Robotic Systems Engineering
Coursework 1: Linear Algebra and Forward Kinematics

Harry Yao
zcabry0@ucl.ac.uk

November 8, 2024

Section 1

1. a.

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Because R is a rotation matrix, so R is a orthogonal

Therefore det(R) = 1, which means $R^T R = I$

This orthogonality implies that the columns of R are orthonormal vectors, meaning each column has a unit norm and each pair of columns is orthogonal to each

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}.$$

$$r_1 = \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}, \quad r_2 = \begin{bmatrix} r_{12} \\ r_{22} \\ r_{32} \end{bmatrix}, \quad r_3 = \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}$$

$$||r_i|| = \sqrt{r_{i1}^2 + r_{i2}^2 + r_{i3}^2}$$

Since R is orthogonal, each column r_i (for $i = 1, 2, 3$) satisfies $||r_i|| = 1$, meaning each column vector has a unit norm.

Therefore $||r_i|| \leq 1$ where $i = 1, 2, \ldots, 9$

b.

$$R_{k,\theta} = I + \sin(\theta)K + (1 - \cos(\theta))K^2$$

where I is the identity matrix, and K is the skew-symmetric matrix of k, given by:

$$K = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}$$

Substitute $-k$ and $-\theta$ into the formula:

$$R_{-k,-\theta} = I + \sin(-\theta)(-K) + (1 - \cos(-\theta))(-K)^2$$

Using the identities $\sin(-\theta) = -\sin(\theta)$ and $\cos(-\theta) = \cos(\theta)$, we get:

$$R_{-k,-\theta} = I - \sin(\theta)K + (1 - \cos(\theta))K^2$$

Therefore:

$$R_{-k,-\theta} = R_{k,\theta}$$

This proves that rotating by θ around axis k is equivalent to rotating by $-\theta$ around axis $-k$.

c. The rotation matrix ${}^aR_b$ represents the orientation of frame b relative to frame a.

- The first row of ${}^aR_b$ represents the projection of the x-axis of frame b onto the x-, y-, and z-axes of frame a.
- The second row represents the projection of the y-axis of frame b onto the x-, y-, and z-axes of frame a.
- The third row represents the projection of the z-axis of frame b onto the x-, y-, and z-axes of frame a.

In summary, each row of ${}^aR_b$ gives the components of one of frame b's axes in the coordinate system of frame a, which describes the relative orientation between the two coordinate frames.

d. Solution of identify the relationship between the axis/angle of rotation and the eigenvector/eigenvalue of a rotation matrix:

- Eigenvector and Rotation Axis:
  For a 3D rotation matrix R, there exists an eigenvector v with an eigenvalue of 1, satisfying the equation:
  $$Rv = v$$
  This eigenvector v represents the axis of rotation, as any vector along this axis remains unchanged in direction by the rotation.
- Eigenvalues and Angle of Rotation:
  In a 3D rotation, the rotation matrix R has two additional complex eigenvalues that are complex conjugates, given by:
  $$\lambda = e^{i\theta} \quad \text{and} \quad \lambda = e^{-i\theta}$$

where θ is the angle of rotation around the axis represented by the eigenvector with eigenvalue 1.
The real part of these complex eigenvalues corresponds to $\cos(\theta)$:

$$\text{Re}(\lambda) = \cos(\theta)$$

This relationship allows us to determine the angle of rotation θ from the real part of the complex eigenvalues.

- **Summary:**
  In summary:
  - The eigenvector with eigenvalue 1 represents the axis of rotation.
  - The angle of rotation θ is related to the real part of the complex eigenvalues by $\cos(\theta) = \text{Re}(\lambda)$.

Section 2

2. a. Example of Gimbal Lock under Y-Z-Y rotation - Rotation about the Y-axis by angle α:

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

- Rotation about the Z-axis by angle β:

$$R_z(\beta) = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Another rotation about the Y-axis by angle γ:

$$R_y(\gamma) = \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) \\ 0 & 1 & 0 \\ -\sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix}$$

Gimbal lock occurs when $\beta = \pm 90°$, because at this point, the first and third rotation axes (both Y-axes) align. This results in the loss of one degree of freedom, making it impossible to control rotation independently in all three directions.

Example of Gimbal Lock under X-Y-Z rotation
- Rotation about the x-axis by angle α:

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

- Rotation about the y-axis by angle β:

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

- Rotation about the z-axis by angle γ:

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Gimbal lock occurs when $\beta = \pm 90°$, causing the x-axis and z-axis to align, reducing the effective degrees of freedom.

