EC4.401 - Assignment 3

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

November 21, 2021

Contents

List of Figures

1 Q1: 3R Manipulator

The figure is shown below

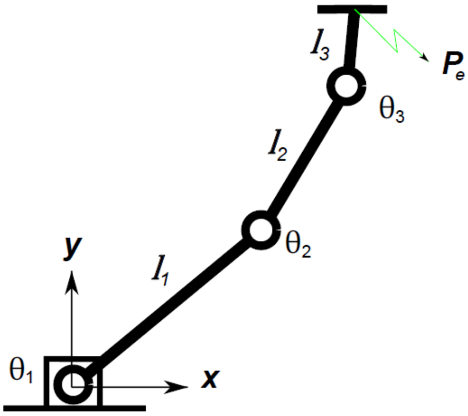


Figure 1: Given 3R manipulator

The forward kinematics for the manipulator shown in Figure 1 is shown below

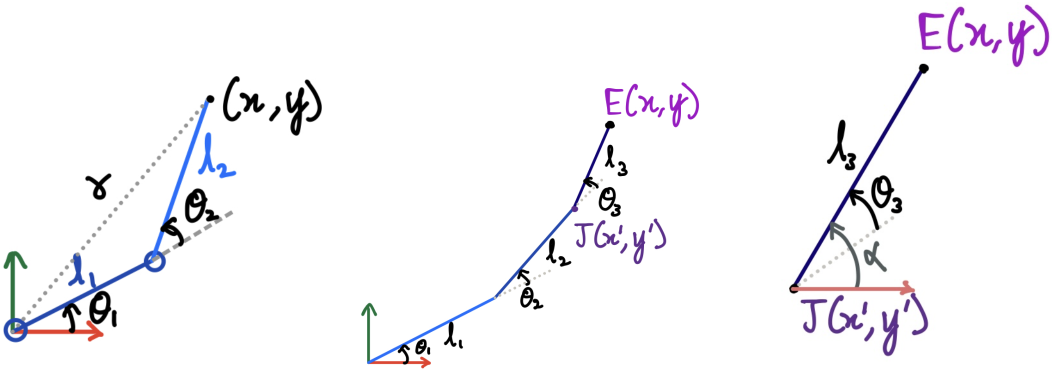
$$\begin{bmatrix} x \\ y \\ \alpha \end{bmatrix} = \begin{bmatrix} l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1+\theta_2+\theta_3) \\ l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1+\theta_2+\theta_3) \\ \theta_1+\theta_2+\theta_3 \end{bmatrix} = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} \\ l_1s_1 + l_2s_{12} + l_3s_{123} \\ \theta_1+\theta_2+\theta_3 \end{bmatrix} \tag{1}$$

Where $[x, y]$ is the position of end effector and α is the orientation in the plane (angle with X axis).

1.1 Inverse Kinematics

Here, given $\mathbf{x} = [x, y, \alpha]^\top$, we need to calculate θ_1, θ_2 and θ_3. We first solve for the θ_1 and θ_2 by focusing on the pose of joint 3 as a 2R manipulator, then solve for θ_3.

The inverse kinematics of a 2R manipulator can be derived using cosine rule and simple geometry as shown in Figure



(a) (b) (c)

Figure 2: 3R Manipulator breakdown
The figure a shows a 2R manipulator that can be visualized as a part of 3R manipulator till the third joint (3R manipulator shown in figure b). The figure c shows the last link separately.

IK of a 2R Manipulator

From figure 2a, it is clear that for a given 2R manipulator, one can derive the joint angles θ_1 and θ_2. The result is shown below

We first find θ_2 using

$$
\begin{aligned}
r^2 = x^2 + y^2 &= (l_1 + l_2\cos(\theta_2))^2 + (l_2\sin(\theta_2))^2 \\
&\Rightarrow x^2 + y^2 = l_1^2 + l_2^2 + 2l_1l_2\cos(\theta_2) \Rightarrow \cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \\
&\Rightarrow \theta_2 = \text{acos}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2}\right) \qquad (2)
\end{aligned}
$$

For θ_1, we do

$$
\begin{aligned}
\tan(\theta_1 + \beta) &= \frac{y}{x} & \tan(\beta) &= \frac{l_2\sin(\theta_2)}{l_1 + l_2\cos(\theta_2)} \\
\Rightarrow \theta_1 + \beta &= \text{atan2}(y, x) & \beta &= \text{atan2}\left(l_2\sin(\theta_2), l_1 + l_2\cos(\theta_2)\right)
\end{aligned}
$$

This gives

$$
\Rightarrow \theta_1 = \text{atan2}(y, x) - \text{atan2}\left(l_2\sin(\theta_2), l_1 + l_2\cos(\theta_2)\right) \qquad (3)
$$

Note that θ_2 is already found using Equation 2.

Extending to 3R

From figure 2c, we can estimate J from E (target end-effector position) using

$$
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x - l_3\cos(\alpha) \\ y - l_3\sin(\alpha) \end{bmatrix} \qquad (4)
$$

We can now feed the point $[x', y']$ (instead of $[x, y]$) to the IK of 2R, that is equations 3 and 2 to get θ_1 and θ_2 respectively. Specifically, we get the inverse kinematics using the equations described below

$$
\begin{aligned}
x_j &= x - l_3\cos(\alpha) & y_j &= y - l_3\sin(\alpha) \\
\theta_2 &= \text{acos}\left(\frac{x_j^2 + y_j^2 - l_1^2 - l_2^2}{2l_1l_2}\right) & & \qquad (5) \\
\theta_1 &= \text{atan2}(y_j, x_j) - \text{atan2}\left(l_2\sin(\theta_2), l_1 + l_2\cos(\theta_2)\right) & & \qquad (6) \\
& & \theta_1 + \theta_2 + \theta_3 &= \alpha \\
\theta_3 &= \alpha - \theta_1 - \theta_2 & & \qquad (7)
\end{aligned}
$$

Using equations 6, 5, and 7 we can obtain the values for θ_1, θ_2 and θ_3 for a 3R manipulator, given the end effector pose $[x, y, \alpha]$.

1.2 Code implementation

The figure 3 shows different stages of an animation that includes moving the end effector along a straight line with a fixed orientation. The code for this is implemented in Appendix A.1.3.

The file `media/3R_IK.mp4` includes a video for illustrating the animation.

1.3 Singularity of 3R

Singularity is a configuration when there is a locking. In such a case, changes in the joint coordinates do not change the end effector position (they bring no velocity). As the robot approaches the singularity, large changes in joint angles are needed to bring small changes to the end effector pose. This mathematically is when the jacobian (either the linear or angular velocity part) becomes zero.

The Jacobian of the 3R manipulator (like the one shown in figure 2b) is shown in equation 8. The determinant of this jacobian matrix is shown in equation 9. It is seen that the determinant of Jacobian is independent of the orientation of the end effector, it is varied only by the angle θ_2.

