# Robotic Fundamentals

## Serial and Parallel Robot Kinematics

23080549
23080538

January 16, 2025

# Contents

## 1. Lynxmotion Forward Kinematics

The Lynxmotion is a robotic arm with **five degrees of freedom (DoF)**, where all joints are revolute. The manipulator representation is shown in Fig. 1, with link frame assignments in the positions corresponding to the joints. Note that frame {0} (not shown) is coincident with frame {1} when $\theta_1$ is zero. We assume that the offset $d1$ has already been applied to frame {1}, and that the joint z-axis for joint frames {4} and {5} intersect at a common point.

For the forward kinematics of the joint spaces, we derive the DH parameters shown in (1), using the **proximal method** (Craig 2018). With the general form of ${}^{i-1}_{i}T$ where $\theta_i$ is the counter-clockwise rotation around the z-axis, we compute the following link transformations:

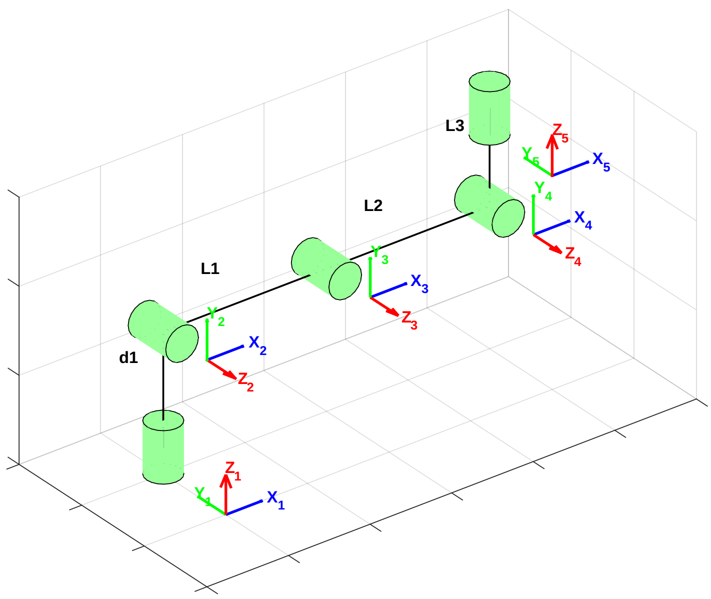


Figure 1: Frame assignments for the Lynxmotion arm

$$
{}^{0}_{1}T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

Note that $d1$ is placed on frame {1} for simplicity.

$$
{}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & L1 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{3}_{4}T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & L2 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{4}_{5}T = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & 1 & L3 \\ -s\theta_5 & -c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}
$$

For practicality, we use any of the following three conventions conversely: $\cos\theta_i$, $c\theta_i$ or $c_i$.

| Joint $i$ | $\alpha_{i-1}$ | $a_{i-1}$ | $d_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | 0 | 0 | $d1$ | $\theta_1$ |
| 2 | $\frac{\pi}{2}$ | 0 | 0 | $\theta_2$ |
| 3 | 0 | $L1$ | 0 | $\theta_3$ |
| 4 | 0 | $L2$ | 0 | $\theta_4$ |
| 5 | $-\frac{\pi}{2}$ | 0 | $L3$ | $\theta_5$ |

Table 1: Proximal Denavit-Hartenberg link parameters of the Lynxmotion

Where we have used the sum of angle formulas: $\psi = \theta_2 + \theta_3 + \theta_4$ and $\mu = \theta_5$

$$
{}^0_5T = {}^0_1T\,{}^1_2T\,{}^2_3T\,{}^3_4T\,{}^4_5T = \begin{bmatrix} c_\psi c_1 c_\mu - s_1 s_\mu & -c_\mu s_1 - c_\psi c_1 s_\mu & -s_\psi c_1 & p_x \\ c_1 s_\mu + c_\psi c_\mu s_1 & c_1 c_\mu - c_\psi s_1 s_\mu & -s_\psi s_1 & p_y \\ s_\psi c_\mu & -s_\psi s_\mu & c_\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}
$$

$$
p_x = c_1(L2c_{23} + L1c_2 - L3s_\psi) \tag{3}
$$
$$
p_y = s_1(L2c_{23} + L1c_2 - L3s_\psi) \tag{4}
$$
$$
p_z = d1 + L2s_{23} + L1s_2 + L3c_\psi \tag{5}
$$

From the forward kinematics equation (2), it is evident that $\mu$ and $\psi$ only impacts the orientation of the end-effector, whereas the rest of the joint angles impact the position $p_y, p_x, p_z$.

We define $d1 = 0.2m$, $L1 = 0.5m$, $L2 = 0.5m$, and $L3 = 0.2m$ and validate the system by testing the position equations using $\theta_i = 0$, resulting in $p_z = 0.4$ and $p_x = 1$, which are the expected co-ordinates of the end-effector given the initial frame configuration.

## 2. Lynxmotion Workspace

The following $\theta_i$ thresholds were chosen to avoid singularities (loss of DoF) in the system.