Gimbal lock basically caused by losing one degree of freedom for the robot's ability to rotate freely in all directions.

Use quaternions instead of traditional Euler angles to represent rotation can avoid gimbal

b. - Quaternion Representation:
A quaternion q is typically represented as:

$$q = w + xi + yj + zk$$

where w is the scalar part, and x, y, and z are the vector components of the quaternion.

- Components of the Quaternion:
Let $q = (w, x, y, z)$. Then:

- w: The real part of the quaternion.
- x, y, z: The imaginary parts corresponding to the rotation axis.

- Rotation Matrix from Quaternion:
The rotation matrix R that corresponds to quaternion $q = (w, x, y, z)$ can be constructed as follows:

$$R = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

- Derivation Steps:
This matrix is derived based on the relationship between the quaternion components and the axis-angle representation of rotations. The entries in the matrix are computed by applying the quaternion rotation to each basis vector.

- Conclusion: Given a quaternion $q = (w, x, y, z)$, the rotation matrix R can be computed using the above formula. This matrix allows you to apply the same rotation in 3D space as the quaternion.

c. 1. Nano-robot with very limited memory storage: **Axis-Angle Representation** or **Compact Quaternion Representation**
2. Nano-robot with very limited computational power: **Rotation Matrix**
3. iPhone navigation system: **Quaternion**
4. Robotic arm with 6 DOF: **Denavit-Hartenberg (D-H) Parameters**

Section 3

3. a. - Quaternion Definition:
A rotation quaternion $q = (w, x, y, z)$ can be represented as:

$$q = w + xi + yj + zk$$

where w is the scalar part, and x, y, z are the vector components.

- Rotation Matrix of a Quaternion:
The rotation matrix R corresponding to quaternion $q = (w, x, y, z)$ is:

$$R = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

- Rotation Matrix of the Negated Quaternion:
For the negated quaternion $-q = (-w, -x, -y, -z)$, substituting $-w, -x, -y, -z$ into the rotation matrix formula yields:

$$R = \begin{bmatrix} 1 - 2(-y)^2 - 2(-z)^2 & 2(-x)(-y) - 2(-w)(-z) & 2(-x)(-z) + 2(-w)(-y) \\ 2(-x)(-y) + 2(-w)(-z) & 1 - 2(-x)^2 - 2(-z)^2 & 2(-y)(-z) - 2(-w)(-x) \\ 2(-x)(-z) - 2(-w)(-y) & 2(-y)(-z) + 2(-w)(-x) & 1 - 2(-x)^2 - 2(-y)^2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

Therefore rotation quaternion q and -q are equivalent

b. Two rotation matrices R_a and R_b are commutative if:

$$R_a R_b = R_b R_a.$$

This commutativity occurs under specific conditions:

1. **Rotations About the Same Axis**: If R_a and R_b represent rotations around the same axis, they commute regardless of the rotation angles. For example, two rotations about the z-axis:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_z(\beta) = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

will commute because

$$R_z(\alpha)R_z(\beta) = R_z(\alpha + \beta) = R_z(\beta)R_z(\alpha).$$

2. **One Rotation Angle is 0° or 180°**: If one of the rotation angles in either R_a or R_b is $0°$ or $180°$, the matrices will commute. For example, a $180°$ rotation around the x-axis:
$$R_x(180°) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix},$$
will commute with any rotation matrix around the y-axis, as it acts as a reflection in the plane orthogonal to x-axis.
3. **Rotations in the Same Plane (Special Cases)**: In some cases, rotations within the same plane may commute, particularly if they are aligned in a way that the order does not affect the outcome.

These conditions simplify the analysis of combined rotations by ensuring that the order of operations does not impact the final orientation.

