alpha) + z_1z_2(\sin^2\alpha + \cos^2\alpha) \tag{1.12}$$

as $\left(cos^2\alpha + sin^2\alpha\right) = 1$, this results in

$$v_1' \cdot v_2' = x_1x_2 + y_1y_2 + z_1z_2 \tag{1.13}$$

which is same as (1.7), thus this proves that the scalar product of the vectors is invariant to rotations when the same rotation is applied to them.

1.c

The length of a vector v does not change when a rotation is applied to it. This is because rotational matrices are orthogonal with a determinant of 1. This shows that the dimension of vector v will not change when a rotation is applied to it. To prove this mathematically, given a vector $v = \begin{bmatrix} x & y & z \end{bmatrix}^T$ and a rotational matrix of angle α around the x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\ \alpha \end{bmatrix} \tag{1.14}$$

The rotation of vector v around the x-axis is given by

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\ \alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y\cos\alpha - z\sin\alpha \\ y\sin\alpha + z\cos\alpha \end{bmatrix} \tag{1.15}$$

The length of the transformed vector is then

$$|v'| = \sqrt{x^2 + (y\cos\alpha - z\sin\alpha)^2 + (y\sin\alpha + z\cos\alpha)^2} \tag{1.16}$$

expanding (1.16) gives

$$|v'| = \sqrt{\begin{aligned} x^2 + y^2\cos^2\alpha - 2yz\cos\alpha\sin\alpha + z^2 sin^2\alpha + \\ y^2 sin^2\alpha + 2yz\sin\alpha\cos\alpha + z^2 cos^2\alpha \end{aligned}} \tag{1.17}$$

$$|v'| = \sqrt{x^2 + y^2\left(cos^2\alpha + sin^2\alpha\right) + z^2\left(sin^2\alpha + cos^2\alpha\right)} \tag{1.18}$$

as $\left(cos^2\alpha + sin^2\alpha\right) = 1$

$$|v'| = \sqrt{x^2 + y^2 + z^2} \tag{1.19}$$

Hence, the length of vector v stays the same when a rotation is applied to it.

$$||Rv|| = |v| \tag{1.20}$$

1.d

Given two points P_1 and P_2 which have coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively. The distance between the two points is given by

$$||P_1 - P_2|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{1.21}$$

Applying the rotational matrix $R_x(\alpha)$ from (1.14) to P_1 and P_2 yields

$$R_{P_1} = \begin{bmatrix} x_1 \\ y_1\cos\alpha - z_1\sin\alpha \\ y_1\sin\alpha + z_1\cos\alpha \end{bmatrix},\ R_{P_2} = \begin{bmatrix} x_2 \\ y_2\cos\alpha - z_2\sin\alpha \\ y_2\sin\alpha + z_2\cos\alpha \end{bmatrix} \tag{1.22}$$

The distance between the transformed points is then given by

$$||R_{P_1} - R_{P_2}|| = \sqrt{\begin{aligned} (x_2 - x_1)^2 + (y_2\cos\alpha - z_2\sin\alpha - (y_1\cos\alpha - z_1\sin\alpha))^2 + \\ (y_2\sin\alpha + z_2\cos\alpha - (y_1\sin\alpha + z_1\cos\alpha))^2 \end{aligned}} \tag{1.23}$$

$$||R_{P_1} - R_{P_2}|| = \sqrt{\begin{aligned} (x_2 - x_1)^2 + ((y_2 - y_1)\cos\alpha - (z_2 - z_1)\sin\alpha)^2 + \\ ((y_2 - y_1)\sin\alpha + (z_2 - z_1)\cos\alpha)^2 \end{aligned}} \tag{1.24}$$

let $a = (y_2 - y_1)$ and $b = (z_2 - z_1)$

$$||R_{P_1} - R_{P_2}|| = \sqrt{(x_2 - x_1)^2 + (a\cos\alpha - b\sin\alpha)^2 + (a\sin\alpha + b\cos\alpha)^2} \tag{1.25}$$

expanding out gives

$$||R_{P_1} - R_{P_2}|| = \sqrt{\begin{aligned}(x_2 - x_1)^2 + a^2\cos^2\alpha + b^2\sin^2\alpha - 2ab\sin\alpha\cos\alpha + \\ a^2\sin^2\alpha + b^2\cos^2\alpha + 2ab\sin\alpha\cos\alpha\end{aligned}} \tag{1.26}$$

removing out the cancelled-out terms and rearranging the equations gives

$$||R_{P_1} - R_{P_2}|| = \sqrt{(x_2 - x_1)^2 + a^2(cos^2\alpha + sin^2\alpha) + b^2(cos^2\alpha + sin^2\alpha)} \tag{1.27}$$

reverting a and b results in

$$||R_{P_1} - R_{P_2}|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{1.28}$$

which is same as (1.21). Hence, the distance between 2 points does not depend on the reference frame which they are defined.