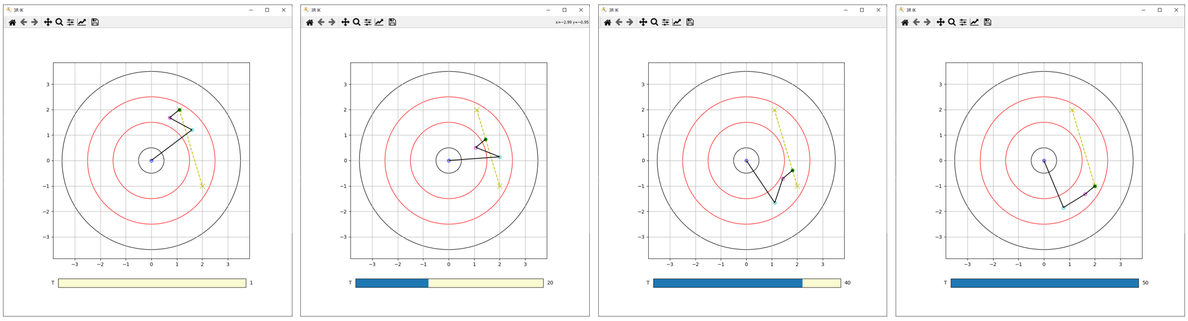


Figure 3: Animations for 3R Inverse Kinematics
A 3R manipulator tracing a path in the 2D plane, while maintaining the orientation. The code is implemented in Appendix A.1.3. The slider can be used move the end effector along the line. The region enclosed between the black circles is the *reachable* workspace and the region enclosed between the red circles is the *dexterous* workspace.

$$\begin{bmatrix} x \\ y \\ \alpha \end{bmatrix} = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} \\ l_1s_1 + l_2s_{12} + l_3s_{123} \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -l_1s_1 - l_2s_{12} - l_3s_{123} & -l_2s_{12} - l_3s_{123} & -l_3s_{123} \\ l_1c_1 + l_2c_{12} + l_3c_{123} & l_2c_{12} + l_3c_{123} & l_3c_{123} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

$$\Rightarrow \mathbf{J} = \begin{bmatrix} -l_1s_1 - l_2s_{12} - l_3s_{123} & -l_2s_{12} - l_3s_{123} & -l_3s_{123} \\ l_1c_1 + l_2c_{12} + l_3c_{123} & l_2c_{12} + l_3c_{123} & l_3c_{123} \\ 1 & 1 & 1 \end{bmatrix} \tag{8}$$

$$\rightarrow \det(\mathbf{J}) = l_1l_2\sin(\theta_1)\cos(\theta_1+\theta_2) + l_1l_2\sin(\theta_1+\theta_2)\cos(\theta_1) = l_1l_2\sin(\theta_2) \tag{9}$$

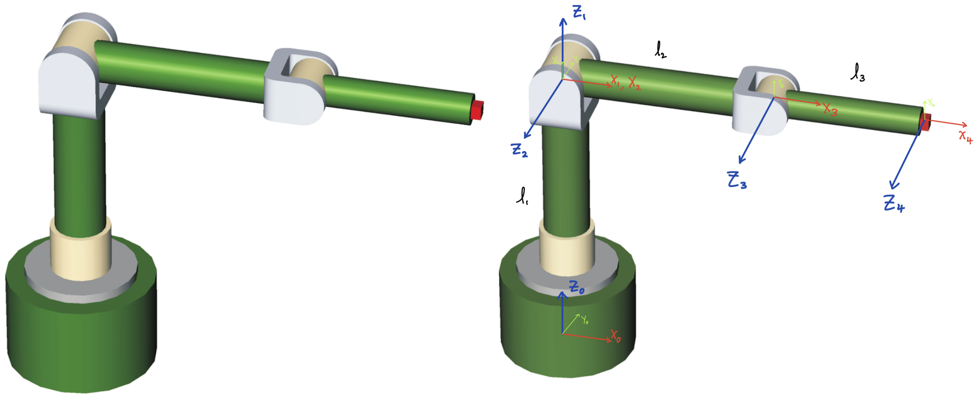
When $\sin(\theta_2) = 0$, then $\theta_2 = 0$ or $\theta_2 = \pi$. These are the singular configurations of the planar 3R manipulator. It can be seen that when such a configuration occurs, the end effector cannot move freely in all directions. It can only move on a loci of a fixed circle (whose radius depends on the orientation of the end effector).

When the orientation is fixed, then the joint angle θ_3 can compensate to fix the orientation. This way, the circle may not be centered at origin, but will be offset (depending upon the link lengths). The end effector, will only be able to trace this circle, it cannot enter the dexterous workspace as shown in figure 3.

2 Q2: Jacobian of 3R Spatial Manipulator

Forward Kinematics

The 3R spatial manipulator (joints in ortho-parallel configuration) can be through of as a 2R manipulator (formed by the last two joints) embed in a plane and that plane itself being able to rotate (because of the first joint).



(a) Manipulator

(b) Frame assignment

Figure 4: 3R ortho-parallel manipulator
On the left (a) is the 3R ortho-parallel manipulator and on the right (b) is the frame assignment using the modern DH convention.

The DH parameters are shown in table 1.

i	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	θ_1	l_1
2	$\pi/2$	0	θ_2	0
3	0	l_2	θ_3	0
4	0	l_3	0	0

Table 1: DH Parameters of 3R ortho-parallel manipulator

The forward kinematics are derived through the equations below

$$
{}^0_1\mathbf{T} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^1_2\mathbf{T} = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^2_3\mathbf{T} = \begin{bmatrix} c_3 & -s_3 & 0 & l_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
{}^3_4\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^0_2\mathbf{T} = \begin{bmatrix} c_1c_2 & -s_2c_1 & s_1 & 0 \\ s_1c_2 & -s_1s_2 & -c_1 & 0 \\ s_2 & c_2 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad
{}^0_3\mathbf{T} = \begin{bmatrix} c_1c_{23} & -s_{23}c_1 & s_1 & l_2c_1c_2 \\ s_1c_{23} & -s_1s_{23} & -c_1 & l_2s_1c_2 \\ s_{23} & c_{23} & 0 & l_1 + l_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}
$$

Which gives

$$
{}^0_4\mathbf{T} = \begin{bmatrix} c_1c_{23} & -s_{23}c_1 & s_1 & (l_2c_2 + l_3c_{23})c_1 \\ s_1c_{23} & -s_1s_{23} & -c_1 & (l_2c_2 + l_3c_{23})s_1 \\ s_{23} & c_{23} & 0 & l_1 + l_2s_2 + l_3s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}
$$

For the angular velocity, we use the Z axis and the rotation matrices of the transforms obtained above.

2.1 Velocity Jacobian

The position of the end effector in the scene is (from equation 11)

$$\mathbf{p} = \begin{bmatrix} (l_2c_2 + l_3c_{23})c_1 \\ (l_2c_2 + l_3c_{23})s_1 \\ l_1 + l_2s_2 + l_3s_{23} \end{bmatrix} \qquad \mathbf{v} = \dot{\mathbf{p}} = \mathbf{J_v} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

$$\mathbf{J_v} = \begin{bmatrix} -(l_2c_2 + l_3c_{23})s_1 & -(l_2s_2 + l_3s_{23})c_1 & l_3s_{23}c_1 \\ (l_2c_2 + l_3c_{23})c_1 & -(l_2s_2 + l_3s_{23})s_1 & l_3s_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{bmatrix} \tag{12}$$

The equation 12 gives the jacobian for the linear velocity part.

2.2 Angular Velocity Jacobian

The angular velocity of the end effector (along XYZ axis) is given by the vector

$$\mathbf{\Omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} {}_0\mathrm{Z}_1 & {}_0\mathrm{Z}_2 & {}_0\mathrm{Z}_3 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} {}^0_1\mathbf{R}\,\mathrm{Z} & {}^0_2\mathbf{R}\,\mathrm{Z} & {}^0_3\mathbf{R}\,\mathrm{Z} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

$$\mathbf{J}_\omega = \begin{bmatrix} {}_0\mathrm{Z}_1 & {}_0\mathrm{Z}_2 & {}_0\mathrm{Z}_3 \end{bmatrix} = \begin{bmatrix} {}^0_1\mathbf{R}\,\mathrm{Z} & {}^0_2\mathbf{R}\,\mathrm{Z} & {}^0_3\mathbf{R}\,\mathrm{Z} \end{bmatrix} \tag{13}$$

Where $\mathrm{Z} = [0, 0, 1]^\top$. We can find the rotation matrices from the transformation matrices in equations 10 and 11 (the first three rows and columns). We get the following

$${}_0\mathrm{Z}_1 = {}^0_1\mathbf{R}\,\mathrm{Z} = {}^0_1\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad {}^0_1\mathbf{R} = \begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}_0\mathrm{Z}_2 = {}^0_2\mathbf{R}\,\mathrm{Z} = {}^0_2\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \qquad {}^0_2\mathbf{R} = \begin{bmatrix} c_1c_2 & -s_2c_1 & s_1 \\ s_1c_2 & -s_1s_2 & -c_1 \\ s_2 & c_2 & 0 \end{bmatrix}$$

$${}_0\mathrm{Z}_3 = {}^0_3\mathbf{R}\,\mathrm{Z} = {}^0_3\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \qquad {}^0_3\mathbf{R} = \begin{bmatrix} c_1c_{23} & -s_{23}c_1 & s_1 \\ s_1c_{23} & -s_1s_{23} & -c_1 \\ s_{23} & c_{23} & 0 \end{bmatrix}$$

Substituting the results of the above equations in equation 13, we get

$$\mathbf{J}_\omega = \begin{bmatrix} {}_0\mathrm{Z}_1 & {}_0\mathrm{Z}_2 & {}_0\mathrm{Z}_3 \end{bmatrix} = \begin{bmatrix} 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{14}$$

Equation 14 gives the jacobian for the angular velocity part.