- $\theta_1$ $[-\pi, \pi]$, allowing full rotation of the base.
- $\theta_2$, $\theta_3$, $\theta_4 \notin [-\pi/2, \pi/2]$ and $\notin 0$, to prevent collinear with other links and full extension of the arm.
- $\theta_5 = 0$, we fix this one to 0 for the workspace calculation as it only affects the orientation of the end effector, it also simplifies the computation.

Figure 2 shows the **reachable workspace**, this was calculated iteratively using the forward kinematics equation (2) and substituting values of $\theta_i$ for the above ranges. The **dexterous workspace** is out scope from this work, in practice the workspace will depend how the arm is configured relative to it's base and the workspace would look more like a mushroom as a surface will obstruct motion.

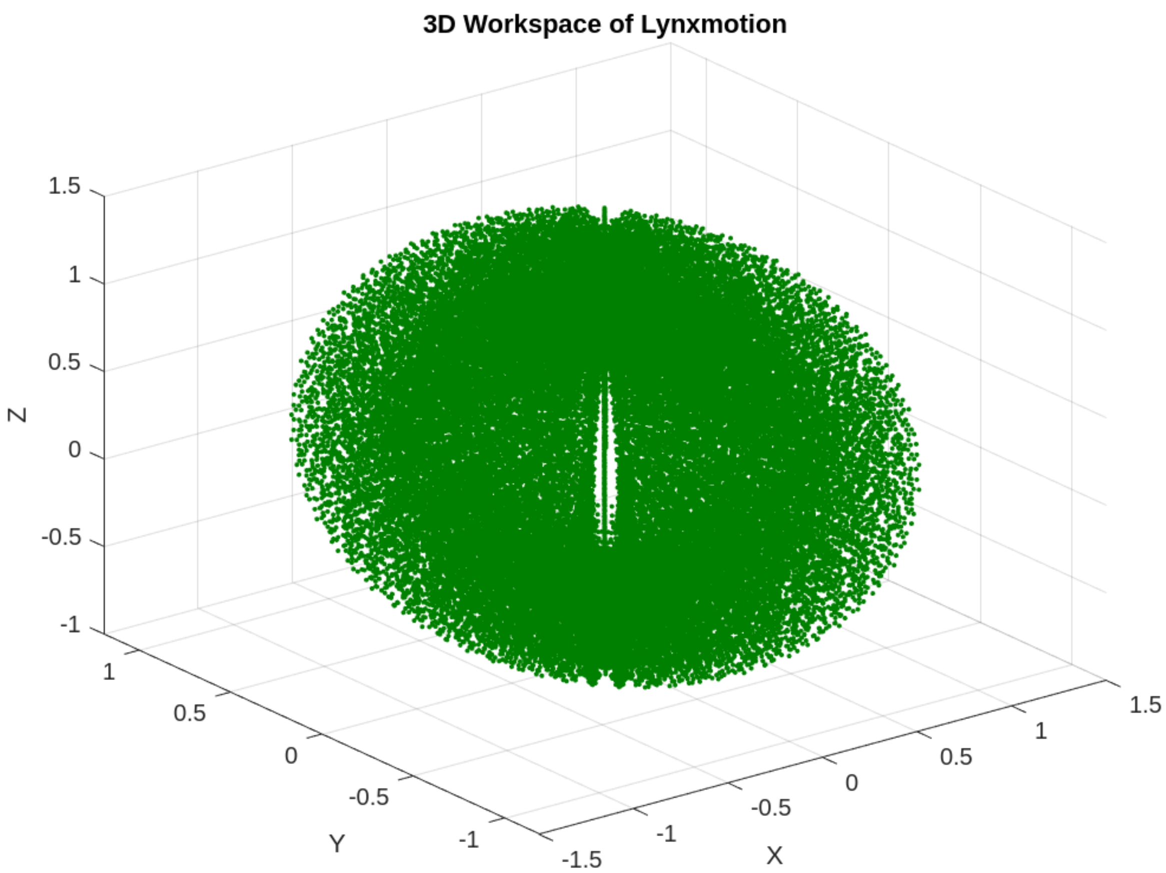


Figure 2: Reachable workspace with at least one degree of freedom

## 3. Lynxmotion Inverse Kinematics

In order to calculate the angles required at each joint to reach a desired position, the inverse kinematics of the system are derived using an analytical and geometrical approach. We start to calculate $\theta_1$ by restating equation (2) that puts dependence on the first joint by inverting ${}^0_1T$ such that we express it in terms of ${}^1_5T$.

$$[{}^0_1T(\theta_1)]^{-1}{}^0_5T = \begin{bmatrix} c\theta_1 & s\theta_1 & 0 & 0 \\ -s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & -d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\psi c_1 c_\mu - s_1 s_\mu & -c_\mu s_1 - c_\psi c_1 s_\mu & -s_\psi c_1 & p_x \\ c_1 s_\mu + c_\psi c_\mu s_1 & c_1 c_\mu - c_\psi s_1 s_\mu & -s_\psi s_1 & p_y \\ s_\psi c_\mu & -s_\psi s_\mu & c_\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^1_5T \tag{6}$$