Section 4

4. a. Check out the code

b. Function: convert_quat2zyx:
This function takes a quaternion as input and returns the Z-Y-X Euler angles.

```
1
2     def convert_quat2zyx(request):
3         # Extract quaternion components from the request
4         qx = request.q.x
5         qy = request.q.y
6         qz = request.q.z
7         qw = request.q.w
```

Listing 1: convert_quat2zyx function

- ‘qx‘, ‘qy‘, ‘qz‘, and ‘qw‘ represent the four components of the quaternion, obtained from the ‘request‘ object. The quaternion is represented by these four values: $q = (qw, qx, qy, qz)$. - Here, ‘request.q.x‘, ‘request.q.y‘, ‘request.q.z‘, and ‘request.q.w‘ are the attributes of the quaternion passed to the function.

```
1
2         # Calculate the Z-Y-X Euler angles
3         x = math.atan2(2 * (qw * qx + qy * qz), 1 - 2*(qx**2
              + qy**2))
4         y = math.asin(2 * (qw * qy - qz * qx))
5         z = math.atan2(2 * (qw * qz + qx * qy), 1 - 2*(qy**2
              + qz**2))
```

Listing 2: Euler angle calculation

- ‘x‘, ‘y‘, and ‘z‘ are the calculated Euler angles in the Z-Y-X rotation order. - ‘x‘ represents the rotation around the X-axis. - ‘y‘ represents the rotation around the Y-axis. - ‘z‘ represents the rotation around the Z-axis. - These angles are computed using trigonometric functions ‘atan2‘ and ‘asin‘ with quaternion components.

```
        response = quat2zyxResponse()
        response.x.data = x
        response.y.data = y
        response.z.data = z
        return response
```

Listing 3: Returning response with Euler angles

- ‘response‘ is created as an instance of ‘quat2zyxResponse‘ and is used to store the calculated Euler angles. - ‘response.x.data‘ stores the X-axis rotation angle. - ‘response.y.data‘ stores the Y-axis rotation angle. - ‘response.z.data‘ stores the Z-axis rotation angle. - The function returns this ‘response‘ object.

c. Function: convert_quat2rodrigues

This function converts a quaternion to Rodrigues representation, which is an axis-angle representation.

```
    def convert_quat2rodrigues(request):
        # Extract quaternion components from the request
        qx = request.q.x
        qy = request.q.y
        qz = request.q.z
        qw = request.q.w
```

Listing 4: convert_quat2rodrigues function

- Similar to the previous function, ‘qx‘, ‘qy‘, ‘qz‘, and ‘qw‘ are the quaternion components extracted from the ‘request‘.

```
        # Calculate the rotation angle and Rodrigues vector
           components
        theta = 2 * math.acos(qw)
        if theta == 0:
            kx, ky, kz = 0, 0, 0
        else:
            sin_half_theta = math.sqrt(1 - qw**2)
            rx = qx / sin_half_theta
            ry = qy / sin_half_theta
            rz = qz / sin_half_theta
            kx = theta * rx
            ky = theta * ry
            kz = theta * rz
```

Listing 5: Rodrigues vector calculation

- ‘theta‘ represents the rotation angle around the axis, calculated as $\theta = 2 \cdot \arccos(qw)$. - ‘kx‘, ‘ky‘, and ‘kz‘ are the components of the Rodrigues vector, which describe the axis and magnitude of the rotation. - If ‘theta‘ is zero, it means there is no rotation, and all Rodrigues vector components are set to zero. - Otherwise, ‘rx‘, ‘ry‘, and ‘rz‘ represent the normalized rotation axis, and ‘kx‘, ‘ky‘, and ‘kz‘ are scaled by ‘theta‘.

```
        response = quat2rodriguesResponse()
        response.x.data = kx
        response.y.data = ky
        response.z.data = kz
        return response
```

Listing 6: Returning response with Rodrigues representation

- ‘response‘ is created as an instance of ‘quat2rodriguesResponse‘ and stores the Rodrigues vector components. - ‘response.x.data‘ stores the X component of the Rodrigues vector. - ‘response.y.data‘ stores the Y component of the Rodrigues vector. - ‘response.z.data‘ stores the Z component of the Rodrigues vector. - The function returns this ‘response‘ object containing the Rodrigues representation.

Section 5

5. a. Each DH parameter is derived as follows:

Link Length a

This parameter a_i represents the offset along the x-axis between consecutive joints:

- $a_1 = 0$: No offset for the base joint.
- $a_2 = 0.302 - 0.437 = -0.155\,\text{m}$: Horizontal offset from joint 1 to joint 2.
- $a_3 = 0.437 - 0.302 = 0.135\,\text{m}$: Offset from joint 2 to joint 3.
- $a_4 = 0$: No offset between joint 3 and joint 4.
- $a_5 = 0$: No offset between joint 4 and joint 5.