3 Q3: Dynamics of 3R

The dynamics (equation of motion) of a manipulator is given by the following equation

$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{15}$$

Where

$$\mathbf{M} = \sum_i \left(m_i \mathbf{J}_{\mathbf{v_i}}^\top \mathbf{J}_{\mathbf{v_i}} + \mathbf{J}_{\omega_\mathbf{i}}^\top \mathbf{I}_{\mathbf{C_i}} \mathbf{J}_{\omega_\mathbf{i}} \right) \qquad \mathbf{C} = \left(\dot{\mathbf{M}} - \frac{\mathbf{1}}{\mathbf{2}}\,\dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial \mathbf{q}} \right) \qquad \mathbf{G} = \frac{\partial \mathbf{U}}{\partial \mathbf{q}} \tag{16}$$

We therefore find the jacobian matrices (from forward kinematics), get the mass matrix $\mathbf{M}$, then get the coriolis and centripetal matrix $\mathbf{C}$. The gravity forces $\mathbf{G}$ is directly calculated from the potential energy. Note that the jacobian matrices $\mathbf{J}_{\mathbf{v_i}}$ and $\mathbf{J}_{\omega_\mathbf{i}}$ are calculated till the center of mass of the i^{th} link (consider a sub-manipulator).

The 3R manipulator given in the question is shown in the figure below

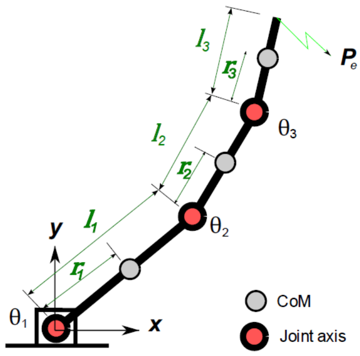


Figure 5: Given 3R Manipulator

3.1 Equation of Motion

Refer to the program in Appendix A.2 for the python code (to solve for the matrices from forward kinematics).

Forward Kinematics

The pose of the center of mass of each link and the end effector is derived. This forward kinematics is done through visual inspection here.

$$\mathbf{P}_{\mathrm{r}_1} = \begin{bmatrix} r_1 c_1 \\ r_1 s_1 \\ \theta_1 \end{bmatrix} \qquad \mathbf{P}_{\mathrm{r}_2} = \begin{bmatrix} l_1 c_1 + r_2 c_{12} \\ l_1 s_1 + r_2 s_{12} \\ \theta_1 + \theta_2 \end{bmatrix} \qquad \mathbf{P}_{\mathrm{r}_3} = \begin{bmatrix} l_1 c_1 + l_2 c_{12} + r_3 c_{123} \\ l_1 s_1 + l_2 s_{12} + r_3 s_{123} \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix} \tag{17}$$

The end effector is given by

$$\mathbf{P}_{\mathrm{ef}} = \begin{bmatrix} l_1 c_1 + l_2 c_{12} + l_3 c_{123} \\ l_1 s_1 + l_2 s_{12} + l_3 s_{123} \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix} \tag{18}$$

Each pose is given by $\mathbf{P} = [x, y, \theta]^\top$. Each $\theta_i = q_i$.

Jacobian Matrices

The Jacobian matrices are now calculated. Note that the linear velocity of center of mass of link i is given by $\mathbf{V_{r_i}} = \mathbf{J_{v_i}}\,\dot{\mathbf{q}}$ and the angular velocity of the center of mass of link i is given by $\mathbf{\Omega_{r_i}} = \mathbf{J}_{\omega_i}\,\dot{\mathbf{q}}$. These will be $3,3$ matrices.

The Jacobian matrices are given by

$$\mathbf{J_{v_1}} = \begin{bmatrix} -r_1 s_1 & 0 & 0 \\ c_1 r_1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{J_{\omega_1}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{19}$$

$$\mathbf{J_{v_2}} = \begin{bmatrix} -l_1 s_1 - r_2 s_{12} & -r_2 s_{12} & 0 \\ c_1 l_1 + c_{12} r_2 & c_{12} r_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{J_{\omega_2}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{20}$$

$$\mathbf{J_{v_3}} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - r_3 s_{123} & -l_2 s_{12} - r_3 s_{123} & -r_3 s_{123} \\ c_1 l_1 + c_{123} r_3 + c_{12} l_2 & c_{123} r_3 + c_{12} l_2 & c_{123} r_3 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{J_{\omega_3}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{21}$$

Mass Matrix

Using the above equations, we get the mass matrix as

$$\mathbf{M} = \sum_i \left(m_i \mathbf{J_{v_i}^\top} \mathbf{J_{v_i}} + \mathbf{J_{\omega_i}^\top} \mathbf{I_{C_i}} \mathbf{J_{\omega_i}} \right) = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \tag{22}$$

Where

$$\begin{aligned}
M_{11} =& I_{zz_1} + I_{zz_2} + I_{zz_3} + 2c_2 l_1 l_2 m_3 + 2c_2 l_1 m_2 r_2 + 2c_3 l_2 m_3 r_3 + 2c_{23} l_1 m_3 r_3 + \\
& l_1^2 m_2 + l_1^2 m_3 + l_2^2 m_3 + m_1 r_1^2 + m_2 r_2^2 + m_3 r_3^2 \\
M_{12} =& I_{zz_2} + I_{zz_3} + c_2 l_1 l_2 m_3 + c_2 l_1 m_2 r_2 + 2c_3 l_2 m_3 r_3 + c_{23} l_1 m_3 r_3 + l_2^2 m_3 + \\
& m_2 r_2^2 + m_3 r_3^2 \\
M_{13} =& I_{zz_3} + c_3 l_2 m_3 r_3 + c_{23} l_1 m_3 r_3 + m_3 r_3^2
\end{aligned} \tag{23}$$

$$\begin{aligned}
M_{21} =& I_{zz_2} + I_{zz_3} + c_2 l_1 l_2 m_3 + c_2 l_1 m_2 r_2 + 2c_3 l_2 m_3 r_3 + c_{23} l_1 m_3 r_3 + l_2^2 m_3 + \\
& m_2 r_2^2 + m_3 r_3^2 \\
M_{22} =& I_{zz_2} + I_{zz_3} + 2c_3 l_2 m_3 r_3 + l_2^2 m_3 + m_2 r_2^2 + m_3 r_3^2 \\
M_{23} =& I_{zz_3} + c_3 l_2 m_3 r_3 + m_3 r_3^2
\end{aligned} \tag{24}$$

$$\begin{aligned}
M_{31} =& I_{zz_3} + c_3 l_2 m_3 r_3 + c_{23} l_1 m_3 r_3 + m_3 r_3^2 \\
M_{32} =& I_{zz_3} + c_3 l_2 m_3 r_3 + m_3 r_3^2 \\
M_{33} =& I_{zz_3} + m_3 r_3^2
\end{aligned} \tag{25}$$

Substitute the values in equations 23, 24 and 25 in equation 22 to get the **Mass Matrix** given by $\mathbf{M}(\mathbf{q})$