Where ${}^1_5T$ is given by:

$${}^1_5T = \begin{bmatrix} c_\mu c_\psi & -s_\mu c_\psi & -s_\psi & p_x \\ s_\mu c_\mu s_1 & c_\mu & 0 & 0 \\ s_\psi c_\mu & -s_\psi s_\mu & c_\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Equating the (2,4) elements from equation (7) and equation (6), yields:

$$-s_1 p_x + c_1 p_y = 0 \tag{8}$$

Which can be written as:

$$\tan(\theta_1) = \frac{p_y}{p_x} \tag{9}$$

Therefore, $\theta_1$ is defined analytically as:

$$\theta_1 = Atan2(p_y, p_x) \tag{10}$$

To calculate $\theta_3$ we use a geometric approach, first we decompose the geometry of the arm into the side-view of the arm (Figure 3). Here we show the triangle formed by $L1$, $L2$, and a line between frame $\{2\}$ with the origin of frame $\{4\}$. The red-dotted line represents the mirror plane where the other solution for $\theta_3$ exists. Note that $r_e$ (end-effector) is equals to $\sqrt{p_x^2 + p_y^2}$.

We apply the law of cosines to solver for $\theta_3$:

$$(z_w - d1)^2 + r_w^2 = L1^2 + L2^2 - 2L1L2\cos(\pi + \theta_3) \tag{11}$$

Given that $\cos(\pi + \theta_3) = -\cos(\theta_3)$:

$$c_3 = \frac{(z_w - d1)^2 + r_w^2 - L1^2 - L2^2}{2L1L2} \tag{12}$$

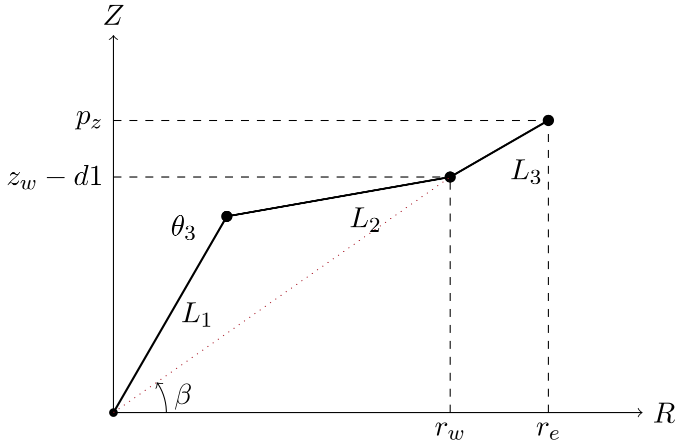


Figure 3: Side view geometry of Lynxmotion

This equation is solved for a value of $\theta_3$ that lies between 0 and $-\pi$, as it is only for these values that the triangle in Figure 3 exists. The other possible solution is found due to symmetry, where $\theta_3' = -\theta_3$.

From Figure 3 we can infer the following identities that relate $\psi$, $\theta_2$ and $\theta_3$ with $z_w$ and $r_w$:

$$r_w = r_e - L3\cos(\psi) \tag{13}$$
$$z_w = p_z - L3\sin(\psi) \tag{14}$$

Where $\psi$ can be obtained by equating ${}^0_5T(3,3)$:

$$\cos(\psi) = {}^0_5T(3,3) \tag{15}$$
$$\sin(\psi) = atan2(\pm\sqrt{1-\cos(\psi)^2}, \cos(\psi)) \tag{16}$$

When calculating for $\theta_3$, we noticed that the equations above only hold if we offset $\psi$ by $\pm\pi/2$ depending on the quadrant of the end-effector, we believe this is due to frame $\{5\}$ being offset by $\pi/2$. We developed the following algorithm to find the correct value of $\theta_3$:

**Algorithm 1** Calculate $\psi_{offset}$

```
if ⁰₅T(1,3) > 0 then
    if cos(ψ) < 0 then ψ_offset = ψ + π/2
    else ψ_offset = ψ − π/2
    end if
else
    if cos(ψ) < 0 then ψ_offset = ψ + π/2
    else ψ_offset = ψ − π/2
    end if
end if
```

We obtain the value of $\theta_3$ using equation (12) and using Atan2 to solve for the two real solutions:

$$s_3 = \pm\sqrt{(1 - c_3^2)} \tag{17}$$
$$\theta_3 = Atan2(s_3, c_3) \tag{18}$$

To solve for $\theta_2$, we find an expression for angle $\beta$ shown in Figure 3, where $\xi$ (not shown in the diagram) is the angle between the red line and the corresponding mirror triangle.

$$\beta = Atan2((z_w - d1), r_w) \tag{19}$$

Using law of cosines again to find $\xi$:

$$\cos\xi = \frac{(z_w - d1)^2 + r_w^2 + L1^2 - L2^2}{2L1\sqrt{(z_w - d1)^2 + r_w^2}} \tag{20}$$

In a similar manner here the arccosine of $\xi$ is between 0 and $\pi$ to preserve the geometry, therefore:

$$\theta_2 = \beta \pm \xi \tag{21}$$

Finally we can compute $\theta_4$ using the sum of the angle formulas we described previously:

$$\theta_4 = \psi - \theta_2 - \theta_3 \tag{22}$$

These are then validated by trying different $\theta_i$ values which can be substituted into the forward kinematic matrix 3.