Question 2

2.a

A succession Y-Z-Y extrinsic Euler rotation is given below

$$R_y(\gamma)R_z(\beta)R_y(\alpha) = \begin{bmatrix} cos(\gamma) & 0 & sin(\gamma) \\ 0 & 1 & 0 \\ -sin(\gamma) & 0 & cos(\gamma) \end{bmatrix} \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{1.29}$$

A succession X-Y-Z intrinsic Tait-Bryan rotation is given below

$$R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.30}$$

When we set the angle of β as 90°, the X-Y-Z Tait-Bryan rotation becomes

$$R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.31}$$

We then multiply out the three matrices to obtain

$$R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha)\cos(\gamma) + \cos(\alpha)\sin(\gamma) & -\sin(\alpha)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & 0 \\ -\cos(\alpha)\cos(\gamma) + \sin(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\gamma) + \sin(\alpha)\cos(\gamma) & 0 \end{bmatrix} \tag{1.32}$$

As we know from the trigonometry identity $\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b)$ and $\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$, we can obtain the final equation for the X-Y-Z rotation as

$$R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{bmatrix} \tag{1.33}$$

From (1.33) it can be seen that changing the values of α and γ will produce the same effect, hence one of the degrees of freedom is lost. This is because the Y axis is aligned with the Z axis, so any change in value of α and γ will only cause a rotation about the Z axis.

In robotics, a gimbal lock is a type of singularity where the robot arm loses one or more degrees of freedom. An example of a gimbal lock in a 6 degree-of-freedom (dof) robotic arm is known as a wrist flip, where the trajectory of the robot causes the joints 4 and 6 to line up. This causes the second wrist axis of the robot arm to flip 180°immediately in order to maintain the desired orientation of the end effector. The other type of singularity is known as a shoulder singularity, which occurs when the centre of the robot's wrist aligns with the axis of joint 1, which causes joints 1 and 4 to flip 180°immediately. The third type of singularity is known as the elbow singularity, where the robot arm is fully stretched, causing the elbow to lock in position.

When a robot arm is controlled in Cartesian mode and passes near a singularity, the joints of the robot may suddenly move at a very high velocity. Hence, gimbal locks in robot arms should be avoided to prevent unpredictable movement of the arm. Gimbal locks can be avoided by taking advantage of the redundant degree of freedom. For example, if 6 degrees of freedom is not required to position and orient the end effector, the extra degree of freedom may allow alternative ways to reach the same position and orientation of the end effector, thus avoiding moves that may cause singularities. Furthermore, as gimbal locks are caused by the representation of orientation with Euler angles, it may be avoided

2.b

Given a quaternion of the form

$$q = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k} \tag{1.34}$$

For a rotation with axis u and angle θ, the equivalent quaternion is given by

$$q = e^{\frac{\theta}{2}(u_x i + u_y j + u_z k)} = \cos\frac{\theta}{2} + (u_x i + u_y j + u_z k)\sin\frac{\theta}{2} \tag{1.35}$$

From (1.34) and (1.35), we can deduce the following

$$\cos\frac{\theta}{2} = q_w,\ u_x\sin\frac{\theta}{2} = q_x,\ u_y\sin\frac{\theta}{2} = q_y,\ u_z\sin\frac{\theta}{2} = q_z, \tag{1.36}$$

By using the trigonometric relation $\cos x = 2\cos^2\frac{x}{2} - 1$ and $\sin x = 2\sin\frac{x}{2}\cos\frac{x}{2}$, the above relations can be further written as

$$\cos\theta = 2{q_w}^2 - 1,\ u_x \sin\theta = 2q_x q_w,\ u_y \sin\theta = 2q_y q_w,\ u_z \sin\theta = 2q_z q_w, \tag{1.37}$$

