RDC: Assignment 2

Avneesh Mishra *
avneesh.mishra@research.iiit.ac.in

December 2, 2021

Contents

Listings

List of Figures

*Roll No: 2021701032

1 Euler ZYX Convention

Equations for rotation matrices from principal axis rotations

$$\begin{aligned}
\mathbf{R}(\hat{Z},\theta) &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{R}(\hat{Y},\theta) &= \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \\
\mathbf{R}(\hat{X},\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}
\end{aligned} \tag{1.1}$$

1.1 Converting Euler angles to Rotation Matrix

The Euler ZYX rotation is given by (consider $s_\theta = \sin(\theta)$ and $c_\theta = \cos(\theta)$)

$$\begin{aligned}
\mathbf{R}_{ZYX}(\alpha,\beta,\gamma) &= \mathbf{R}(\hat{Z},\alpha)\mathbf{R}(\hat{Y},\beta)\mathbf{R}(\hat{X},\gamma) \\
&= \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \\
&= \begin{bmatrix} c_\alpha c_\beta & -s_\alpha c_\gamma + s_\beta s_\gamma c_\alpha & s_\alpha s_\gamma + s_\beta c_\alpha c_\gamma \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - s_\gamma c_\alpha \\ -s_\beta & s_\gamma c_\beta & c_\beta c_\gamma \end{bmatrix}
\end{aligned} \tag{1.2}$$

A Python function that can do this is written in Appendix A.1 and given with this document.

1.2 Converting Rotation Matrix to Euler angles

The Equation 1.2 has to be reversed. The rotation matrix is given as

$$\mathrm{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Relating the terms from Equation 1.2, we have the following values for

- α being the angle of rotation about Z axis
- β being the angle of rotation about Y axis
- γ being the angle of rotation about X axis

In the equation below (in a generic setting)

$$\begin{aligned}
\alpha &= \arctan2\left(r_{21}, r_{11}\right) \\
\beta &= \arctan2\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) = \arctan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right) \\
\gamma &= \arctan2\left(r_{32}, r_{33}\right)
\end{aligned} \tag{1.3}$$

Singularity

If the value of $\beta = \pm 90°$, then $r_{21} = r_{11} = r_{32} = r_{33} = 0$. This makes resolving individual α and γ impossible. The two cases are described below

Case 1 If the value of $\beta = 90°$. The Equation 1.2 basically becomes

$$\mathbf{R}_{ZYX}\left(\alpha, \beta = \frac{\pi}{2}, \gamma\right) = \begin{bmatrix} 0 & -\sin(\alpha-\gamma) & \cos(\alpha-\gamma) \\ 0 & \cos(\alpha-\gamma) & \sin(\alpha-\gamma) \\ -1 & 0 & 0 \end{bmatrix} \tag{1.4}$$

Case 2 If the value of $\beta = -90°$, the Equation 1.2 becomes

$$\mathbf{R}_{ZYX}\left(\alpha, \beta = -\frac{\pi}{2}, \gamma\right) = \begin{bmatrix} 0 & -\sin(\alpha+\gamma) & -\cos(\alpha+\gamma) \\ 0 & \cos(\alpha+\gamma) & -\sin(\alpha+\gamma) \\ 1 & 0 & 0 \end{bmatrix} \tag{1.5}$$

All the equations above are implemented as a Python function, code is presented in Appendix A.2 and given with this document.

2 3R Manipulator Kinematics and Workspace

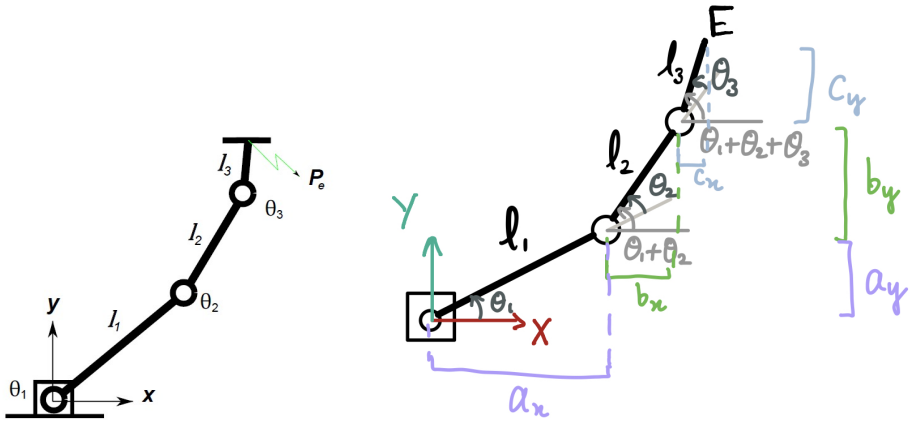
2.1 End effector tool position

Assuming that the end effector is attached at $\mathbf{P_e}$ as shown in Figure 1a, the state that describes the end effector is (E_x, E_y, E_θ). The terms are described in Figure 1. The end effector tool position is given by

$$\begin{bmatrix} E_x \\ E_y \\ E_\theta \end{bmatrix} = \begin{bmatrix} a_x + b_x + c_x \\ a_y + b_y + c_y \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix} = \begin{bmatrix} l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1+\theta_2+\theta_3) \\ l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1+\theta_2+\theta_3) \\ \theta_1+\theta_2+\theta_3 \end{bmatrix} \tag{2.1}$$

This is written in the following shorthand convention

$$\begin{bmatrix} E_x \\ E_y \\ E_\theta \end{bmatrix} = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} \\ l_1s_1 + l_2s_{12} + l_3s_{123} \\ \theta_1+\theta_2+\theta_3 \end{bmatrix} \tag{2.2}$$



(a) 3R Robot (b) FK through geometry

Figure 1: 3R robot and forward kinematics
In sub-figure a, the given manipulator is shown. In sub-figure b, the forward kinematics of the given 3R is depicted. Note that $a_x = l_1\cos(\theta_1)$, $a_y = l_1\sin(\theta_1)$, $b_x = l_2\cos(\theta_1+\theta_2)$, $b_y = l_2\sin(\theta_1+\theta_2)$, $c_x = l_3\cos(\theta_1+\theta_2+\theta_3)$ and $c_y = l_3\sin(\theta_1+\theta_2+\theta_3)$; these values can be derived from geometry. The end effector position, given as $\mathbf{P_e}$ in sub-figure a is written as $\mathbf{E} = (E_x, E_y, E_\theta)$ in sub-figure b.