Coriolis and Centripetal Matrix

Using 16, we get the coriolis and centripetal part as

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} = \left(\dot{\mathbf{M}} - \frac{\mathbf{1}}{\mathbf{2}}\,\dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial \mathbf{q}} \right) \dot{\mathbf{q}} = \dot{\mathbf{M}}\,\dot{\mathbf{q}} - \frac{1}{2} \begin{bmatrix} \dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial q_1} \dot{\mathbf{q}} \\ \vdots \\ \dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial q_n} \dot{\mathbf{q}} \end{bmatrix} = \dot{\mathbf{M}}\,\dot{\mathbf{q}} - \frac{1}{2} \begin{bmatrix} \dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial q_1} \dot{\mathbf{q}} \\ \dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial q_2} \dot{\mathbf{q}} \\ \dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial q_3} \dot{\mathbf{q}} \end{bmatrix} \tag{26}$$

Calculating the time derivative of the mass matrix is done as follows

$$\frac{\mathrm{d}\mathbf{M}}{\mathrm{d}t} = \dot{\mathbf{M}} = \begin{bmatrix} \frac{\mathrm{d}M_{11}}{\mathrm{d}t} & \frac{\mathrm{d}M_{12}}{\mathrm{d}t} & \frac{\mathrm{d}M_{13}}{\mathrm{d}t} \\ \frac{\mathrm{d}M_{21}}{\mathrm{d}t} & \frac{\mathrm{d}M_{22}}{\mathrm{d}t} & \frac{\mathrm{d}M_{23}}{\mathrm{d}t} \\ \frac{\mathrm{d}M_{31}}{\mathrm{d}t} & \frac{\mathrm{d}M_{32}}{\mathrm{d}t} & \frac{\mathrm{d}M_{33}}{\mathrm{d}t} \end{bmatrix} \tag{27}$$

The derivative of element i, j can be simplified as follows

$$\frac{\mathrm{d}M_{ij}}{\mathrm{d}t} = \frac{\mathrm{d}M_{ij}}{\mathrm{d}q_1}\frac{\mathrm{d}q_1}{\mathrm{d}t} + \frac{\mathrm{d}M_{ij}}{\mathrm{d}q_2}\frac{\mathrm{d}q_2}{\mathrm{d}t} + \frac{\mathrm{d}M_{ij}}{\mathrm{d}q_3}\frac{\mathrm{d}q_3}{\mathrm{d}t} = M_{ij1}\frac{\mathrm{d}q_1}{\mathrm{d}t} + M_{ij2}\frac{\mathrm{d}q_2}{\mathrm{d}t} + M_{ij3}\frac{\mathrm{d}q_3}{\mathrm{d}t} \tag{28}$$

The coriolis and centripetal matrix is therefore achieved as

$$\mathbf{C} = \left(\dot{\mathbf{M}} - \frac{\mathbf{1}}{\mathbf{2}}\dot{\mathbf{q}}^\top \frac{\partial \mathbf{M}}{\partial \mathbf{q}}\right) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{29}$$

Where

$$\begin{aligned}
C_{11} =& -2\dot{q}_2 l_1 l_2 m_3 s_2 - 2\dot{q}_2 l_1 m_2 r_2 s_2 - 2\dot{q}_3 l_2 m_3 r_3 s_3 - 2 l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right) \\
C_{12} =& -\dot{q}_2 l_1 l_2 m_3 s_2 - \dot{q}_2 l_1 m_2 r_2 s_2 - 2\dot{q}_3 l_2 m_3 r_3 s_3 - l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right) \\
C_{13} =& -m_3 r_3 \left(\dot{q}_3 l_2 s_3 + l_1 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right)
\end{aligned} \tag{30}$$

$$\begin{aligned}
C_{21} =& 1.0\dot{q}_1 l_1 l_2 m_3 s_2 + 1.0\dot{q}_1 l_1 m_2 r_2 s_2 + 1.0\dot{q}_1 l_1 m_3 r_3 s_{23} - 0.5\dot{q}_2 l_1 l_2 m_3 s_2 - 0.5\dot{q}_2 l_1 m_2 r_2 s_2 - \\
& 0.5\dot{q}_2 l_1 m_3 r_3 s_{23} - 0.5\dot{q}_3 l_1 m_3 r_3 s_{23} - 2.0\dot{q}_3 l_2 m_3 r_3 s_3 \\
C_{22} =& 0.5\dot{q}_1 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) - 2\dot{q}_3 l_2 m_3 r_3 s_3 \\
C_{23} =& m_3 r_3 \left(0.5\dot{q}_1 l_1 s_{23} - \dot{q}_3 l_2 s_3\right)
\end{aligned} \tag{31}$$

$$\begin{aligned}
C_{31} =& m_3 r_3 \left(1.0\dot{q}_1 l_1 s_{23} + 1.0\dot{q}_1 l_2 s_3 - 0.5\dot{q}_2 l_1 s_{23} + 1.0\dot{q}_2 l_2 s_3 - 0.5\dot{q}_3 l_1 s_{23} - 0.5\dot{q}_3 l_2 s_3\right) \\
C_{32} =& m_3 r_3 \left(0.5\dot{q}_1 \left(l_1 s_{23} + 2 l_2 s_3\right) + 1.0\dot{q}_2 l_2 s_3 - 0.5\dot{q}_3 l_2 s_3\right) \\
C_{33} =& 0.5 m_3 r_3 \left(\dot{q}_1 \left(l_1 s_{23} + l_2 s_3\right) + \dot{q}_2 l_2 s_3\right)
\end{aligned} \tag{32}$$

Substitute the values in equations 30, 31 and 32 in equation 29 to get the **Coriolis and Centripetal Matrix** given by $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$.

Gravity Vector

We first calculate the potential energy of the system: we use the y value of the poses $\mathbf{P}_{\mathrm{r}_i}$ (), then using 16, we get the gravity vector $\mathbf{G}(\mathbf{q})$. The potential energy is given by

$$\begin{aligned}
\mathbf{U} =& -\left(m_1\, g\, \mathbf{P}_{\mathrm{r}_1}[y] + m_2\, g\, \mathbf{P}_{\mathrm{r}_2}[y] + m_3\, g\, \mathbf{P}_{\mathrm{r}_3}[y]\right) \\
=& -g\left(l_1 m_2 s_1 + l_1 m_3 s_1 + l_2 m_3 s_{12} + m_1 r_1 s_1 + m_2 r_2 s_{12} + m_3 r_3 s_{123}\right)
\end{aligned} \tag{33}$$

The partial derivative with respect to $\mathbf{q}$ yields the gravity vector, which is given by

$$\mathbf{G}(\mathbf{q}) = \frac{\partial \mathbf{U}}{\partial \mathbf{q}} = \begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix} = \begin{bmatrix} -g\left(c_1 l_1 m_2 + c_1 l_1 m_3 + c_1 m_1 r_1 + c_{123} m_3 r_3 + c_{12} l_2 m_3 + c_{12} m_2 r_2\right) \\ -g\left(c_{123} m_3 r_3 + c_{12} l_2 m_3 + c_{12} m_2 r_2\right) \\ -c_{123} g m_3 r_3 \end{bmatrix} \tag{34}$$

The equation 34 gives the **Gravity Vector** given by $\mathbf{G}(\mathbf{q})$.