# 4. Lynxmotion arm task

The lynxmotion arm was tasked with tracing a 'M', resulting in five individual cartesian co-ordinates of the end-effector.

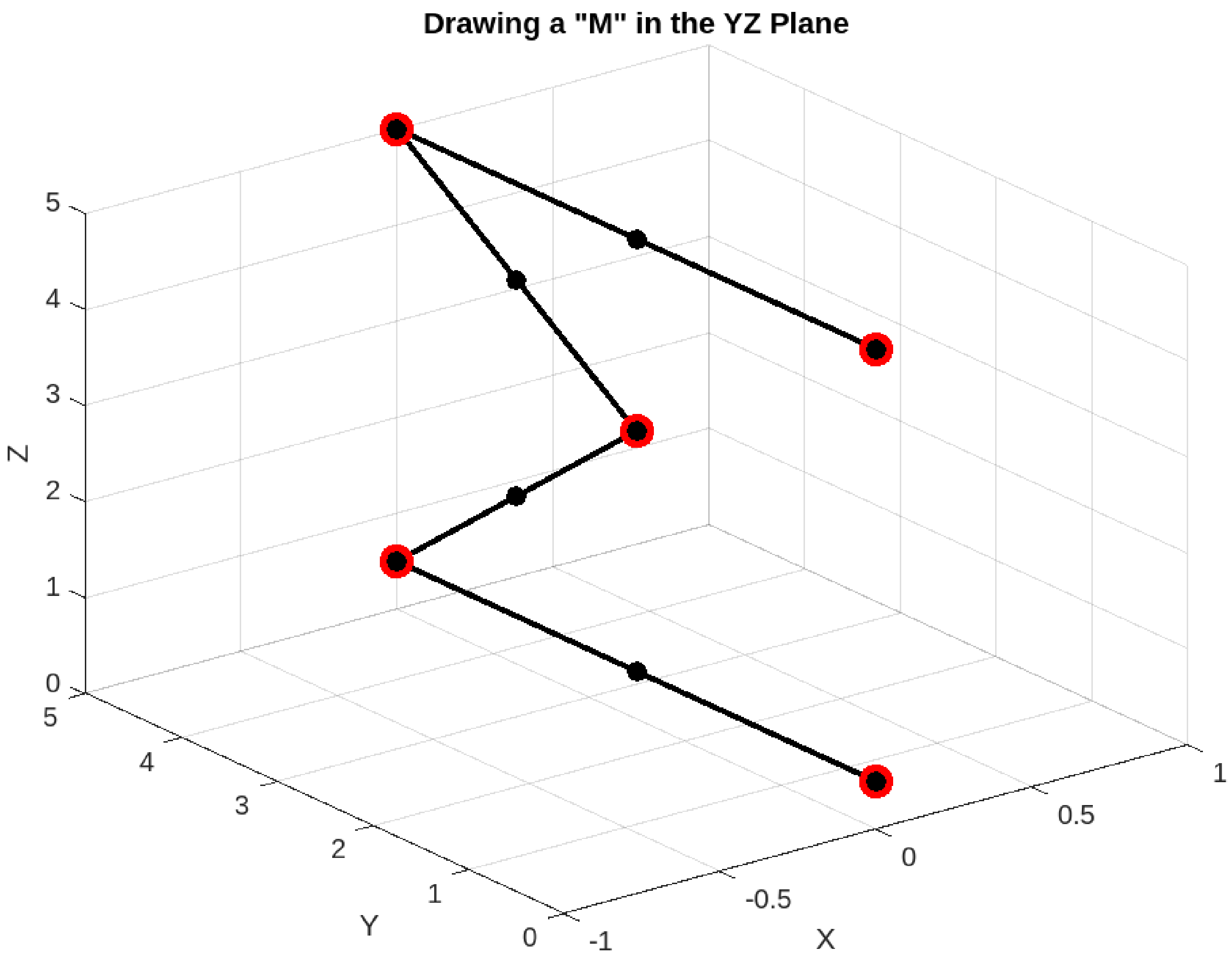


Figure 4: Visualization of task

The target coordinates for the end effector are in Cartesian coordinates (**X, Y, Z**).

$$M_{\text{points}} = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0.5 & 0.8 \\ 0 & 1 & 1.0 \\ 0 & 0 & 1.0 \end{bmatrix}$$

Using the method described in Section 3, the inverse kinematics for each point was calculated.

## 4.1 Trajectory path planning

Once the inverse kinematics for the end-effector of each point has been solved, three different trajectory paths between the points were chosen:

1. Free motion - Using parabolic blends.
2. Straight motion - Using cubic polynomials and via points.
3. Obstacle avoidance - Bug2 algorithm.