2.2 Forward Kinematics and Workspace

The forward kinematics of the manipulator is given in Equation 2.2. However, to represent it as a homogeneous SE(2) transformation, we can write it as

$$\begin{aligned} \mathbf{E} &= \begin{bmatrix} \cos(\theta_1+\theta_2+\theta_3) & -\sin(\theta_1+\theta_2+\theta_3) & l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1+\theta_2+\theta_3) \\ \sin(\theta_1+\theta_2+\theta_3) & \cos(\theta_1+\theta_2+\theta_3) & l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1+\theta_2+\theta_3) \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_{123} & -s_{123} & l_1c_1 + l_2c_{12} + l_3c_{123} \\ s_{123} & c_{123} & l_1s_1 + l_2s_{12} + l_3s_{123} \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.3}$$

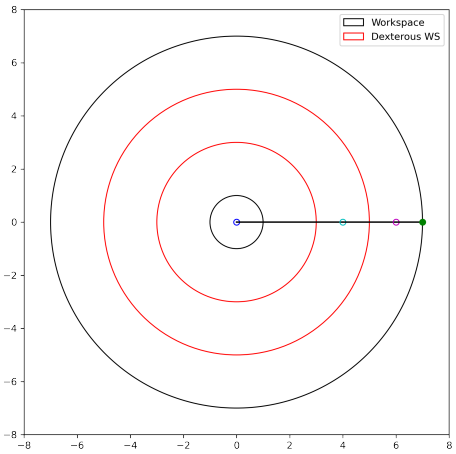
The graphical derivation of this is shown in Figure 1b and has been derived earlier. This is implemented in code in Appendix A.3.1. The code to generate Figure 2 is implemented in Appendix A.3.2. The code for an interactive tool for demonstrating forward kinematics is in A.3.3.

Workspace

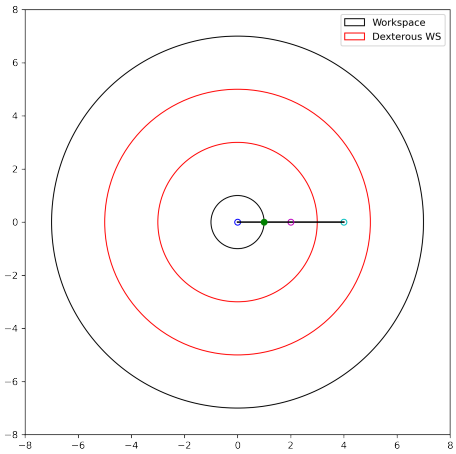
1. The *reachable workspace* is given by the set of points that the manipulator can reach.
2. The *dexterous workspace* is given by the set of points that the manipulator can reach with any arbitrary orientation.

These are shown in Figure 2, along with a few configurations. The reachable workspace is in the region bounded by black circles, the dexterous workspace is in the region bounded by red circles. The corresponding code is in Appendix A.3.2.

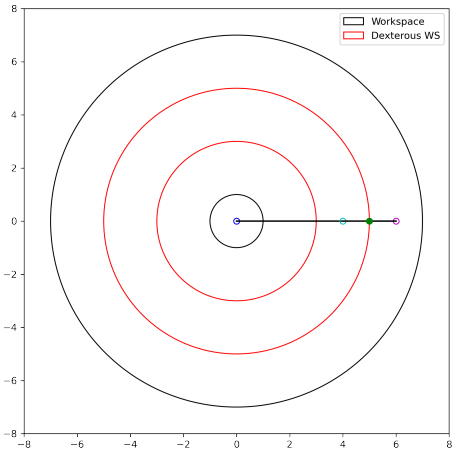
From figures 2c and 2d, it is clear that whenever the joint 2 can trace a full circle with radius l_3, the end effector (which will be at the center of that circle) will be in the dexterous workspace, as it will be possible to orient it at any angle at that endpoint.



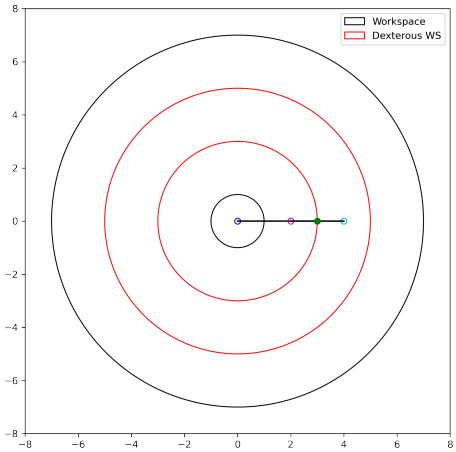
(a) $(0, 0, 0)$



(b) $(0, \pi, 0)$



(c) $(0, 0, \pi)$



(d) $(0, \pi, \pi)$

Figure 2: 3R Workspace and Dexterous Workspace

The reachable workspace and the dexterous workspace of a 3R manipulator. For this figure, the link lengths are $l_1 = 1$, $l_2 = 2$, $l_3 = 4$. The joint 1 is shown in blue circle, the joint 2 is shown in cyan circle, the joint 3 is shown in magenta circle and the end effector is shown in green filled circle. The red circles show the bounds of the dexterous workspace and the black circles show the bounds of the reachable workspace. The figures are captioned with the $(\theta_1, \theta_2, \theta_3)$ values for joint angles. Sub-figures a and b show the outer and inner limits of the *reachable* workspace. Sub-figures c and d show the outer and inner limits of the *dexterous* workspace. The radius of outer black circle is 7 unite, and the radius of inner black circle is 1 unit. The radius of outer red circle is 5 units and that of inner red circle is 3 units. The corresponding code for this output can be found in Appendix A.3.2 and **interactive code** is in Appendix A.3.3.