Equation of Motion

The joint efforts (torques) τ are given by

$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{35}$$

$$= \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{36}$$

Where

$$
\begin{aligned}
\tau_1 =& \ddot{q}_1 \left(I_{zz_1} + I_{zz_2} + I_{zz_3} + 2c_2 l_1 l_2 m_3 + 2c_2 l_1 m_2 r_2 + 2c_3 l_2 m_3 r_3 + 2c_{23} l_1 m_3 r_3 + l_1^2 m_2 + l_1^2 m_3 \right.\\
&\left. + l_2^2 m_3 + m_1 r_1^2 + m_2 r_2^2 + m_3 r_3^2\right) + \ddot{q}_2 \left(I_{zz_2} + I_{zz_3} + c_2 l_1 l_2 m_3 + c_2 l_1 m_2 r_2 + 2c_3 l_2 m_3 r_3 \right.\\
&\left. + c_{23} l_1 m_3 r_3 + l_2^2 m_3 + m_2 r_2^2 + m_3 r_3^2\right) + \ddot{q}_3 \left(I_{zz_3} + c_3 l_2 m_3 r_3 + c_{23} l_1 m_3 r_3 + m_3 r_3^2\right)\\
&- 2\dot{q}_1 \left(\dot{q}_2 l_1 l_2 m_3 s_2 + \dot{q}_2 l_1 m_2 r_2 s_2 + \dot{q}_3 l_2 m_3 r_3 s_3 + l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right) - \dot{q}_2 \left(\dot{q}_2 l_1 l_2 m_3 s_2 \right.\\
&\left. + \dot{q}_2 l_1 m_2 r_2 s_2 + 2\dot{q}_3 l_2 m_3 r_3 s_3 + l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right) - \dot{q}_3 m_3 r_3 \left(\dot{q}_3 l_2 s_3 + l_1 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right)\\
&- g \left(c_1 l_1 m_2 + c_1 l_1 m_3 + c_1 m_1 r_1 + c_{123} m_3 r_3 + c_{12} l_2 m_3 + c_{12} m_2 r_2\right)
\end{aligned}
\tag{37}
$$

$$
\begin{aligned}
\tau_2 =& 1.0 I_{zz_2} \ddot{q}_1 + 1.0 I_{zz_2} \ddot{q}_2 + 1.0 I_{zz_3} \ddot{q}_1 + 1.0 I_{zz_3} \ddot{q}_2 + 1.0 I_{zz_3} \ddot{q}_3 + 1.0 \ddot{q}_1 c_2 l_1 l_2 m_3 + 1.0 \ddot{q}_1 c_2 l_1 m_2 r_2\\
&+ 2.0 \ddot{q}_1 c_3 l_2 m_3 r_3 + 1.0 \ddot{q}_1 c_{23} l_1 m_3 r_3 + 1.0 \ddot{q}_1 l_2^2 m_3 + 1.0 \ddot{q}_1 m_2 r_2^2 + 1.0 \ddot{q}_1 m_3 r_3^2 + 2.0 \ddot{q}_2 c_3 l_2 m_3 r_3\\
&+ 1.0 \ddot{q}_2 l_2^2 m_3 + 1.0 \ddot{q}_2 m_2 r_2^2 + 1.0 \ddot{q}_2 m_3 r_3^2 + 1.0 \ddot{q}_3 c_3 l_2 m_3 r_3 + 1.0 \ddot{q}_3 m_3 r_3^2 + 1.0 \dot{q}_1^2 l_1 l_2 m_3 s_2\\
&+ 1.0 \dot{q}_1^2 l_1 m_2 r_2 s_2 + 1.0 \dot{q}_1^2 l_1 m_3 r_3 s_{23} - 2.0 \dot{q}_1 \dot{q}_3 l_2 m_3 r_3 s_3 - 2.0 \dot{q}_2 \dot{q}_3 l_2 m_3 r_3 s_3 - 1.0 \dot{q}_3^2 l_2 m_3 r_3 s_3\\
&- 1.0 c_{123} g m_3 r_3 - 1.0 c_{12} g l_2 m_3 - 1.0 c_{12} g m_2 r_2
\end{aligned}
\tag{38}
$$

$$
\begin{aligned}
\tau_3 =& 1.0 I_{zz_3} \ddot{q}_1 + 1.0 I_{zz_3} \ddot{q}_2 + 1.0 I_{zz_3} \ddot{q}_3 + 1.0 \ddot{q}_1 c_3 l_2 m_3 r_3 + 1.0 \ddot{q}_1 c_{23} l_1 m_3 r_3 + 1.0 \ddot{q}_1 m_3 r_3^2\\
&+ 1.0 \ddot{q}_2 c_3 l_2 m_3 r_3 + 1.0 \ddot{q}_2 m_3 r_3^2 + 1.0 \ddot{q}_3 m_3 r_3^2 + 1.0 \dot{q}_1^2 l_1 m_3 r_3 s_{23} + 1.0 \dot{q}_1^2 l_2 m_3 r_3 s_3\\
&+ 2.0 \dot{q}_1 \dot{q}_2 l_2 m_3 r_3 s_3 + 1.0 \dot{q}_2^2 l_2 m_3 r_3 s_3 - 1.0 c_{123} g m_3 r_3
\end{aligned}
\tag{39}
$$

Substituting the values in equations 37, 38 and 39 in equation 36, we get the joint efforts (equations of motion) τ.

3.2 Symmetric Mass Matrix

From equation 22, it is evident that $M_{12} = M_{21}$, $M_{13} = M_{31}$ and $M_{23} = M_{32}$. Therefore $\mathbf{M} = \mathbf{M}^\top$, in other words, the mass matrix is a **symmetric matrix**.

Since $\mathbf{M} = \mathbf{M}^\top$,

$$
\mathbf{M} = \mathbf{M}^\top \Rightarrow \mathbf{M} - \mathbf{M}^\top = \mathbf{0} \tag{40}
$$

This is also verified through the program in appendix A.2.

3.3 Skew-Symmetric matrix in model

A matrix A is skew symmetric if $A = -A^\top$.

Deriving the Equation of motion for an open-chain manipulator

$$
L(\theta, \dot{\theta}) = \frac{1}{2} \dot{\theta}^\top M(\theta) \dot{\theta} - V(\theta) \qquad \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i} = \Upsilon_i
$$

$$
\frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}_i} = \frac{d}{dt} \left(\sum_{j=1}^{n} M_{ij} \dot{\theta}_j \right) = \sum_{j=1}^{n} \left(M_{ij} \ddot{\theta}_j + \dot{M}_{ij} \dot{\theta}_j \right)
$$

$$
\frac{\partial L}{\partial \theta_i} = \frac{1}{2} \sum_{j,k=1}^{n} \frac{\partial M_{kj}}{\partial \theta_i} \dot{\theta}_k \dot{\theta}_j - \frac{\partial V}{\partial \theta_i}
$$

$$
\sum_{j=1}^{n} M_{ij} \ddot{\theta}_j + \sum_{j,k=1}^{n} \left(\frac{\partial M_{ij}}{\partial \theta_k} \dot{\theta}_j \dot{\theta}_k - \frac{1}{2} \frac{\partial M_{kj}}{\partial \theta_i} \dot{\theta}_k \dot{\theta}_j \right) + \frac{\partial V}{\partial \theta_i}(\theta) = \Upsilon_i \tag{41}
$$

From the model of the Coriolis matrix, it can be written in terms of Christoffel symbols as follows

$$
C(q, \dot{q}) := \{c_{ij}\} = \frac{1}{2} \left\{ \sum_{k=1}^{n} \left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{kj}}{\partial q_i} \right) \dot{q}_k \right\} \tag{42}
$$

Where $M(q) = \{m_{ij}\}$ is the mass matrix. Using this, it can be shown that $(\dot{M} - 2C)$ is skew symmetric.

$$\left(\dot{M}-2C\right)_{ij} = \dot{M}_{ij}(\theta) - 2C_{ij}(\theta) \tag{43}$$

$$= \sum_{k=1}^{n} \frac{\partial M_{ij}}{\partial \theta_k}\dot{\theta}_k - \frac{\partial M_{ij}}{\partial \theta_k}\dot{\theta}_k - \frac{\partial M_{ik}}{\partial \theta_j}\dot{\theta}_k + \frac{\partial M_{kj}}{\partial \theta_i}\dot{\theta}_k \tag{44}$$

$$= \sum_{k=1}^{n} \frac{\partial M_{kj}}{\partial \theta_i}\dot{\theta}_k - \frac{\partial M_{ik}}{\partial \theta_j}\dot{\theta}_k \tag{45}$$

Switching i and j shows that $\left(\dot{M}-2C\right)^{\top} = -\left(\dot{M}-2C\right)$. This proof is from the section 3.2 of Murray-1994 [1].

However, for the matrices defined above, the matrix $\left(\dot{M}-2C\right)$ does not appear to come out as a skew symmetric matrix. This may be because of the choice of C. Check the output at the end of appendix A.2 and the resultant matrix in equation 51. To obtain a skew symmetric matrix, we use the Cristoffel symbols.