### 4.1.1 Free motion - Parabolic blends

For the parabolic blend calculations, we need two points to define the start and end of the motion. These are represented as $\theta_0$ (start) and $\theta_f$ (end). The motion between these points is divided into three phases: an acceleration phase, a constant velocity phase, and a deceleration phase. When multiple points are involved, this process is repeated for each consecutive pair of points. For example:

- M point 1 to 2
- M point 2 to 3
- M point 3 to 4
- M point 4 to 5

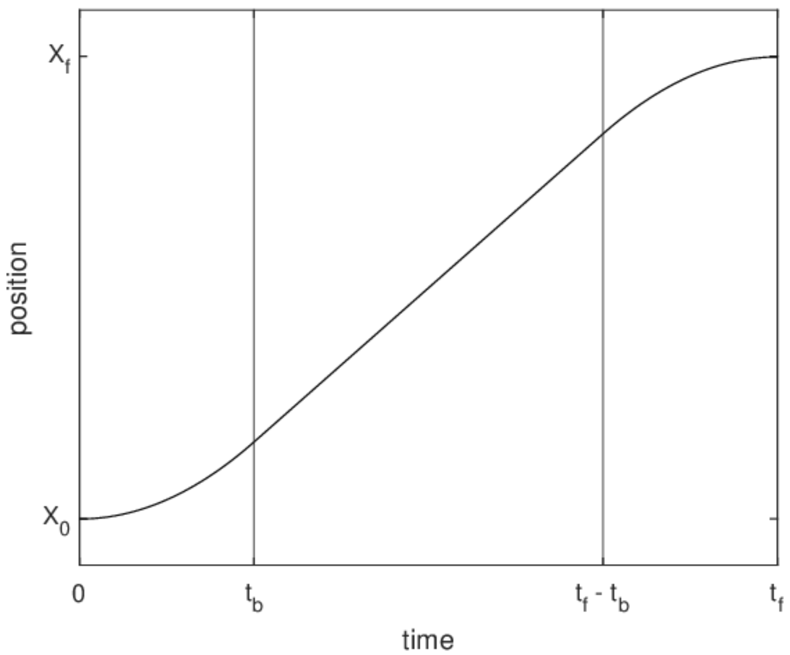


Figure 5: Parabolic blends example

The start of each section is normally denoted by $\theta_0$ and the final position as $\theta_f$.
The parameters needed to calculate the trajectory are:

- Start position: $\theta_0$ (from the inverse kinematics solution for point $m_1$).
- End position: $\theta_f$ (from the inverse kinematics solution for point $m_2$).
- Total time: $t = 2$ seconds for the motion between $m_1$ and $m_2$.
- Acceleration: $\ddot{\theta}$ (to be determined based on motion requirements).

Once the parameters are set we have to calculate the minimum acceleration $\ddot{\theta}$.
This can be calculated with the equation

$$\ddot{\theta} \geq \frac{4(\theta_f - \theta_0)}{t^2} \tag{23}$$

so

$$\ddot{\theta} \geq \frac{4(m2 - m1)}{2^2} \tag{24}$$

Next we calculate the blend time with the following equation.

$$t_b = \frac{\frac{t}{2} - \sqrt{\ddot{\theta}^2 t^2 - 4\ddot{\theta}(\theta_f - \theta_0)}}{2\ddot{\theta}} \tag{25}$$

so

$$t_b = \frac{\frac{t}{2} - \sqrt{\ddot{\theta}^2 2^2 - 4\ddot{\theta}(m2 - m1)}}{2\ddot{\theta}} \tag{26}$$

next we calculate the position at the end of the blend

$$\theta_b = \theta_0 + \frac{1}{2}\ddot{\theta}\ t_b^2 \tag{27}$$

so

$$\theta_b = M1 + \frac{1}{2}\ddot{\theta}\ t_b^2 \tag{28}$$

Once these calculations have been establish we need to calculate the trajectory for each phase of the motion.

- Acceleration phase:

  $$\theta(t) = \theta_0 + \frac{1}{2}\ddot{\theta}t^2 \tag{29}$$

  This is the initial blend

- constant velocity phase:

  $$\theta(t) = \theta_b + \ddot{\theta} \cdot t_b \cdot (t - t_b) \tag{30}$$

- deceleration phase:

  $$\theta(t) = \theta_f - \frac{1}{2}\ddot{\theta}(t - t_f)^2 \tag{31}$$

  This is the final blend

For each pair of consecutive points, the calculations above determine the smooth motion between them, ensuring smooth transitions in acceleration, constant velocity, and smooth deceleration. This process is repeated for each of the joints in the robot's arm, ensuring coordinated motion across all degrees of freedom.