3 Axis angle convention

Axis-angle to Rotation Matrix

The rotation matrix, in terms of the Axis-angle convention is given by

$$\mathbf{R} = \mathbf{I} + (\sin(\theta))\,[\hat{\mathbf{n}}]_\times + (1-\cos(\theta))\,[\hat{\mathbf{n}}]_\times^2 \tag{3.1}$$

Where θ is the angle of rotation about unit axis $\hat{\mathbf{n}}$ (that is, $n_x^2 + n_y^2 + n_z^2 = 1$). The $[\hat{\mathbf{n}}]_\times$ is the cross product matrix given by

$$\hat{\mathbf{n}} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \qquad [\hat{\mathbf{n}}]_\times = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \qquad [\hat{\mathbf{n}}]_\times^2 = \begin{bmatrix} -n_z^2 - n_y^2 & n_x n_y & n_x n_z \\ n_y n_x & -n_x^2 - n_z^2 & n_y n_z \\ n_z n_x & n_z n_y & -n_y^2 - n_x^2 \end{bmatrix} \tag{3.2}$$

Substituting this to get $\mathbf{R}$ in the equations above, we get

$$\begin{aligned} \mathbf{R} &= \mathbf{I} + (\sin(\theta))\,[\hat{\mathbf{n}}]_\times + (1-\cos(\theta))\,[\hat{\mathbf{n}}]_\times^2 \\ &= \begin{bmatrix} 1+(1-\cos(\theta))(n_x^2-1) & -n_z\sin(\theta)+(1-\cos(\theta))n_x n_y & n_y\sin(\theta)+(1-\cos(\theta))n_x n_z \\ n_z\sin(\theta)+(1-\cos(\theta))n_y n_x & 1+(1-\cos(\theta))(n_y^2-1) & -n_x\sin(\theta)+(1-\cos(\theta))n_y n_z \\ -n_y\sin(\theta)+(1-\cos(\theta))n_z n_x & n_x\sin(\theta)+(1-\cos(\theta))n_z n_y & 1+(1-\cos(\theta))(n_z^2-1) \end{bmatrix} \end{aligned} \tag{3.3}$$

3.1 Converting Rotation Matrix to Axis Angle

To convert rotation matrix to axis-angle numbers (that is, to get $\mathbf{n}$ and θ from $\mathbf{R}$), we can refer to the equation 3.3 and backtrack. We get the following equations

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \theta = \arccos\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right) \qquad \mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \frac{1}{2\sin(\theta)}\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix} \tag{3.4}$$

Validating θ

We can substitute for θ from Equation 3.4 and validate using Equation 3.3.

$$\begin{aligned} &\theta = \arccos\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right) \\ &\Rightarrow \theta = \arccos\left(\frac{1+(1-\cos(\theta))(n_x^2-1)+1+(1-\cos(\theta))(n_y^2-1)+1+(1-\cos(\theta))(n_z^2-1)-1}{2}\right) \\ &\Rightarrow \theta = \arccos\left(\frac{2+(1-\cos(\theta))(n_x^2-1)+(1-\cos(\theta))(n_y^2-1)+(1-\cos(\theta))(n_z^2-1)}{2}\right) \\ &\Rightarrow \theta = \arccos\left(\frac{2+(1-\cos(\theta))(n_x^2+n_y^2+n_z^2-3)}{2}\right) \\ &\Rightarrow \theta = \arccos\left(\frac{2+(1-\cos(\theta))(-2)}{2}\right) = \arccos\left(\frac{2-2+2\cos(\theta)}{2}\right) \\ &\Rightarrow \theta = \arccos\left(\cos(\theta)\right) \Rightarrow \theta = \theta \end{aligned} \tag{3.5}$$

This validates that the formula for θ in Equation 3.4 is correct.

Validating n

We can substitute for $\mathbf{n}$ from Equation 3.4 and validate using Equation 3.3.

$$
\begin{aligned}
\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} &= \frac{1}{2\sin(\theta)} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \\
\Rightarrow \mathbf{n} &= \frac{1}{2\sin(\theta)} \begin{bmatrix} [n_x \sin(\theta) + (1 - \cos(\theta)) n_z n_y] - [-n_x \sin(\theta) + (1 - \cos(\theta)) n_y n_z] \\ [n_y \sin(\theta) + (1 - \cos(\theta)) n_x n_z] - [-n_y \sin(\theta) + (1 - \cos(\theta)) n_z n_x] \\ [n_z \sin(\theta) + (1 - \cos(\theta)) n_y n_x] - [-n_z \sin(\theta) + (1 - \cos(\theta)) n_x n_y] \end{bmatrix} \\
\Rightarrow \mathbf{n} &= \frac{1}{2\sin(\theta)} \begin{bmatrix} 2n_x \sin(\theta) \\ 2n_y \sin(\theta) \\ 2n_z \sin(\theta) \end{bmatrix} \\
\Rightarrow \mathbf{n} &= \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}
\end{aligned}
\tag{3.6}
$$

This validates that the formula for $\mathbf{n}$ in Equation 3.4 is correct.

3.2 Conversion functions

The functions to convert a rotation matrix to axis-angle and axis-angle to rotation matrix are implemented in Appendix A.4.

4 DH Parameters of a 7R Manipulator

Modified DH Parameters

The DH Parameters are given by

1. a_{i-1} *link length*: Distance from Z_{i-1} to Z_i measured along X_{i-1}
2. α_{i-1} *link twist*: Angle measured from Z_{i-1} to Z_i measured along X_{i-1} (using right hand thumb rule)
3. d_i *joint offset*: Distance from X_{i-1} to X_i measured along Z_i
4. θ_i *joint angle*: Angle measured from X_{i-1} to X_i measured along Z_i (using right hand thumb rule)