Cristoffel Symbols

We use the following equation for the new Coriolis matrix (a better method to find the coriolis matrix)

$$\mathbf{C}_{\text{cris}} := \{c_{ij}\} = \left\{ \frac{1}{2}\sum_{k=1}^{n} \left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{kj}}{\partial q_i} \right) \dot{q}_k \right\} \tag{46}$$

Following the above (see the last segment of code in Appendix A.2), we get

$$\mathbf{C}_{\text{cris}} = \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix} \tag{47}$$

Where

$$\begin{aligned}
V_{11} =& -1.0\dot{q}_2 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) - 1.0\dot{q}_3 m_3 r_3 \left(l_1 s_{23} + l_2 s_3\right) \\
V_{12} =& -1.0\dot{q}_1 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) - 1.0\dot{q}_2 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) - \\
& 1.0\dot{q}_3 m_3 r_3 \left(l_1 s_{23} + l_2 s_3\right) \\
V_{13} =& -1.0 m_3 r_3 \left(l_1 s_{23} + l_2 s_3\right)\left(\dot{q}_1 + \dot{q}_2 + \dot{q}_3\right) && (48) \\
V_{21} =& \dot{q}_1 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) - 1.0\dot{q}_3 l_2 m_3 r_3 s_3 \\
V_{22} =& -1.0\dot{q}_3 l_2 m_3 r_3 s_3 \\
V_{23} =& -1.0 l_2 m_3 r_3 s_3 \left(\dot{q}_1 + \dot{q}_2 + \dot{q}_3\right) && (49) \\
V_{31} =& m_3 r_3 \left(\dot{q}_1 \left(l_1 s_{23} + l_2 s_3\right) + 1.0\dot{q}_2 l_2 s_3\right) \\
V_{32} =& 1.0 l_2 m_3 r_3 s_3 \left(\dot{q}_1 + \dot{q}_2\right) \\
V_{33} =& 0 && (50)
\end{aligned}$$

Substitute the equations 48, 49, 50 in equation 47 to get the new coriolis matrix $\mathbf{C}_{\text{cris}}$ through Cristoffel symbols.

In Appendix A.2, through ssm_{cris}, it is verified that $\dot{\mathbf{M}} - 2\mathbf{C}_{\text{cris}}$ is a skew symmetric matrix. Check equation 55 for the proof of this.

[1] Murray, R.M., Li, Z., and Sastry, S. (1994). *A Mathematical Introduction to Robotic Manipulation.* CRC Press

A Appendix

A.1 3R Inverse Kinematics Animation

A.1.1 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.

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

# %% Function definitions
# 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.1.2 Inverse Kinematics

The code to generate the joint angles is given below. The function `ik_3r` calculates and returns the $(\theta_1, \theta_2, \theta_3)$ that are the joint values for the manipulator.

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

# %% Functions
# IK of 3R
def ik_3r(x, y, al, l1=2, l2=1, l3=0.5):
    """
    The inverse kinematics of a 3R manipulator

    Parameters:
    - x, y, al: The point (x, y) and pose (al) in the plane
    - l1, l2, l3    default: (2, 1, 0.5) respectively
        The link lengths

    Returns:

    """
```

```python
    xj = x - l3*np.cos(al)
    yj = y - l3*np.sin(al)
    # Angles
    th2 = np.arccos((xj**2+yj**2-l1**2-l2**2)/(2*l1*l2))
    th1 = np.arctan2(yj, xj) - \
        np.arctan2(l2*np.sin(th2), l1+l2*np.cos(th2))
    th3 = al - th1 - th2
    # Return angles
    return th1, th2, th3

# %%
```

Equations 6, 5 and 7 are used for the implementation above.

A.1.3 3R IK Animation

The animation for IK (of a 3R manipulator) is generated by the code below. This depends upon code in Appendix A.1.1 and A.1.2. Set the variables `ik_start`, `ik_end` and `ik_th` accordingly.

```python
# %% Import everything
from matplotlib import pyplot as plt
from matplotlib import patches as patch
from matplotlib import widgets as wd
import numpy as np
# Kinematics
from ik_3r import ik_3r
from fk_3r import jfk_min_3r

# %% Variables
l1, l2, l3 = map(float, [2, 1, 0.5])    # Link lengths
# Line properties
ln_start = [1.1, 2]     # Starting point (x, y)
ln_end = [2, -1]        # Ending point (x, y)
ln_th = np.deg2rad(40)  # Angle to maintain throughout
num_ts = 50             # Number of timesteps
# Minimal representation poses
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]
# Inverse Kinematics
ik_min = lambda x, y, th: ik_3r(x, y, th, l1, l2, l3)
# Axis limit
lims = [-(l1+l2+l3)*1.1, (l1+l2+l3)*1.1]
# Interpolate
t = np.linspace(0, 1, num_ts)
ln_t = np.vstack((
    np.array(ln_start)[0] * (1-t) + np.array(ln_end)[0] * t,
    np.array(ln_start)[1] * (1-t) + np.array(ln_end)[1] * t))

# %% Show in figure
fig = plt.figure("3R IK", (8, 8))
plt.subplots_adjust(bottom=0.2)
ax = fig.add_subplot()
ax.set_aspect('equal', adjustable='box')
# Limits
ax.grid()
ax.set_xlim(lims)
ax.set_ylim(lims)
# 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'))
# Plot line
ax.plot(ln_start[0], ln_start[1], 'yx',
    ln_end[0], ln_end[1], 'yx', ms=10)
ax.plot(ln_t[0], ln_t[1], 'y--')
# Inverse kinematics
x, y, th = ln_start[0], ln_start[1], ln_th
a1, a2, a3 = ik_min(x, y, th)
```

```
ef_p, j3_p, j2_p = jfk_min(a1, a2, a3)
pj1, pj2, pj3 = ax.plot(    # Joints
    [0], [0], 'bo',
    [j2_p[0]], [j2_p[1]], 'co',
    [j3_p[0]], [j3_p[1]], 'mo', fillstyle='none'
)
pef, = ax.plot([ef_p[0]], [ef_p[1]], 'go') # End effector
bl1, bl2, bl3 = ax.plot(    # Body links
    [0, j2_p[0]], [0, j2_p[1]], 'k-',
    [j2_p[0], j3_p[0]], [j2_p[1], j3_p[1]], 'k-',
    [j3_p[0], ef_p[0]], [j3_p[1], ef_p[1]], 'k-'
)

# %% Graphics object
ts = [1, len(t)]
axcolor = 'lightgoldenrodyellow'
axj1 = plt.axes([0.19, 0.1, 0.65, 0.03], fc=axcolor)
sts = wd.Slider(axj1, "T", ts[0], ts[1], valinit=0, valstep=1)
# Update function for graphics handle
def on_slider_update(tsn_f):
    tsn = int(tsn_f) - 1
    # Target point
    x, y, th = ln_t[0][tsn], ln_t[1][tsn], ln_th
    a1, a2, a3 = ik_min(x, y, th)   # Angles
    ef_p, j3_p, j2_p = jfk_min(a1, a2, a3)  # FK
    # Update points
    pj2.set_xdata([j2_p[0]])
    pj2.set_ydata([j2_p[1]])
    pj3.set_xdata([j3_p[0]])
    pj3.set_ydata([j3_p[1]])
    pef.set_xdata([ef_p[0]])
    pef.set_ydata([ef_p[1]])
    # Body links
    bl1.set_xdata([0, j2_p[0]])
    bl1.set_ydata([0, j2_p[1]])
    bl2.set_xdata([j2_p[0], j3_p[0]])
    bl2.set_ydata([j2_p[1], j3_p[1]])
    bl3.set_xdata([j3_p[0], ef_p[0]])
    bl3.set_ydata([j3_p[1], ef_p[1]])
    # Update render
    fig.canvas.draw_idle()
# Set update function
sts.on_changed(on_slider_update)

# %% Main plot
plt.show()

# %%
```

A.2 3R Dynamic Modeling

The code to generate and dynamic model (equations) of the 3R manipulator is presented in this section.