Link Twist α

The twist α_i is the angle between the z-axes of two consecutive joints:

- $\alpha_1 = -\pi/2$: The axis of joint 2 is perpendicular to joint 1.
- $\alpha_2 = 0$: The axis of joint 3 is aligned with joint 2.
- $\alpha_3 = 0$: The axis of joint 4 is aligned with joint 3.
- $\alpha_4 = -\pi/2$: The axis of joint 5 is perpendicular to joint 4.
- $\alpha_5 = 0$: The end-effector is aligned along the same axis as joint 5.

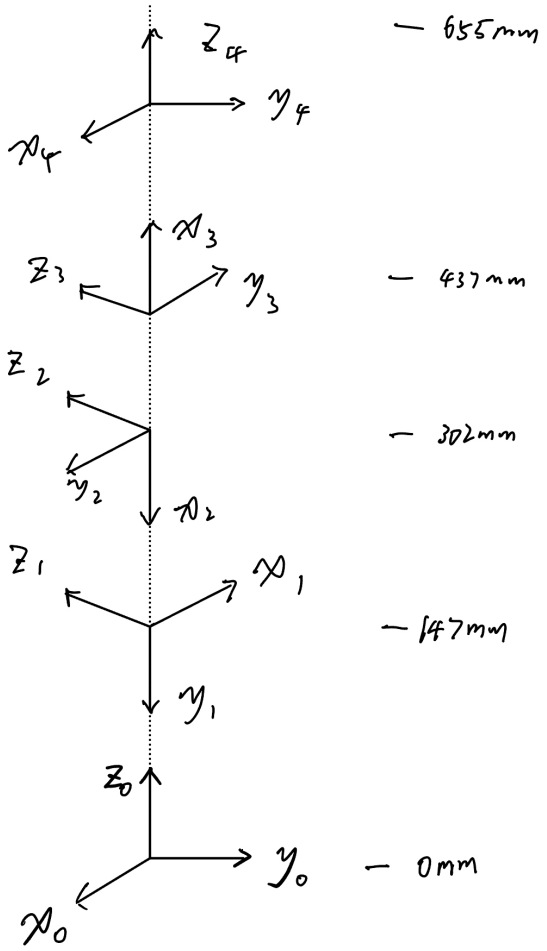

Figure 1: Frame Image of the Robot Manipulator

Link Offset d

The offset d_i represents the distance along the z-axis between consecutive joints:

- $d_1 = 0.147 - 0 = 0.147\,\text{m}$: Vertical offset between the base and joint 1, which corresponds to the physical height of $147\,\text{mm}$.
- $d_2 = 0$: No vertical offset between joint 1 and joint 2.
- $d_3 = 0$: No vertical offset between joint 2 and joint 3.
- $d_4 = 0$: No vertical offset between joint 3 and joint 4.
- $d_5 = 0.655 - 0.437 = 0.218\,\text{m}$: Offset corresponding to the end-effector's height.

Joint Angle θ

The joint angle θ_i determines the rotation of each joint:

- $\theta_1 = 0$: The base is aligned with the x-axis.

- $\theta_2 = 12.5^\circ = 12.5 \cdot \frac{\pi}{180} \approx 0.218$ rad: The angle of joint 2.
- $\theta_3 = -\frac{\pi}{6}$ rad $= -30^\circ$: Downward tilt at joint 3.
- $\theta_4 = -\frac{\pi}{2}$ rad $= -90^\circ$: Perpendicular orientation for joint 4.
- $\theta_5 = 0$: No rotation for the end-effector.

Joint	a (m)	α (rad)	d (m)	θ (rad)
Joint 1	0.000	-1.5708	0.147	0.0000
Joint 2	-0.155	0.0000	0.000	0.2182
Joint 3	0.135	0.0000	0.000	-0.5236
Joint 4	0.000	-1.5708	0.000	-1.5708
Joint 5	0.000	0.0000	0.218	0.0000

Table 1: DH Parameters for Kuka youBot Manipulator

b. Function: `standard_dh`

```
        A = np.array([
            [np.cos(theta), -np.sin(theta) * np.cos(alpha),
               np.sin(theta) * np.sin(alpha), a * np.cos(
               theta)],
            [np.sin(theta), np.cos(theta) * np.cos(alpha), -
               np.cos(theta) * np.sin(alpha), a * np.sin(
               theta)],
            [0, np.sin(alpha), np.cos(alpha), d],
            [0, 0, 0, 1]
        ])-
```

Listing 7: standard_dh function

This code computes the standard DH transformation matrix A, which describes the rotation and translation relationships between two adjacent joints. The matrix A is structured as follows:

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Each component of this matrix corresponds to: - $\cos(\theta)$ and $\sin(\theta)$: representing rotation around the z-axis. - $a\cos(\theta)$ and $a\sin(\theta)$: representing translation along the x-axis. - $\sin(\alpha)$ and $\cos(\alpha)$: representing rotation around the x-axis. - d: representing translation along the z-axis.