The Figur below show the various joint positions of the robotic arm for each point of the 'M'.

trajectory_results.csv

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| | **Segment** | **Joint** | **Blend1_Start** | **Linear_Start** | **Blend2_Start** | **Blend1_End** | **Linear_End** | **Blend2_End** |
| | Number | Number | Number | Number | Number | Number | Number | Number |
| 1 | Segment | Joint | Blend1_Start | Linear_Start | Blend2_Start | Blend1_End | Linear_End | Blend2_End |
| 2 | 1 | 1 | 0 | 0.2618 | 1.309 | 0.2618 | 1.309 | 1.5708 |
| 3 | 1 | 2 | 2.6362 | 2.1969 | 0.4394 | 2.1969 | 0.4394 | 0 |
| 4 | 1 | 3 | -1.0654 | -0.8713 | -0.0945 | -0.8713 | -0.0945 | 0.0997 |
| 5 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 2 | 1 | 1.5708 | 1.5708 | 1.5708 | 1.5708 | 1.5708 | 1.5708 |
| 8 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 2 | 3 | 0.0997 | 0.2291 | 0.7467 | 0.2291 | 0.7467 | 0.8761 |
| 10 | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 3 | 1 | 1.5708 | 1.5708 | 1.5708 | 1.5708 | 1.5708 | 1.5708 |
| 13 | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 3 | 3 | 0.8761 | 0.8609 | 0.8005 | 0.8609 | 0.8005 | 0.7854 |
| 15 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 4 | 1 | 1.5708 | 1.309 | 0.2618 | 1.309 | 0.2618 | 0 |
| 18 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 4 | 3 | 0.7854 | 0.9163 | 1.4399 | 0.9163 | 1.4399 | 1.5708 |
| 20 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 4 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 6: Parabolic blends- Joint position results

### 4.2 Straight line motion with cubic polynomials and via points

Cubic polynomial trajectory planning is use to create smooth motions between two point with the help of via points.

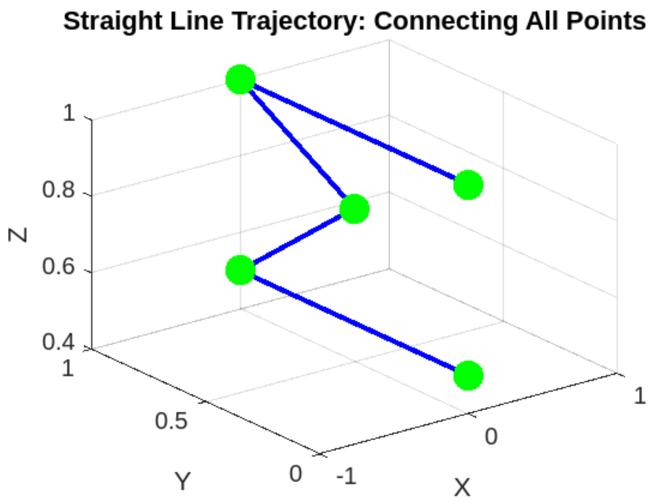


Figure 7: **Visualization of Straight line trajectories between M points**

To define the cubic polynomial trajectory, we need the following parameters:

- Start position: $\theta_0$ (from the inverse kinematics solution for point $m_1$).
- End position: $\theta_f$ (from the inverse kinematics solution for point $m_2$).
- Start velocity: $\dot{\theta}_0 = 0$ (assumes the motion begins from rest).
- End velocity: $\dot{\theta}_f = 0$ (assumes the motion ends at rest).
- Total time: $t_f = 2$ seconds (the total time to travel between $m_1$ and $m_2$).

The cubic polynomial equation for the joint position is expressed as:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Where:

- $a_0, a_1, a_2, a_3$ are coefficients determined by the boundary conditions.
- $\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2$ (velocity).
- $\ddot{\theta}(t) = 2a_2 + 6a_3 t$ (acceleration).

Boundary conditions (explain boundary conditions a bit better)
To solve for the coefficients $a_0, a_1, a_2, a_3$, we apply the following boundary conditions:

$$\begin{aligned}\theta(0) &= \theta_0,\\ \theta(t_f) &= \theta_f,\\ \dot{\theta}(0) &= \dot{\theta}_0,\\ \dot{\theta}(t_f) &= \dot{\theta}_f.\end{aligned}$$

Substituting these into the cubic polynomial, we derive the coefficients:

$$\begin{aligned}
a_0 &= \theta_0, \\
a_1 &= \dot{\theta}_0, \\
a_2 &= \frac{3(\theta_f - \theta_0)}{t_f^2} - \frac{2\dot{\theta}_0 + \dot{\theta}_f}{t_f}, \\
a_3 &= \frac{-2(\theta_f - \theta_0)}{t_f^3} + \frac{\dot{\theta}_0 + \dot{\theta}_f}{t_f^2}.
\end{aligned}$$

Substituting these into the cubic polynomial, we derive the coefficients:

$$\begin{aligned}
a_0 &= \theta_0, \\
a_1 &= \dot{\theta}_0, \\
a_2 &= \frac{3(\theta_f - \theta_0)}{t_f^2} - \frac{2\dot{\theta}_0 + \dot{\theta}_f}{t_f}, \\
a_3 &= \frac{-2(\theta_f - \theta_0)}{t_f^3} + \frac{\dot{\theta}_0 + \dot{\theta}_f}{t_f^2}.
\end{aligned}$$

**Motion Phases**
Once the coefficients are determined, the trajectory can be divided into three conceptual phases:

- **Acceleration Phase**: The motion begins with increasing velocity as dictated by the cubic equation and its derivative.
- **Constant Velocity Phase**: While not explicitly constant (due to the cubic nature), the middle of the motion tends to approximate uniform motion for shorter durations.
- **Deceleration Phase**: The motion slows as it approaches the target position, ensuring smooth stopping.