```
# %% Import everything
import sympy as sp
import numpy as np
from IPython.display import display

# %% Variables
# Joint angles
t1, t2, t3 = sp.symbols(r"\theta_1, \theta_2, \theta_3")
# Link lengths
l1, l2, l3 = sp.symbols(r"l_1, l_2, l_3")
# C.O.M. offsets
r1, r2, r3 = sp.symbols(r"r_1, r_2, r_3")
# Link masses
m1, m2, m3 = sp.symbols(r"m_1, m_2, m_3")
# Moment of inertias (all along Z axis only)
izz1, izz2, izz3 = sp.symbols(r"I_{zz_1}, I_{zz_2}, I_{zz_3}")
Icc1, Icc2, Icc3 = [sp.Matrix(np.diag([0, 0, izz])) for izz in \
    [izz1, izz2, izz3]]

```

```python
# %% Forward kinematics
pr1 = sp.Matrix([
    [r1*sp.cos(t1)],
    [r1*sp.sin(t1)],
    [0],
    [0], [0], [t1]
])  # Pose of COM (x, y, z, tx, ty, tz) of link 1 -> r1
pr2 = sp.Matrix([
    [l1*sp.cos(t1) + r2*sp.cos(t1+t2)],
    [l1*sp.sin(t1) + r2*sp.sin(t1+t2)],
    [0],
    [0], [0], [t1+t2]
])  # Pose of COM of link 2 -> r2
pr3 = sp.Matrix([
    [l1*sp.cos(t1) + l2*sp.cos(t1+t2) + r3*sp.cos(t1+t2+t3)],
    [l1*sp.sin(t1) + l2*sp.sin(t1+t2) + r3*sp.sin(t1+t2+t3)],
    [0],
    [0], [0], [t1+t2+t3]
])  # Pose of COM of link 3 -> r3
pef = sp.Matrix([
    [l1*sp.cos(t1) + l2*sp.cos(t1+t2) + l3*sp.cos(t1+t2+t3)],
    [l1*sp.sin(t1) + l2*sp.sin(t1+t2) + l3*sp.sin(t1+t2+t3)],
    [0],
    [0], [0], [t1+t2+t3]
])  # Pose of end effector -> pef

# %% Shorthand subs
sh_subs = {
    sp.sin(t1): sp.symbols(r"s_1"),
    sp.cos(t1): sp.symbols(r"c_1"),
    sp.sin(t2): sp.symbols(r"s_2"),
    sp.cos(t2): sp.symbols(r"c_2"),
    sp.sin(t3): sp.symbols(r"s_3"),
    sp.cos(t3): sp.symbols(r"c_3"),
    sp.sin(t1+t2): sp.symbols(r"s_{12}"),
    sp.cos(t1+t2): sp.symbols(r"c_{12}"),
    sp.sin(t2+t3): sp.symbols(r"s_{23}"),
    sp.cos(t2+t3): sp.symbols(r"c_{23}"),
    sp.sin(t1+t2+t3): sp.symbols(r"s_{123}"),
    sp.cos(t1+t2+t3): sp.symbols(r"c_{123}")
}   # Angles in short hand

# %% Jacobians
Jv1 = sp.Matrix.hstack(pr1.diff(t1), pr1.diff(t2),
    pr1.diff(t3))[0:3,:]     # Jv1: Velocity (Pr1)
Jv2 = sp.Matrix.hstack(pr2.diff(t1), pr2.diff(t2),
    pr2.diff(t3))[0:3,:]     # Jv2: Velocity (Pr2)
Jv3 = sp.Matrix.hstack(pr3.diff(t1), pr3.diff(t2),
    pr3.diff(t3))[0:3,:]     # Jv3: Velocity (Pr3)
# Get the velocity jacobian using (2nd for example)
# print(sp.latex(sp.simplify(Jv2).subs(sh_subs)))

Jw1 = sp.Matrix.hstack(pr1.diff(t1), pr1.diff(t2),
    pr1.diff(t3))[3:6,:]     # Jw1: Angular Velocity (Pr1)
Jw2 = sp.Matrix.hstack(pr2.diff(t1), pr2.diff(t2),
    pr2.diff(t3))[3:6,:]     # Jw2: Angular Velocity (Pr2)
Jw3 = sp.Matrix.hstack(pr3.diff(t1), pr3.diff(t2),
    pr3.diff(t3))[3:6,:]     # Jw3: Angular Velocity (Pr3)
# Get the angular velocity jacobian using (2nd for example)
# print(sp.latex(sp.simplify(Jw2).subs(sh_subs)))

# %% Mass matrix
M = m1 * Jv1.T * Jv1 + Jw1.T * Icc1 * Jw1 + \
    m2 * Jv2.T * Jv2 + Jw2.T * Icc2 * Jw2 + \
    m3 * Jv3.T * Jv3 + Jw3.T * Icc3 * Jw3
# Get cell values using (2nd row, 3rd column example)
# print(sp.latex(sp.simplify(M[1,2]).subs(sh_subs)))
M = sp.simplify(M)

# %% Time dependent symbols
t = sp.Symbol("t")
q1 = sp.Function(r"q_1")(t)
q2 = sp.Function(r"q_2")(t)
```

```python
q3 = sp.Function(r"q_3")(t)
q = sp.Matrix([[q1], [q2], [q3]])
q_dot = q.diff(t)
M_t = M.subs({t1:q1, t2:q2, t3:q3})
M_dot = M_t.diff(t) # Time derivative for mass matrix

# %% Coriolis and Centripetal Matrix
qdt_Mdiff = sp.Matrix.vstack(
    q_dot.T * M_t.diff(q1),
    q_dot.T * M_t.diff(q2),
    q_dot.T * M_t.diff(q3))     # q.T * diff(M, q)
C_q_qdot = sp.simplify(M_dot - (1/2) * qdt_Mdiff)

# %% Gravity Vector
g = sp.symbols(r"g")
U = -(m1*g*pr1[1] + m2*g*pr2[1] + m3*g*pr3[1])
U_t = sp.simplify(U.subs({t1:q1, t2:q2, t3:q3}))    # Potential energy
U_tm = sp.Matrix([U_t]) # As a 1 element matrix
G = sp.Matrix.vstack(U_tm.diff(q1), U_tm.diff(q2), U_tm.diff(q3))

# %% Final torque equation
q_ddot = q_dot.diff(t)   # Second time derivative
tau = sp.simplify(M_t * q_ddot + C_q_qdot * q_dot + G)

# %% Shorthand subs
sh_subs = {
    q1.diff(t): sp.symbols(r"\dot{q}_1"),
    q2.diff(t): sp.symbols(r"\dot{q}_2"),
    q3.diff(t): sp.symbols(r"\dot{q}_3"),
    q1.diff(t, 2): sp.symbols(r"\ddot{q}_1"),
    q2.diff(t, 2): sp.symbols(r"\ddot{q}_2"),
    q3.diff(t, 2): sp.symbols(r"\ddot{q}_3"),
    q1: sp.symbols(r"q_1"),
    q2: sp.symbols(r"q_2"),
    q3: sp.symbols(r"q_3"),
    sp.sin(q1): sp.symbols(r"s_1"),
    sp.cos(q1): sp.symbols(r"c_1"),
    sp.sin(q2): sp.symbols(r"s_2"),
    sp.cos(q2): sp.symbols(r"c_2"),
    sp.sin(q3): sp.symbols(r"s_3"),
    sp.cos(q3): sp.symbols(r"c_3"),
    sp.sin(q1+q2): sp.symbols(r"s_{12}"),
    sp.cos(q1+q2): sp.symbols(r"c_{12}"),
    sp.sin(q2+q3): sp.symbols(r"s_{23}"),
    sp.cos(q2+q3): sp.symbols(r"c_{23}"),
    sp.sin(q1+q2+q3): sp.symbols(r"s_{123}"),
    sp.cos(q1+q2+q3): sp.symbols(r"c_{123}")
}
# Get Coriolis and Centripetal Matrix (2nd row, 3rd column example)
# print(sp.latex(sp.simplify(C_q_qdot[1,2].subs(sh_subs))))