Furthermore, the followings can be found

$$u_x u_y(1\ -\cos\theta) = 2q_x q_y,\ u_x u_z(1\ -\cos\theta) = 2q_x q_z,\ u_y u_z(1\ -\cos\theta) = 2q_y q_z, \tag{1.38}$$

and finally the followings

$${u_x}^2(1\ -\cos\theta) = 2{q_x}^2,\ {u_y}^2(1\ -\cos\theta) = 2{q_y}^2,\ {u_z}^2(1\ -\cos\theta) = 2{q_z}^2, \tag{1.39}$$

Given the axis-angle to rotation matrix

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix} \tag{1.40}$$

we can substitute the relations found in (1.36)-(1.39) into 1.40. Hence, the Quaternion to rotation matrix is

$$R = \begin{bmatrix} 1-2({q_j}^2+{q_k}^2) & 2(q_i q_j - q_k q_r) & 2(q_i q_k + q_j q_r) \\ 2(q_i q_j + q_k q_r) & 1-2({q_i}^2+{q_k}^2) & 2(q_j q_k - q_j q_r) \\ 2(q_i q_k - q_j q_r) & 2(q_j q_k + q_i q_r) & 1-2({q_i}^2+{q_j}^2) \end{bmatrix} \tag{1.41}$$

2.c

In a Nano-robots with very limited memory storage, Euler angle representation should be used. In a Nano-robots with very limited computational power, Rotational matrix shouuld be used. In an Iphone navigation system as well as a robotic arm with 6 DOF, quaternion representation should be used.

Question 3

3.a

Matrix decomposition factorises a matrix into a product of matrices, which changes the basis sets and expose the structure more easily. Matrix decompositions are useful for solving linear systems in a computationally efficient manner.

3.b

Singular Value Decomposition (SVD) is a factorisation of a real or complex matrix, it generalises the eigendecomposition to any $m \times n$ matrix, which represent overdetermined and underdetermined linear systems. Every matrix $\boldsymbol{A}$ can be expressed as

$$A = U\Sigma V^T \tag{1.42}$$

where U is an orthonormal $m \times m$ matrix, Σ is an $m \times n$, real and non-negative diagonal matrix, and V is an orthonormal $n \times n$ matrix. As SVD uses orthonormal matrices and is thus better behaved.

Question 4

4.a

(Code in cw1/cw1q4 srv)

4.b

To create a service that converts a quaternion representation to an euler angle representation, we first import "quat2zyx", "quat2zyxResponse" as well as "quat2zyxRequest" from the cw1q4_srv folder.

Listing 1: Import Service classes

```
from cw1q4\_srv.srv import quat2zyx, quat2zyxResponse, ...
    quat2zyxRequest
```

We then modified the "convert_quat2zyx(request)" function to convert a quaternion representation to an Euler angle representation as shown below.

Listing 2: convert_quat2zyx(request)

```
def convert_quat2zyx(request):
    qx = request.q.x
    qy = request.q.y
    qz = request.q.z
    qw = request.q.w

    response = quat2zyxResponse()
```

We first defined the request which contains the quaternians qx, qy, qz and qw, and we defined the response to be "quat2zyxResponse()". The Euler angles can be obtained from the quaternions as shown below [1]

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} arctan\frac{2(q_0q_1+q_2q_3}{1-2({q_1}^2+{q_2}^2} \\ arcsin(2(q_0q_2-q_3q_1)) \\ arctan\frac{2(q_0q_3+q_1q_2}{1-2({q_2}^2+{q_3}^2} \end{bmatrix} \tag{1.43}$$

where q_0, q_1, q_2, q_3 are q_w, q_x, q_y and q_z, respectively. Hence, we can implement (1.43) to the code directly as shown below

Listing 3: convert_quat2zyx(request) continue

```
response.x = np.arctan2(2*(qw*qx + qy*qz),1-2*(qx*qx+qy*qy)

y1 = 2*(qw*qy - qz*qx)
y1 = 1 if y1 > 1 else y1
y1 = -1 if y1< -1 else y1
response.y = np.arcsin(y1)

response.z = np.arctan2(2*(qw*qz + qx*qy), 1-2*(qy*qy+qz*qz)
```

The function np.arctan2 was used as arctan would only produce results between $-\pi/2$ and $\pi/2$. The `if` statement is implemented such that the angle y will be 90°if the input is out of range, as sin can only have values between -1 and 1. The response function then stores the requested Euler angles.