This matrix is used to transform the local coordinate frame i to the previous coordinate frame $i-1$.

Function: `forward_kinematics`

```
        for i in range(up_to_joint):

            a = dh_dict['a'][i]
            alpha = dh_dict['alpha'][i]
            d = dh_dict['d'][i]
            theta = joints_readings[i] +  dh_dict['theta'][i
                ]

            T_i = standard_dh(a, alpha, d, theta)

            T = np.dot(T, T_i)
```

Listing 8: forward_kinematics function

This code segment computes the forward kinematics by iteratively applying the DH transformations up to the specified joint index, `up_to_joint`.

1. `for i in range(up_to_joint)`: The loop iterates through each joint up to the `up_to_joint` index.

2. The DH parameters (`a`, `alpha`, `d`, and `theta`) for each joint are extracted from the dictionary `dh_dict`.

3. `theta = joints_readings[i] + dh_dict['theta'][i]`: Adjusts the angle `theta` by adding the current joint reading to the static DH parameter value.

4. `T_i = standard_dh(a, alpha, d, theta)`: Calls the `standard_dh` function to generate the 4x4 transformation matrix for the current joint.

5. `T = np.dot(T, T_i)`: Multiplies the accumulated transformation matrix `T` by the current transformation matrix `T_i`. This cumulative multiplication yields the transformation matrix representing the pose of the specified joint relative to the base.

Function: `fkine_wrapper`

```
        transform = TransformStamped()

        for i in range(5):

            T01 = forward_kinematics(youbot_dh_parameters,
                list(joint_msg.position), i+1)

            transform.header.stamp = rospy.Time.now()
            transform.header.frame_id = 'base_link'
            transform.child_frame_id = 'A' + str(i+1)

            transform.transform.translation.x = T01[0, 3]
```

```
            transform.transform.translation.y = T01[1, 3]
            transform.transform.translation.z = T01[2, 3]
            transform.transform.rotation = rotmat2q(T01)

            br.sendTransform(transform)
```

Listing 9: fkine_wrapper function

This segment of code publishes the transformations between each joint in the robotic arm based on the computed forward kinematics.

`transform = TransformStamped()`: Initializes a new ROS `TransformStamped` message to hold the transformation data.

`for i in range(5)`: Loops through each of the five joints, calculating and broadcasting the transformation for each joint.

`T01 = forward_kinematics(youbot_dh_parameters, list(joint_msg.posit i+1)`: Calculates the transformation matrix from the base frame to the current joint `i+1` using the forward kinematics function and the current joint angles from `joint_msg.position`.

`transform.header.stamp = rospy.Time.now()`: Sets the timestamp for the transformation to the current ROS time.

`transform.header.frame_id = 'base_link'` and `transform.child_frame_id = 'A' + str(i+1)`: Specifies the parent frame as `base_link` and sets the child frame to the name `A` followed by the joint index (e.g., `A1`, `A2`).

`transform.transform.translation.x = T01[0, 3]` (and similar lines for `y` and `z`): Assigns the translation components of the transformation matrix `T01` to the transform message.

`transform.transform.rotation = rotmat2q(T01)`: Converts the rotation matrix part of `T01` to a quaternion and assigns it to the transform message.

`br.sendTransform(transform)`: Publishes the transformation, making it available for other ROS nodes to access and use.

c. The DH parameters are essential for describing the kinematic configuration of a robotic manipulator. They consist of four parameters for each joint:

- a_{i-1}: Link length, the distance along the X_{i-1} axis from Z_{i-1} to Z_i.
- α_{i-1}: Link twist, the angle between Z_{i-1} and Z_i around the X_{i-1} axis.
- d_i: Link offset, the distance along the Z_i axis from X_{i-1} to X_i.
- θ_i: Joint angle, the angle between X_{i-1} and X_i around the Z_i axis.