# Get the Potential energy using
# print(sp.latex(U_t.subs(sh_subs)))

# Get the Gravity vector using (2nd element example)
# print(sp.latex(G[1].subs(sh_subs)))

# Get the torque / effort using (2nd element example)
# print(sp.latex(tau[1].subs(sh_subs)))

# %% Question 3.2: Check whether M is symmetric
if M_t.T - M_t == sp.Matrix(np.zeros((3,3))):
    print("The mass matrix is symmetric (M = M.T)")

# %% Question 3.3: Check if M_dot - 2 * C is skew symmetric
ssm = sp.simplify(M_dot - 2 * C_q_qdot)
if sp.simplify(ssm.T + ssm) == sp.Matrix(np.zeros((3,3))):
    print(f"M_dot - 2C is skew symmetric")
else:
    print(f"M_dot - 2C is not skew symmetric")
    sum_val = sp.simplify(ssm + ssm.T).subs(sh_subs)
    try:
        display(sum_val)
```

```python
        # Get output using (2nd row, 3rd col as example)
        # print(sp.latex(ssm[1, 2].subs(sh_subs)))
    except:
        print(sum_val)

# %% C using cristoffel symbols
def cris_symb(i, j, k):
    def term(i, j, k):
        return M_t[i, j].diff(q[k])
    return term(i, j, k) + term(k, i, j) - term(k, j, i)

def c_ij(i, j):
    return (1/2) * ( \
            cris_symb(i, j, 0) * q_dot[0] + \
            cris_symb(i, j, 1) * q_dot[1] + \
            cris_symb(i, j, 2) * q_dot[2])


C_cris = sp.simplify(sp.Matrix([
    [c_ij(0, 0), c_ij(0, 1), c_ij(0, 2)],
    [c_ij(1, 0), c_ij(1, 1), c_ij(1, 2)],
    [c_ij(2, 0), c_ij(2, 1), c_ij(2, 2)],
]))
# Get output using (2nd row, 3rd col as example)
# print(sp.latex(C_cris[1, 2].subs(sh_subs)))
ssm_cris = sp.simplify(M_dot - 2 * C_cris)
if sp.simplify(ssm_cris.T + ssm_cris) == sp.Matrix(np.zeros((3,3))):
    print(f"M_dot - 2C_cris is skew symmetric")
else:
    print(f"M_dot - 2C_cris is not skew symmetric")
    display(sp.simplify(ssm_cris + ssm_cris.T).subs(sh_subs))
    # Get output using (2nd row, 3rd col as example)
    # print(sp.latex(ssm_cris[1, 2].subs(sh_subs)))
```

The output of this program is

The mass matrix is symmetric (M = M.T)
M_dot - 2C is not skew symmetric
M_dot - 2C_cris is skew symmetric

The program also displays the value of ssm.T + ssm, where ssm = $\dot{\mathbf{M}} - \mathbf{2C}$. Run this in VSCode Python Interactive for best results.

The matrix $\dot{\mathbf{M}} - \mathbf{2C}$ turns out to be

$$\text{ssm} = \dot{\mathbf{M}} - \mathbf{2C} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix} \tag{51}$$

Where

$$\begin{aligned}
S_{11} =& 2\dot{q}_2 l_1 l_2 m_3 s_2 + 2\dot{q}_2 l_1 m_2 r_2 s_2 + 2\dot{q}_3 l_2 m_3 r_3 s_3 + 2 l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right) \\
S_{12} =& \dot{q}_2 l_1 l_2 m_3 s_2 + \dot{q}_2 l_1 m_2 r_2 s_2 + 2\dot{q}_3 l_2 m_3 r_3 s_3 + l_1 m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right) \\
S_{13} =& m_3 r_3 \left(\dot{q}_3 l_2 s_3 + l_1 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right)
\end{aligned} \tag{52}$$

$$\begin{aligned}
S_{21} =& -2.0\dot{q}_1 l_1 l_2 m_3 s_2 - 2.0\dot{q}_1 l_1 m_2 r_2 s_2 - 2.0\dot{q}_1 l_1 m_3 r_3 s_{23} + 2.0\dot{q}_3 l_2 m_3 r_3 s_3 \\
S_{22} =& -\dot{q}_1 l_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) + 2\dot{q}_3 l_2 m_3 r_3 s_3 \\
S_{23} =& m_3 r_3 \left(-\dot{q}_1 l_1 s_{23} + \dot{q}_3 l_2 s_3\right)
\end{aligned} \tag{53}$$

$$\begin{aligned}
S_{31} =& -2.0 m_3 r_3 \left(\dot{q}_1 l_1 s_{23} + \dot{q}_1 l_2 s_3 + \dot{q}_2 l_2 s_3\right) \\
S_{32} =& -m_3 r_3 \left(1.0\dot{q}_1 \left(l_1 s_{23} + 2 l_2 s_3\right) + 2.0\dot{q}_2 l_2 s_3\right) \\
S_{33} =& -1.0 m_3 r_3 \left(\dot{q}_1 \left(l_1 s_{23} + l_2 s_3\right) + \dot{q}_2 l_2 s_3\right)
\end{aligned} \tag{54}$$

That is not a skew-symmetric matrix, however, using the coriolis matrix obtained using Cristoffel Symbols, we get the matrix $\dot{\mathbf{M}} - 2\mathbf{C}_{\text{cris}}$ as ssm_{cris}

$$\mathrm{ssm}_{\mathrm{cris}} = \dot{\mathbf{M}} - 2\mathbf{C}_{\mathrm{cris}} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \tag{55}$$

Where

$$\begin{aligned}
B_{11} =& 0 \\
B_{12} =& l_1 \left(2.0\dot{q}_1 l_2 m_3 s_2 + 2.0\dot{q}_1 m_2 r_2 s_2 + 2.0\dot{q}_1 m_3 r_3 s_{23} + 1.0\dot{q}_2 l_2 m_3 s_2 + 1.0\dot{q}_2 m_2 r_2 s_2 + \right. \\
& \left. 1.0\dot{q}_2 m_3 r_3 s_{23} + 1.0\dot{q}_3 m_3 r_3 s_{23}\right) \\
B_{13} =& m_3 r_3 \left(-\dot{q}_3 l_2 s_3 - l_1 s_{23} \left(\dot{q}_2 + \dot{q}_3\right) + 2.0 \left(l_1 s_{23} + l_2 s_3\right) \left(\dot{q}_1 + \dot{q}_2 + \dot{q}_3\right)\right) && (56) \\
B_{21} =& -l_1 \left(2\dot{q}_1 \left(l_2 m_3 s_2 + m_2 r_2 s_2 + m_3 r_3 s_{23}\right) + \dot{q}_2 l_2 m_3 s_2 + \dot{q}_2 m_2 r_2 s_2 + m_3 r_3 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right) \\
B_{22} =& 0 \\
B_{23} =& l_2 m_3 r_3 s_3 \left(2.0\dot{q}_1 + 2.0\dot{q}_2 + 1.0\dot{q}_3\right) && (57) \\
B_{31} =& -m_3 r_3 \left(2\dot{q}_1 \left(l_1 s_{23} + l_2 s_3\right) + 2.0\dot{q}_2 l_2 s_3 + \dot{q}_3 l_2 s_3 + l_1 s_{23} \left(\dot{q}_2 + \dot{q}_3\right)\right) \\
B_{32} =& -l_2 m_3 r_3 s_3 \left(2.0\dot{q}_1 + 2.0\dot{q}_2 + \dot{q}_3\right) \\
B_{33} =& 0 && (58)
\end{aligned}$$

This is a **skew-symmetric matrix** as its transpose is the negative of itself. This can be verified by the diagonal values being zero and the off diagonal terms being the negative of their transpose correspondence. Basically $\mathrm{ssm}_{\mathrm{cris}} + \mathrm{ssm}_{\mathrm{cris}}^{\top} = 0$.