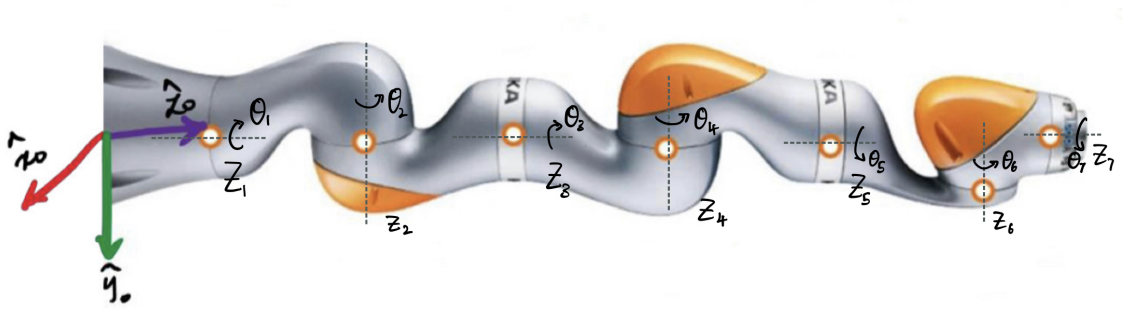
The transformation matrix to represent frame $\{i\}$ in frame $\{i-1\}$ is given by

$$
{}^{i-1}_{i}\mathbf{T} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \cos(\alpha_{i-1})\sin(\theta_i) & \cos(\alpha_{i-1})\cos(\theta) & -\sin(\alpha_{i-1}) & -d_i\sin(\alpha_{i-1}) \\ \sin(\alpha_{i-1})\sin(\theta_i) & \sin(\alpha_{i-1})\cos(\theta) & \cos(\alpha_{i-1}) & d_i\cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}
$$

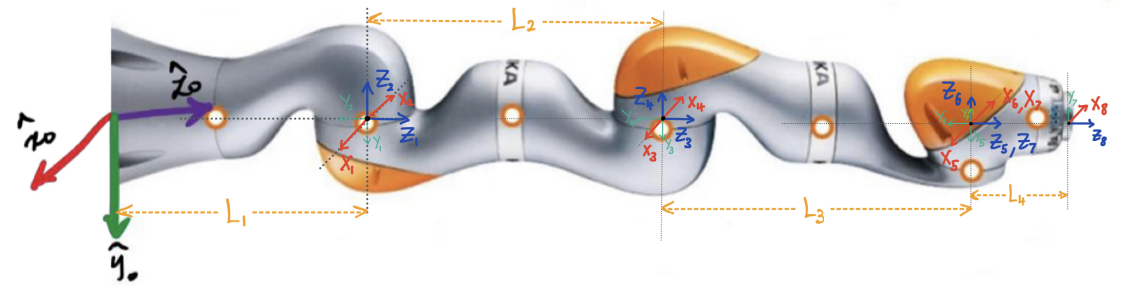
We use Equation 4.1 to construct transformation matrices from DH parameters

4.1 DH Parameters of a 7R Robot

The given robot is a KUKA LBR iiwa collaborative robot.



(a) Axis



(b) Frames

Figure 3: Frames and Axis of given 7R robot
In sub-figure a, the axis of joints are shown. In sub-figure b, the frames (assigned according to modified DH convention) are shown. Only the Z axis and X axis are emphasized (as Y axis can be deduced using $Y = Z \times X$). In the home configuration, it is assumed that there is no vertical offset in the axis that are aligned horizontally (all origins are colinear).

The axis of rotations (joint axis), and the frames are given in Figure 3. The DH Parameters of the 7R robot is mentioned in Table 1.

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	L_1	θ_1
2	0	$\pi/2$	0	$\pi+\theta_2$
3	0	$\pi/2$	L_2	$\pi+\theta_3$
4	0	$\pi/2$	0	$\pi+\theta_4$
5	0	$\pi/2$	L_3	$\pi+\theta_5$
6	0	$\pi/2$	0	$\pi+\theta_6$
7	0	$\pi/2$	0	θ_7
8	0	0	L_4	0

Table 1: Table of DH Parameters for 7R
These are the DH Parameters for the manipulator given in Figure 3. Note that the frame {8} is only to accommodate for the tooltip.

4.2 Transformation matrices from 7R DH Parameters

A Python script that can return transformation matrices given the DH parameters is mentioned in Appendix A.5.

Using the Equation 4.1 and table 1, the following equations can be derived

$$
{}^{0}_{1}\mathbf{T} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^{1}_{2}\mathbf{T} = \begin{bmatrix} -c_2 & s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -s_2 & -c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^{2}_{3}\mathbf{T} = \begin{bmatrix} -c_3 & s_3 & 0 & 0 \\ 0 & 0 & -1 & -L_2 \\ -s_3 & -c_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{3}_{4}\mathbf{T} = \begin{bmatrix} -c_4 & s_4 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^{4}_{5}\mathbf{T} = \begin{bmatrix} -c_5 & s_5 & 0 & 0 \\ 0 & 0 & -1 & -L_3 \\ -s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^{5}_{6}\mathbf{T} = \begin{bmatrix} -c_6 & s_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{6}_{7}\mathbf{T} = \begin{bmatrix} c_7 & -s_7 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_7 & c_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^{7}_{8}\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}
$$

Using the above equations, the end effector can be derived as

$$
\begin{aligned}
{}^{0}_{7}\mathbf{T} &= {}^{0}_{1}\mathbf{T}\,{}^{1}_{2}\mathbf{T}\,{}^{2}_{3}\mathbf{T}\,{}^{3}_{4}\mathbf{T}\,{}^{4}_{5}\mathbf{T}\,{}^{5}_{6}\mathbf{T}\,{}^{6}_{7}\mathbf{T}\,{}^{7}_{8}\mathbf{T} \\
&= \begin{bmatrix} r_{11} & r_{12} & r_{13} & -L_2c_1s_2 - L_3\left(c_1c_4s_2 + s_4\left(c_1c_2c_3 - s_1s_3\right)\right) \\ r_{21} & r_{22} & r_{23} & -L_2s_1s_2 - L_3\left(c_4s_1s_2 + s_4\left(c_1s_3 + c_2c_3s_1\right)\right) \\ r_{31} & r_{32} & r_{33} & L_1 + L_2c_2 + L_3\left(c_2c_4 - c_3s_2s_4\right) \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{0}_{8}\mathbf{T} &= {}^{0}_{7}\mathbf{T}\,{}^{7}_{8}\mathbf{T} \\
&= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{4.3}
$$

The elements for the rotation matrix are mentioned in A.1. The elements for the translation part are mentioned in A.2.