1. Convert xacro to URDF:
   The 'youbot_arm_only.urdf.xacro' file is processed using `rosrun xacro xacro` to generate a simplified 'youbot_arm_only.urdf' file. This step transforms the XACRO macros into explicit URDF tags, which helps standardize the file structure and make data extraction more straightforward.

2. Data Parsing:
   With the URDF file available, we iterate through each joint and link definition to collect essential kinematic data. This involves:

- Identifying joint types and names
- Extracting parent and child link relationships
- Obtaining joint origins, including position and orientation
- Retrieving joint axis directions

Data Sources

The primary data source is the URDF (Unified Robot Description Format) file provided, which contains detailed information about the robot's joints and links, including their positions (`xyz`), orientations (`rpy`), and parent-child relationships.

Calculation Process and Formulas

We will calculate the DH parameters for each joint using the information from the URDF file.

Joint 1 *Data from URDF*:

- Origin: $\texttt{xyz} = [0.024,\ 0,\ 0.096]$, $\texttt{rpy} = [0,\ 0,\ 2.9670597283903604]$
- Axis: $[0,\ 0,\ -1]$

Calculations:

1. a_0:

$$\begin{aligned} a_0 &= 0.024\,\text{m (from } x\text{-component of origin)} + 0.009\,\text{m (structural offset)} \\ &= 0.033\,\text{m} \end{aligned}$$

2. α_0:

$$\alpha_0 = -\frac{\pi}{2}\,\text{rad (to align } Z_0 \text{ with } Z_1)$$

3. d_1:

$$\begin{aligned} d_1 &= 0.096\,\text{m (from } z\text{-component of origin)} + 0.051\,\text{m (base height)} \\ &= 0.147\,\text{m} \end{aligned}$$

4. θ_1:

$$\theta_1 = \theta_{1,\text{variable}}$$

5. **Joint Offset for Joint 1**:

$$\theta_{1,\text{offset}} = 2.9670597283903604\,\text{rad} \quad (\approx 170^\circ)$$
(from `rpy` value in URDF)

Joint 2 *Data from URDF*:

- Origin: $\texttt{xyz} = [0.033,\ 0,\ 0.019]$, $\texttt{rpy} = [0,\ -1.1344640137963142,\ 0]$
- Axis: $[0,\ 1,\ 0]$

Calculations:

1. a_1:

$$a_1 = 0.155\,\text{m (link length from specifications)}$$

2. α_1:

$$\alpha_1 = 0\,\text{rad (since } Z_1 \parallel Z_2\text{)}$$

3. d_2:

$$d_2 = 0\,\text{m (axes intersect)}$$

4. θ_2:

$$\theta_2 = \theta_{2,\text{variable}} - \frac{\pi}{2}$$

5. **Joint Offset for Joint 2**:

$$\theta_{2,\text{offset}} = -1.1344640137963142\,\text{rad} \quad (\approx -65^\circ)$$
(from $\texttt{rpy}$ value in URDF)

Joint 3 *Data from URDF*:

- Origin: $\texttt{xyz} = [0,\ 0,\ 0.155]$, $\texttt{rpy} = [0,\ 2.548180707911721,\ 0]$
- Axis: $[0,\ 1,\ 0]$

Calculations:

1. a_2:

$$a_2 = 0.135\,\text{m (link length)}$$

2. α_2:

$$\alpha_2 = 0\,\text{rad}$$

3. d_3:

$$d_3 = 0\,\text{m}$$

4. θ_3:

$$\theta_3 = \theta_{3,\text{variable}}$$

5. **Joint Offset for Joint 3**:

$$\theta_{3,\text{offset}} = 2.548180707911721\,\text{rad} \quad (\approx 146^\circ)$$
(from $\texttt{rpy}$ value in URDF)

Joint 4 *Data from URDF*:

- Origin: $\texttt{xyz} = [0,\ 0,\ 0.135]$, $\texttt{rpy} = [0,\ -1.7889624832941877,\ 0]$
- Axis: $[0,\ 1,\ 0]$

Calculations:

1. a_3:

$$a_3 = 0\,\text{m}$$

2. α_3:

$$\alpha_3 = -\frac{\pi}{2}\,\text{rad}$$

3. d_4:

$$d_4 = 0\,\text{m}$$

4. θ_4:

$$\theta_4 = \theta_{4,\text{variable}} - \frac{\pi}{2}$$

5. **Joint Offset for Joint 4**:

$$\theta_{4,\text{offset}} = -1.7889624832941877\,\text{rad} \quad (\approx -102.5^\circ)$$
(from $\texttt{rpy}$ value in URDF)