4.c

We created a service that converts a quaternion representation to a rodrigues representation. We first imported the "quat2rodrigues", "quat2rodriguesResponse" as well as the "quat2rodriguesRequest" from the cw1q4_srv folder.

Listing 4: Import Service classes

```
from cw1q4\_srv.srv import quat2rodrigues, quat2rodriguesResponse,
quat2rodriguesRequest
```

We then modified the "convert_quat2rodrigues(request)" function to convert a quaternion representation to a rodrigues representation as shown below.

Listing 5: convert_quat2rodrigues(request)

```
def convert_quat2rodrigues(request):
    qx = request.q.x
    qy = request.q.y
    qz = request.q.z
    qw = request.q.w

    response = quat2rodriguesResponse()
```

Similar to the previous function, We first defined the request which contains the quaternians qx, qy, qz and qw, and we defined the response to be "quat2rodriguesResponse()".

The conversion from quaternions to rodrigues is given by

$$\vec{b} = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix} \tag{1.44}$$

where b_x, b_y and b_z are the rodrigues vector given by

$$b_x = \tan(\frac{1}{2}\theta)s_x,\ b_y = \tan(\frac{1}{2}\theta)s_y,\ b_z = \tan(\frac{1}{2}\theta)s_z \tag{1.45}$$

where $\vec{s}$ are the rodrigues parameters deduced by a unit vector around which the rotation is performed [2], and were previously defined as u_x, u_y and u_z in (1.36).

```
theta = np.arccos(qw)*2
sx = qx/np.sin(theta/2)
sy = qy/np.sin(theta/2)
sz = qz/np.sin(theta/2)
response.x = np.tan(thetaa/2)*sx
response.y = np.tan(thetaa/2)*sy
response.z = np.tan(thetaa/2)*sz
```

Forward Kinematics

Question 5

1.a

The frames of the KUKA Youbot is shown in Fig. .1, which starts from the base link up to link 5. The DH parameter of the robot is shown in Table 1. The base link is located at the centre with frame z_0 pointing up, x_0 pointing to the left and y_0 pointing out of diagram. Link 1 is placed at 147 mm above the base link along the direction of z_0, hence has a link offset of 0.147 m. As link 1 is rotated 90°about the x_0 and z_0, it has a link twist and joint angle offset of 90°/. Link 2 is placed at 155 mm away from link 1 along x_2, hence has a link length of 155 mm. As link 2 is rotated 90°about the z axis, there is a 90°offset in the joint angle. Link 3 is placed at 135 mm away from link 2 along x_3, so it has a link length of 135 mm. As the orientation of link 3 has not changed, the link twist and joint angle offset is 0. Link 4 is placed at 0 mm away from link 3, and has a link twist and joint angle offset of -90°. Link 4 is placed as such as it only needs to provide a rotation for link 5. Finally, link 5 is placed at 128 mm away from link 4, which is the distance from A4 on the diagram up to the tip of the end effector. The orientation of link 5 has not changed, so there is no link twist or link offset reqired.