4.3 Validating transformations obtained by DH

In the code mentioned in Appendix A.5, the following home position transformation was obtained for {7} and {8} frame. It can be verified from Figure 3 that this is correct.

$$
{}^{0}_{7}\mathbf{T} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & L_1 + L_2 + L_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad
{}^{0}_{8}\mathbf{T} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & L_1 + L_2 + L_3 + L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}
$$

A Appendix

A.1 Answer 1.1: Code for Euler Angles to Rotation Matrix

The function `euzyx_to_rotm` below converts ZYX Euler angles to $SO(3)$ rotation matrix

```python
# %% Import everything
import numpy as np

# %% Function definitions
# Convert euler angles to rotation matrix
def euzyx_to_rotm(az, ay, ax):
    """
    Convert Euler ZYX angles to Rotation Matrix
    Parameters:
    - az: Along Z axis
    - ay: Along Y axis
    - ax: Along X axis
    Returns:
    - rot_m: A 3x3 rotation matrix
    """
    ca, sa = np.cos(az), np.sin(az) # Angle: Z
    cb, sb = np.cos(ay), np.sin(ay) # Angle: Y
    cg, sg = np.cos(ax), np.sin(ax) # Angle: X
    rot_m = np.array([
        [ca*cb, -sa*cg+sb*sg*ca, sa*sg+sb*ca*cg],
        [sa*cb, sa*sb*sg+ca*cg, sa*sb*cg-sg*ca],
        [-sb, sg*cb, cb*cg]
    ], dtype=float)
    return rot_m

# %%
```

A.2 Answer 1.2: Code for Rotation Matrix to Euler Angles

The function `rotm_to_euzyx` below converts an $SO(3)$ rotation matrix to ZYX Euler angles

```python
# %% Import everything
import numpy as np

# %% Define Functions
# Convert rotation matrix to ruler angles
def rotm_to_euzyx(rot_m):
    """
    Convert Rotation Matrix to Euler ZYX angles. In case of
    singularity, the rotation about Z is assumed to be 0.
    Paraemters:
    - rot_m: A 3x3 SO(3) Rotation Matrix
    Returns:
    - az: Angle of rotation about Z axis (radians)
    - ay: Angle of rotation about Y axis (radians)
    - ax: Angle of rotation about X axis (radians)
    """
    # Angles
    az, ay, ax = None, None, None
    # Procedure
    if (np.isclose(rot_m[2][0], -1)):   # Beta = 90 deg
        az = 0  # Alpha = 0
        ay = np.pi/2
        ax = -np.arctan2(rot_m[1][2], rot_m[0][2])
    elif (np.isclose(rot_m[2][0], 1)):  # Beta = -90 deg
        az = 0  # Alpha = 0
        ay = -np.pi/2
        ax = np.arctan2(-rot_m[1][2], -rot_m[0][2])
    else:   # General case: Not singularity
        az = np.arctan2(rot_m[1][0], rot_m[0][0])
        ay = np.arctan2(-rot_m[2][0],
            np.sqrt(rot_m[0][0]**2+rot_m[1][0]**2))
        ax = np.arctan2(rot_m[2][1], rot_m[2][2])
    return az, ay, ax

# %%
```

A.3 Answer 2.2: Codes pertaining to the 3R manipulator

A.3.1 Forward Kinematics

The code to generate the position of each joint as well as end effector is given below. The function `jfk_min_3r` calculates and returns the (x, y, θ) values of joint 2, joint 3 and the end effector.

The function `min_to_se2` converts the (x, y, θ) values to a $3, 3$ homogeneous SE(2) transformation matrix.

```
# %% Import everything
import numpy as np

# %% Function definitions
# Convert minimal representation to SE2
def min_to_se2(min_repr):
    """
    Converts a minimal representation (x, y, theta) to a 3x3
    homogeneous transformation matrix SE(2)
    Parameters:
    - min_repr: np.ndarray or list  shape: (3,)
        The values of (x, y, theta)
    Returns:
    - se2_repr: np.ndarray      shape: (3, 3)
        The SE(2) representation
    """
    x, y, theta = list(map(float, min_repr))
    return np.array([
        [np.cos(theta), -np.sin(theta), x],
        [np.sin(theta), np.cos(theta), y],
        [0, 0, 1]
    ])


# Forward Kinematics of EF and Joints of 3R manipulator
def jfk_min_3r(t1, t2, t3, l1, l2, l3):
    """
    Return the Forward Kinematics, with joint positions as well. This
    is helpful when plotting.
    Parameters:
    - t1, t2, t3: float(s)
        The joint angles (in radians)
    - l1, l2, l3: float(s)
        The link lengths
    Returns:
    - ef_min: np.ndarray     shape: (3,)
        The (x, y, theta) pose of the end effector
    - j3_min: np.ndarray     shape: (3,)
        The (x, y, theta) pose of the 3rd joint (link 2 to 3)
    - j2_min: np.ndarray     shape: (3,)
        The (x, y, theta) pose of the 3rd joint (link 1 to 2)
    """
    # Joint 2
    j2_min = np.array([l1*np.cos(t1), l1*np.sin(t1), t1+t2])
    # Joint 3
    j3_min = np.array([
        l1*np.cos(t1) + l2*np.cos(t1+t2),
        l1*np.sin(t1) + l2*np.sin(t1+t2),
        t1+t2+t3
    ])
    # End effector
    ef_min = np.array([
        l1*np.cos(t1) + l2*np.cos(t1+t2) + l3*np.cos(t1+t2+t3),
        l1*np.sin(t1) + l2*np.sin(t1+t2) + l3*np.sin(t1+t2+t3),
        t1+t2+t3
    ])
    return ef_min, j3_min, j2_min

# %%
```