When moving through multiple points ($m_1 \to m_2 \to m_3$), the process is repeated for each segment. For trajectories passing through via points, continuity of position, velocity, and acceleration can be maintained by solving for coefficients across all segments.

### 4.2.1 Calculation for M points

Given:

$$\theta_0 = 0, \quad \theta_f = 90^\circ, \quad \dot{\theta}_0 = 0, \quad \dot{\theta}_f = 0, \quad t_f = 2 \text{ seconds}$$

The coefficients are calculated as:

$$\begin{aligned}
a_0 &= 0, \\
a_1 &= 0, \\
a_2 &= \frac{3(90 - 0)}{2^2} - \frac{0}{2} = 67.5, \\
a_3 &= \frac{-2(90 - 0)}{2^3} + \frac{0}{2^2} = -22.5.
\end{aligned}$$

Thus, the trajectory equation becomes:

$$\theta(t) = 0 + 0t + 67.5t^2 - 22.5t^3.$$

### 4.3 Obstacle Avoidance Task

A Lynxmotion robotic arm is being used to serve drinks to guests as part of a technical demonstration. Within the arm's workspace, there is a pillar extending from floor to ceiling, which represents the obstacle. As part of its task, the robotic arm must move its end effector from point $m_1(x_1, y_1)$ to point $m_2(x_2, y_2)$, without coming into contact with the pillar.

To ensure that the end effector avoids collision with the pillar, the Bug2 algorithm is used. This algorithm uses an m-line (motion line), which represents the desired straight-line path between $m_1$ and $m_2$. If the end effector detects the obstacle while following the m-line, it will trace the obstacle's boundary until it can resume its path along the m-line.

#### 4.3.1 M-Line Definition

To define the m-line, we first calculate its slope ($m$) using the slope-point formula:

$$m = \frac{y_2 - y_1}{x_2 - x_1}.$$

The m-line equation is then expressed in point-slope form:

$$y - y_1 = m(x - x_1),$$

where $(x_1, y_1)$ is a point on the line, $m$ is the slope, and $x, y$ are variables representing any point on the line.

For computational ease, we can convert this equation to the standard form:

$$Ax + By + C = 0,$$

where:

$$A = -(y_2 - y_1), \quad B = x_2 - x_1, \quad C = -(Ax_1 + By_1).$$

This form simplifies calculations, such as determining whether a point lies on the m-line or if the line intersects with the obstacle's boundary.

#### 4.3.2 Obstacle Representation - Pillar

The pillar is modeled as a circle in the $xy$-plane, as the robotic arm operates in two dimensions. The circle has a center at $(h, k)$ and a radius $r$, and is represented by the equation:

$$(x - h)^2 + (y - k)^2 = r^2.$$

#### 4.3.3 Intersection of M-Line and Cylinder

To find where the m-line intersects the circle, we substitute the m-line equation $y = mx + c$ (from point-slope form) into the circle equation:

$$(x - h)^2 + (mx + c - k)^2 = r^2.$$

Simplifying, we obtain a quadratic equation in terms of $x$:

$$(1 + m^2)x^2 + (2mc - 2hk)x + (h^2 + c^2 - 2kc + k^2 - r^2) = 0.$$

The solutions for $x$ are determined using the quadratic formula:

$$c = y1 - mx1$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

where:

$$a = 1 + m^2, \quad b = 2mc - 2hk, \quad c = h^2 + c^2 - 2kc + k^2 - r^2.$$

These $x$-values correspond to the points of intersection between the m-line and the circle. Substituting these $x$-values back into the m-line equation yields the corresponding $y$-coordinates.

### 4.4 Obstacle Avoidance and IK for joints

In order to validate the end effector path along the m-line, the IK for joint angles should be calculated. This will validate the trajectory produced by the Bug2 algorithm.

It would make sense to generate the joint value for the following points:

- $m_1$ starting/initial point
- From $m_1$ first intersection point with the cylinder.
- The bypass point along the obstacle's boundary.
- From the bypass point to $m_2$.