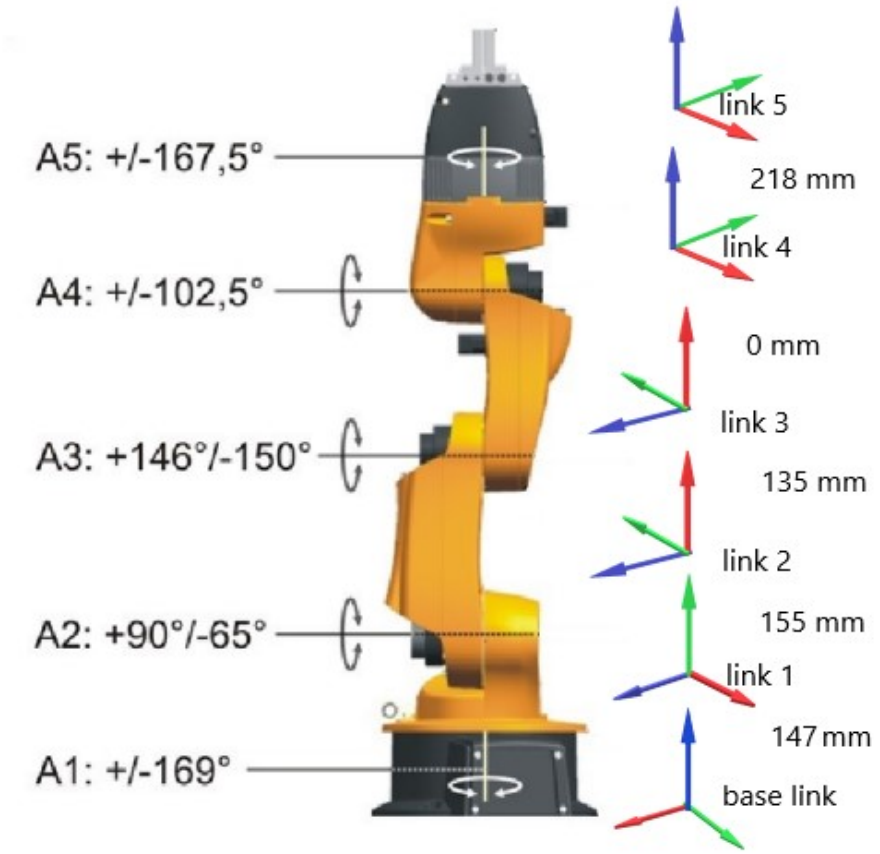

Figure .1: Frames of the robot joints

Table 1: DH Parameters

i	a (m)	alpha (rad)	d (m)	theta (rad)
1	0	pi/2	0.147	$\theta_1 + pi/2$
2	0.155	0	0	$\theta_2 + pi/2$
3	0.135	0	0	θ_3
4	0	-pi/2	0	$\theta_4 - pi/2$
5	0	0	0.128	θ_5

1.b

The code for cw1q5b.node includes 5 parts, which are the "youbot_dh_parameters" dictionary, "standard_dh()" function, "forward_kinematics()" function, "fkine_wrapper()" function as well as the subscriber.

We firstly added the dh parameters from Table 1 to the dictionaary of the code. We then defined the "standard_dh()" function, which outputs the transformation matrix given by

$$ {}^{i-1}\boldsymbol{T}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

which describes a coordinate transformation from the concurrent coordinate system i to the previous coordinate system $i-1$. Next, the "forward_kinematics()" function consists of a for loop which multiplies using np.dot() the pose of frame_up_to_joint with respect to the base of the robot. Then, the "fkine_wrapper()" function is responsible for calling the "forward_kinematics()" functions, which will return 5 transformation matrices $0T1$, $0T2$, $0T3$, $0T4$ and $0T5$. We then placed the translation and rotation of x, y and y in a list so that it can be broadcasted using tf2. Finally, we initialised the subscriber to the topic that publishes the joint angles, and configured it to have "fkine_wrapper()" as call back and pass the broadcaster as an additional argument to the callback.

1.c

From on the URDF file, the link link, link offset as well as the joint offset can be found. Where the link offset and link length is given in the ¡origin¿ xyz and the joint offset is given in the rpy as radians. From the observation of the URDF file, the new DH parameter is given in Table 2 and the link offset is given as

$$ [170pi/180, 65pi/180, -146pi/180, 102.5pi/180, 167.5pi/180] \tag{2.2} $$

As a result, using the new DH parameters derived, the frames are now stuck with the joint of the robot in Rviz.

1.d

(code implemented)

Table 2: DH Parametersurdf

i	a (m)	alpha (rad)	d (m)	theta (rad)
1	-0.024-0.033	pi/2 + pi	0.096 + 0.19	0
2	0.155	0	0	0
3	0.135	0	0	pi
4	0	pi/2 + pi	0	0
5	-0.002	0	0.130	0

References

[1] J. luis Blanco, "A tutorial on se(3) transformation parameterizations and on-manifold optimization," 2013.

[2] J. S. Dai, "Euler–rodrigues formula variations, quaternion conjugation and intrinsic connections," *Mechanism and Machine Theory*, vol. 92, pp. 144–152, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094114X15000415