A.3.2 Dexterous and Reachable Workspace

The code below shows the *reachable* and *dexterous* workspace.

```
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt
```

```python
from matplotlib import patches as patch
from fk_3r import jfk_min_3r, min_to_se2

# %% Parameters for robot
l1, l2, l3 = 4, 2, 1
axlim = [-8, 8]
jfk_min = lambda t1, t2, t3: jfk_min_3r(t1, t2, t3, l1, l2, l3)
fk_min = lambda t1, t2, t3: jfk_min(t1, t2, t3)[0]
t1, t2, t3 = map(float, np.deg2rad([0, 0, 0]))  # Joint angles (deg)

# %% Show in figure
ef, j3, j2 = jfk_min(t1, t2, t3)
# Workspace circle radius (outer and inner)
wo_cr = abs(fk_min(0, 0, 0)[0])
wi_cr = abs(fk_min(0, np.pi, 0)[0])
print(f"Work space: Outer: {wo_cr:.3f}, Inner: {wi_cr:.3f}")
# Dexterous workspace circle radius (outer and inner)
dwo_cr = abs(fk_min(0, 0, np.pi)[0])
dwi_cr = abs(fk_min(0, np.pi, np.pi)[0])
print(f"Dexterous space: Outer: {dwo_cr:.3f}, Inner: {dwi_cr:.3f}")
fig = plt.figure("3R FK", (8, 8))
ax = fig.add_subplot()
ax.set_aspect('equal', adjustable='box')
# Workspaces
ax.add_patch(patch.Circle((0, 0), wo_cr, fill=False, ec='k',
    label="Workspace"))
ax.add_patch(patch.Circle((0, 0), wi_cr, fill=False, ec='k'))
ax.add_patch(patch.Circle((0, 0), dwo_cr, fill=False, ec='r',
    label="Dexterous WS"))
ax.add_patch(patch.Circle((0, 0), dwi_cr, fill=False, ec='r'))
ax.plot(0, 0, 'bo', fillstyle='none')
ax.plot([0, j2[0]], [0, j2[1]], 'k')
ax.plot(j2[0], j2[1], 'co', fillstyle='none')
ax.plot([j2[0], j3[0]], [j2[1], j3[1]], 'k')
ax.plot(j3[0], j3[1], 'mo', fillstyle='none')
ax.plot([j3[0], ef[0]], [j3[1], ef[1]], 'k')
ax.plot(ef[0], ef[1], 'go')
ax.set_xlim(axlim)
ax.set_ylim(axlim)
ax.legend()
# fig.savefig("ex2-2-dws-inner.png", dpi=300)
plt.show()

# %%
```

This code produces different sub-figures in Figure 2. To change the manipulator orientation, set different joint angles (as captioned in the sub-figures).

A.3.3 Interactive FK

The code below allows one to run forward kinematics in an interactive manner

```python
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import patches as patch
from matplotlib import widgets as wd
from fk_3r import jfk_min_3r, min_to_se2

# %% Robot variables
l1, l2, l3 = map(float, [4, 2, 1])
axlims = [-8, 8]
t1, t2, t3 = map(float, [0, 0, 0])
jfk_min = lambda t1, t2, t3: jfk_min_3r(t1, t2, t3, l1, l2, l3)
fk_min = lambda t1, t2, t3: jfk_min(t1, t2, t3)[0]

# %% Variables
fig = plt.figure("3R Manipulator", (8, 8))
ax = fig.add_subplot()
ax.set_aspect('equal', adjustable='box')
ax.set_xlim(axlims)
ax.set_ylim(axlims)
plt.subplots_adjust(bottom=0.33)
# Reachable and dexterous workspace
wo_cr = abs(fk_min(0, 0, 0)[0])
```

```
wi_cr = abs(fk_min(0, np.pi, 0)[0])
dwo_cr = abs(fk_min(0, 0, np.pi)[0])
dwi_cr = abs(fk_min(0, np.pi, np.pi)[0])
ax.add_patch(patch.Circle((0, 0), wo_cr, fill=False, ec='k'))
ax.add_patch(patch.Circle((0, 0), wi_cr, fill=False, ec='k'))
ax.add_patch(patch.Circle((0, 0), dwo_cr, fill=False, ec='r'))
ax.add_patch(patch.Circle((0, 0), dwi_cr, fill=False, ec='r'))
# Starting pose
ef_pose, j3_pose, j2_pose = jfk_min(t1, t2, t3)
p1, p2, p3 = ax.plot(  # Point to joints and end effector
    [0], [0], 'bo',
    [j2_pose[0]], [j2_pose[1]], 'co',
    [j3_pose[0]], [j3_pose[1]], 'mo', fillstyle='none'
)
pef, = ax.plot([ef_pose[0]], [ef_pose[1]], 'go')
bl1, bl2, bl3 = ax.plot(    # Body links
    [0, j2_pose[0]], [0, j2_pose[1]], 'k-',
    [j2_pose[0], j3_pose[0]], [j2_pose[1], j3_pose[1]], 'k-',
    [j3_pose[0], ef_pose[0]], [j3_pose[1], ef_pose[1]], 'k-'
)

# %% Graphics objects
jlim = [0, 2*np.pi]
axcolor = 'lightgoldenrodyellow'
axj1 = plt.axes([0.19, 0.2, 0.65, 0.03], fc=axcolor)
axj2 = plt.axes([0.19, 0.15, 0.65, 0.03], fc=axcolor)
axj3 = plt.axes([0.19, 0.1, 0.65, 0.03], fc=axcolor)
sj1 = wd.Slider(axj1, "J1", jlim[0], jlim[1], valinit=t1)
sj2 = wd.Slider(axj2, "J2", jlim[0], jlim[1], valinit=t2)
sj3 = wd.Slider(axj3, "J3", jlim[0], jlim[1], valinit=t3)
# Update function for graphics handle
def on_slider_change(val):
    # Calculate FK (from slider values)
    ef_pose, j3_pose, j2_pose = jfk_min(sj1.val, sj2.val, sj3.val)
    # Points
    p1.set_xdata([0])
    p1.set_ydata([0])
    p2.set_xdata([j2_pose[0]])
    p2.set_ydata([j2_pose[1]])
    p3.set_xdata([j3_pose[0]])
    p3.set_ydata([j3_pose[1]])
    pef.set_xdata([ef_pose[0]])
    pef.set_ydata([ef_pose[1]])
    # Body links
    bl1.set_xdata([0, j2_pose[0]])
    bl1.set_ydata([0, j2_pose[1]])
    bl2.set_xdata([j2_pose[0], j3_pose[0]])
    bl2.set_ydata([j2_pose[1], j3_pose[1]])
    bl3.set_xdata([j3_pose[0], ef_pose[0]])
    bl3.set_ydata([j3_pose[1], ef_pose[1]])
    # Update render
    fig.canvas.draw_idle()
# Set the update function
sj1.on_changed(on_slider_change)
sj2.on_changed(on_slider_change)
sj3.on_changed(on_slider_change)

# %% Main plot
plt.show()
```