Joint 5 *Data from URDF*:

- Origin: $\texttt{xyz} = [-0.002,\ 0,\ 0.130]$, $\texttt{rpy} = [0,\ 0,\ 2.9234264970905017]$
- Axis: $[0,\ 0,\ -1]$

Calculations:

1. a_4:

$$a_4 = -0.002\,\text{m (from } x\text{-component of origin)}$$

2. α_4:

$$\alpha_4 = 0\,\text{rad}$$

3. d_5:

$$\begin{aligned} d_5 &= 0.130\,\text{m (from } z\text{-component of origin)} + 0.088\,\text{m (end-effector length)} \\ &= 0.218\,\text{m} \end{aligned}$$

4. θ_5:

$$\theta_5 = \theta_{5,\text{variable}} - \pi$$

5. **Joint Offset for Joint 5**:

$$\theta_{5,\text{offset}} = 2.9234264970905017\,\text{rad} \quad (\approx 167.5^\circ)$$
(from $\texttt{rpy}$ value in URDF)

Final DH Parameters and Joint Offsets

DH Parameters

$$
\begin{aligned}
\text{youbot\_dh\_parameters} = \{ & \\
& a : [0.033,\ 0.155,\ 0.135,\ 0.0,\ -0.002], \\
& \alpha : \left[-\frac{\pi}{2},\ 0.0,\ 0.0,\ -\frac{\pi}{2},\ 0.0\right], \\
& d : [0.147,\ 0.0,\ 0.0,\ 0.0,\ 0.218], \\
& \theta : [0,\ -\frac{\pi}{2},\ 0,\ -\frac{\pi}{2},\ -\pi] \\
\} &
\end{aligned}
$$

Joint Offsets

$$
\begin{aligned}
\text{youbot\_joint\_offsets} = [& \\
& 2.9670597283903604, \quad \text{(Offset for Joint 1)} \\
& -1.1344640137963142, \quad \text{(Offset for Joint 2)} \\
& 2.548180707911721, \quad \text{(Offset for Joint 3)} \\
& -1.7889624832941877, \quad \text{(Offset for Joint 4)} \\
& 2.9234264970905017 \quad \text{(Offset for Joint 5)} \\
] &
\end{aligned}
$$

Formulas Used

- Link length:

$$a_{i-1} = \text{Distance along } X_{i-1} \text{ from } Z_{i-1} \text{ to } Z_i$$

- Link twist:

$$\alpha_{i-1} = \text{Angle between } Z_{i-1} \text{ and } Z_i \text{ about } X_{i-1}$$

- Link offset:

$$d_i = \text{Distance along } Z_i \text{ from } X_{i-1} \text{ to } X_i$$

- Joint angle:

$$\theta_i = \theta_{i,\text{variable}} + \theta_{i,\text{offset}}$$

Calculation of Joint Offsets

The joint offsets are obtained from the initial rotation angles specified in the URDF file's `rpy` (roll, pitch, yaw) values for each joint. These offsets represent the angle by which each joint is rotated from its zero position.

Joint 1 Offset

$$\begin{aligned}\theta_{1,\text{offset}} &= \text{Yaw angle from URDF} \\ &= 2.9670597283903604\,\text{rad}\end{aligned}$$

Joint 2 Offset

$$\begin{aligned}\theta_{2,\text{offset}} &= \text{Pitch angle from URDF} \\ &= -1.1344640137963142\,\text{rad}\end{aligned}$$

Joint 3 Offset

$$\begin{aligned}\theta_{3,\text{offset}} &= \text{Pitch angle from URDF} \\ &= 2.548180707911721\,\text{rad}\end{aligned}$$

Joint 4 Offset

$$\begin{aligned}\theta_{4,\text{offset}} &= \text{Pitch angle from URDF} \\ &= -1.7889624832941877\,\text{rad}\end{aligned}$$

Joint 5 Offset

$$\begin{aligned}\theta_{5,\text{offset}} &= \text{Yaw angle from URDF} \\ &= 2.9234264970905017\,\text{rad}\end{aligned}$$

These offsets are used to adjust the joint angles in the DH parameters, ensuring that the model accurately reflects the physical configuration of the manipulator.

d. Answer in the code

END OF COURSEWORK