A.4 Answer 3.2: Axis-angle and Rotation Matrix conversions

The function `rotm_to_axang` in the code below converts a $3,3$ rotation matrix to axis and angle form. The function `axang_to_rotm` in the code below does the opposite (converts axis and angle form to a rotation matrix).

```
# %% Import everything
import numpy as np

# %% Function definitions

# Rotation matrix to Axis angle
def rotm_to_axang(rot_m):
    """
```

```
    Convert rotation matrix to axis-angle representation
    Parameters:
    - rot_m: np.ndarray     shape: (3, 3)
        Rotation Matrix
    Returns:
    - ax: np.ndarray    shape: (3,1)
        The axis of rotation
    - ang: float
        The angle in radians
    """
    # Parse elements
    r11 = rot_m[0][0]
    r12 = rot_m[0][1]
    r13 = rot_m[0][2]
    r21 = rot_m[1][0]
    r22 = rot_m[1][1]
    r23 = rot_m[1][2]
    r31 = rot_m[2][0]
    r32 = rot_m[2][1]
    r33 = rot_m[2][2]
    # Angle
    ang_rad = np.arccos((r11+r22+r33-1)/2)
    # Axis
    ax = (1/(2*np.sin(ang_rad))) * np.array([
        [r32-r23],
        [r13-r31],
        [r21-r12]
    ])
    return ax.flatten(), ang_rad

# Axis-angle to rotation matrix
def axang_to_rotm(ax, ang):
    """
    Convert axis-angle representation to rotation matrix
    Parameters:
    - ax: np.ndarray    shape: (3,)
        Axis of rotation (should be normalized beforehand)
    - ang: float
        The angle of rotation (in radians) using right-hand thumb rule
    Returns:
    - rot_m: np.ndarray     shape: (3, 3)
        The resultant rotation matrix
    """
    nx, ny, nz = map(float, ax)
    st, ct = np.sin(ang), np.cos(ang)
    vt = 1-ct  # For brevity
    rot_m = np.array([
        [1+vt*(nx**2-1), -nz*st+vt*nx*ny, ny*st+vt*nx*nz],
        [nz*st+vt*ny*nx, 1+vt*(ny**2-1), -nx*st+vt*ny*nz],
        [-ny*st+vt*nz*nx, nx*st+vt*nz*ny, 1+vt*(nz**2-1)]
    ])
    return rot_m

# %% Main test
if __name__ == "__main__":
    ax, ang = np.array([1, 0, 0]), np.deg2rad(45)
    ax = ax/np.linalg.norm(ax)
    rot_m = axang_to_rotm(ax, ang)
    ax2, ang2 = rotm_to_axang(rot_m)
    if np.allclose(ax, ax2) and np.allclose(ang, ang2):
        print("Functions seem to map correctly")

# %%
```

A.5 Answer 4.2: Code for DH Parameters of a 7R robot

The function `DH_tf` in the code below returns a transformation matrix given the four DH parameters.

```
# %% Import everything
import numpy as np
import sympy as sp

# %% Functions
# Return DH Parameters
def DH_tf(a, al, d, th):
```

```
    """
    Returns the 4x4 homogeneous transformation matrix, given the four
    modified DH parameters.
    Parameters:
    - a: float      Link length (i-1)
    - al: float     Link twist (i-1)
    - d: float      Joint offset (i)
    - th: float     Joint angle / twist (i)
    Returns
    - htf: sp.Matrix    shape: (4, 4)
        A homogeneous transformation matrix
    """
    c = sp.cos
    s = sp.sin
    return sp.Matrix([
        [c(th), -s(th), 0, a],
        [c(al)*s(th), c(al)*c(th), -s(al), -d*s(al)],
        [s(al)*s(th), s(al)*c(th), c(al), d*c(al)],
        [0, 0, 0, 1]
    ])

# %%
# Joint variables
t1, t2, t3 = sp.symbols(r'\theta_1, \theta_2, \theta_3')
t4, t5, t6 = sp.symbols(r'\theta_4, \theta_5, \theta_6')
t7 = sp.symbols(r"\theta_7")
# Link offsets (joint offsets in DH)
l1, l2, l3, l4 = sp.symbols(r"L_1, L_2, L_3, L_4")
# Some symbols
pi = sp.pi
pi_2 = pi/2

# %% DH Parameters
dh_params = [
    [0, 0, l1, t1],
    [0, pi_2, 0, pi+t2],
    [0, pi_2, l2, pi+t3],
    [0, pi_2, 0, pi+t4],
    [0, pi_2, l3, pi+t5],
    [0, pi_2, 0, pi+t6],
    [0, pi_2, 0, t7],
    [0, 0, l4, 0]
]

# %% Transforms
tf_0_1 = DH_tf(*dh_params[0])
tf_1_2 = DH_tf(*dh_params[1])
tf_2_3 = DH_tf(*dh_params[2])
tf_3_4 = DH_tf(*dh_params[3])
tf_4_5 = DH_tf(*dh_params[4])
tf_5_6 = DH_tf(*dh_params[5])
tf_6_7 = DH_tf(*dh_params[6])
tf_7_8 = DH_tf(*dh_params[7])

# %% Transformations in home reference frame
tf_0_2 = sp.simplify(tf_0_1 * tf_1_2)
tf_0_3 = sp.simplify(tf_0_2 * tf_2_3)
tf_0_4 = sp.simplify(tf_0_3 * tf_3_4)
tf_0_5 = sp.simplify(tf_0_4 * tf_4_5)
tf_0_6 = sp.simplify(tf_0_5 * tf_5_6)
tf_0_7 = sp.simplify(tf_0_6 * tf_6_7)
tf_0_8 = sp.simplify(tf_0_7 * tf_7_8)

# %% Home position verification
subs = {
    t1:0, t2: 0, t3: 0, t4: 0, t5: 0, t6: 0, t7: 0
}
home_pose_7 = sp.simplify(tf_0_7.subs(subs))
home_pose_8 = sp.simplify(tf_0_8.subs(subs))

# %% Short hand substitutions
sh_subs = {
    sp.sin(t1): sp.symbols("s_1"),
    sp.cos(t1): sp.symbols("c_1"),
    sp.sin(t2): sp.symbols("s_2"),
    sp.cos(t2): sp.symbols("c_2"),
```

```
    sp.sin(t3): sp.symbols("s_3"),
    sp.cos(t3): sp.symbols("c_3"),
    sp.sin(t4): sp.symbols("s_4"),
    sp.cos(t4): sp.symbols("c_4"),
    sp.sin(t5): sp.symbols("s_5"),
    sp.cos(t5): sp.symbols("c_5"),
    sp.sin(t6): sp.symbols("s_6"),
    sp.cos(t6): sp.symbols("c_6"),
    sp.sin(t7): sp.symbols("s_7"),
    sp.cos(t7): sp.symbols("c_7"),
}
tf_0_7_sh = sp.simplify(tf_0_7.subs(sh_subs))
tf_0_8_sh = sp.simplify(tf_0_8.subs(sh_subs))

# %%
```

The variable `home_pose_7` and `home_pose_8` contains the home pose of frames {7} and {8} respectively.

Rotation Matrix elements

Using the shorthand notation $c_\theta = \cos(\theta)$, $s_\theta = \sin(\theta)$, we get the following values for elements in the rotation matrix. These are for equation 4.3.

$$
\begin{aligned}
r_{11} =& c_7 \left(c_6 \left(c_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) + s_5 \left(c_1 c_2 s_3 + c_3 s_1\right)\right) + s_6 \left(c_1 c_4 s_2 + s_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right)\right) \\
&+ s_7 \left(c_5 \left(c_1 c_2 s_3 + c_3 s_1\right) - s_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right)\right) \\
r_{12} =& c_7 \left(c_5 \left(c_1 c_2 s_3 + c_3 s_1\right) - s_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right)\right) \\
&- s_7 \left(c_6 \left(c_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) + s_5 \left(c_1 c_2 s_3 + c_3 s_1\right)\right) + s_6 \left(c_1 c_4 s_2 + s_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right)\right) \\
r_{13} =& - c_6 \left(c_1 c_4 s_2 + s_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) + s_6 \left(c_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) + s_5 \left(c_1 c_2 s_3 + c_3 s_1\right)\right) \\
\\
r_{21} =& - c_7 \left(c_6 \left(c_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right) + s_5 \left(c_1 c_3 - c_2 s_1 s_3\right)\right) - s_6 \left(c_4 s_1 s_2 + s_4 \left(c_1 s_3 + c_2 c_3 s_1\right)\right)\right) \\
&- s_7 \left(c_5 \left(c_1 c_3 - c_2 s_1 s_3\right) - s_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right)\right) \\
r_{22} =& - c_7 \left(c_5 \left(c_1 c_3 - c_2 s_1 s_3\right) - s_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right)\right) \\
&+ s_7 \left(c_6 \left(c_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right) + s_5 \left(c_1 c_3 - c_2 s_1 s_3\right)\right) - s_6 \left(c_4 s_1 s_2 + s_4 \left(c_1 s_3 + c_2 c_3 s_1\right)\right)\right) \\
r_{23} =& - c_6 \left(c_4 s_1 s_2 + s_4 \left(c_1 s_3 + c_2 c_3 s_1\right)\right) - s_6 \left(c_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right) + s_5 \left(c_1 c_3 - c_2 s_1 s_3\right)\right) \\
\\
r_{31} =& - c_7 \left(c_6 \left(c_5 \left(c_2 s_4 + c_3 c_4 s_2\right) - s_2 s_3 s_5\right) + s_6 \left(c_2 c_4 - c_3 s_2 s_4\right)\right) + s_7 \left(c_5 s_2 s_3 + s_5 \left(c_2 s_4 + c_3 c_4 s_2\right)\right) \\
r_{32} =& c_7 \left(c_5 s_2 s_3 + s_5 \left(c_2 s_4 + c_3 c_4 s_2\right)\right) + s_7 \left(c_6 \left(c_5 \left(c_2 s_4 + c_3 c_4 s_2\right) - s_2 s_3 s_5\right) + s_6 \left(c_2 c_4 - c_3 s_2 s_4\right)\right) \\
r_{33} =& c_6 \left(c_2 c_4 - c_3 s_2 s_4\right) - s_6 \left(c_5 \left(c_2 s_4 + c_3 c_4 s_2\right) - s_2 s_3 s_5\right)
\end{aligned}
\tag{A.1}
$$

Tooltip transformation matrix elements

These are for equation 4.3.

$$
\begin{aligned}
t_x =& - L_2 c_1 s_2 - L_3 \left(c_1 c_4 s_2 + s_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) \\
&- L_4 \left(c_6 \left(c_1 c_4 s_2 + s_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) - s_6 \left(c_5 \left(c_1 s_2 s_4 - c_4 \left(c_1 c_2 c_3 - s_1 s_3\right)\right) + s_5 \left(c_1 c_2 s_3 + c_3 s_1\right)\right)\right) \\
\\
t_y =& - L_2 s_1 s_2 - L_3 \left(c_4 s_1 s_2 + s_4 \left(c_1 s_3 + c_2 c_3 s_1\right)\right) \\
&- L_4 \left(c_6 \left(c_4 s_1 s_2 + s_4 \left(c_1 s_3 + c_2 c_3 s_1\right)\right) + s_6 \left(c_5 \left(c_4 \left(c_1 s_3 + c_2 c_3 s_1\right) - s_1 s_2 s_4\right) + s_5 \left(c_1 c_3 - c_2 s_1 s_3\right)\right)\right) \\
\\
t_z =& L_1 + L_2 c_2 + L_3 \left(c_2 c_4 - c_3 s_2 s_4\right) + L_4 \left(c_6 \left(c_2 c_4 - c_3 s_2 s_4\right) - s_6 \left(c_5 \left(c_2 s_4 + c_3 c_4 s_2\right) - s_2 s_3 s_5\right)\right)
\end{aligned}
\tag{A.2}
$$