# 5. Planar Parallel Robot

## 5.1 Inverse Kinematics

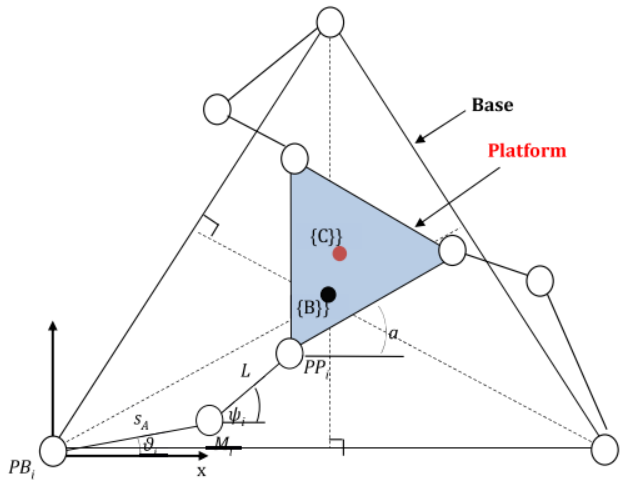


Figure 8: Parallel Robot Configuration

- Six passive revolute joints at $M_i$ and $PP_i$, where $i = 1, 2, 3$. Three active actuated joints at $PB_i$.
- A fixed base with centre {B} and a moving platform (shaded blue) with centre {C}.
- The base and platform are both equilateral triangles, where the side lengths can be calculated as $R\sqrt{3}$, where $R$ is the distance from the center to the vertices.
- $S = 170$ mm, $L = 130$ mm
- $r_{plat} = 130$ mm, $r_{base} = 290$ mm
- $a$ = angle platform offset

The parallel robot can be decomposed intro three distinct 3R serial manipulators, with their initial frame being attached at the base triangle vertices PB1, PB2 and PB3. The radius of the base triangle ($\overrightarrow{PB_iB}$) is given by $r_{base}$. The vertices are calculated using trigonometric relationships:

$$PB1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{32}$$

$$PB2 = \begin{bmatrix} r_{base}\sqrt{3} \\ 0 \\ 0 \end{bmatrix} \tag{33}$$

$$PB3 = \begin{bmatrix} r_{base}\frac{\sqrt(3)}{2} \\ \frac{3}{2}r_{base} \\ 0 \end{bmatrix} \tag{34}$$

$$\tag{35}$$

**The global coordinate frame $PB1$ is used to express the positions of all the attachment points and centers.**

The moving platform, with three attachment points PP1, PP2 and PP3 at its vertices - with centre {C} and radius ($\overrightarrow{PP_iC}$) given by $r_{plat}$. The platform's orientation is defined by a rotation angle $a$, relative to the x-axis. These vectors are calculated in the local frame of the platform:

$$\overrightarrow{CPP_1} = \begin{bmatrix} -r_{plat}\cos\frac{\pi}{6} \\ -r_{plat}\sin\frac{\pi}{6} \\ 0 \end{bmatrix} \tag{36}$$

$$\overrightarrow{CPP_2} = \begin{bmatrix} -r_{plat}\cos\frac{\pi}{6} + \frac{2\pi}{3} \\ -r_{plat}\sin\frac{\pi}{6} + \frac{2\pi}{3} \\ 0 \end{bmatrix} \tag{37}$$

$$\overrightarrow{CPP_3} = \begin{bmatrix} -r_{plat}\cos\frac{\pi}{6} + \frac{4\pi}{3} \\ -r_{plat}\sin\frac{\pi}{6} + \frac{4\pi}{3} \\ 0 \end{bmatrix} \tag{38}$$

Note that $\frac{\pi}{6}$ comes from the geometry of the equilateral triangle and the offsets $\frac{2\pi}{3}$ and $\frac{4\pi}{3}$ come from the increase in angle by 120° relative to the position of each serial arm.

We define frame {B} as:

$$B = \begin{bmatrix} r_{base}\frac{\sqrt{3}}{2} \\ \frac{r_{base}}{2} \\ 0 \end{bmatrix} \tag{39}$$

and frame {C} as:

$$C = \begin{bmatrix} x_c \\ y_c \\ 0 \end{bmatrix} \tag{40}$$

We then calculate $\overrightarrow{BPBi}$ as:

$$\overrightarrow{BPPi} = R_{BC}\overrightarrow{CPPi} + \overrightarrow{BC} \tag{41}$$

where $R_{bc}$ is given by:

$$R_{BC} = \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{42}$$

We proceed by calculating BPP1, BPP2 and BPP3 in terms of $x_c$ and $y_c$. This is followed by calculating $\overrightarrow{PB_iPP_i}$ which is given by:

$$\overrightarrow{PB_iPP_i} = \overrightarrow{BPP_i} - \overrightarrow{BPB_i} \tag{43}$$

Applying (43) and (40), give the following expressions for the base and platform points with regards to the world reference frame:

$$PB_1PP_1 = \begin{bmatrix} x_c + 65\sin(a) - 65\sqrt{3}\cos(a) \\ y_c - 65\cos(a) - 65\sqrt{3}\sin(a) \\ 0 \end{bmatrix} \tag{44}$$

$$PB_2PP_2 = \begin{bmatrix} x_c + 65\,\sin(a) + 65\,\sqrt{3}\,\cos(a) \\ y_c - 65\,\cos(a) + 65\,\sqrt{3}\,\sin(a) \\ 0 \end{bmatrix} \tag{45}$$

$$PB_3PP_3 = \begin{bmatrix} x_c - 130\,\sin(a) \\ y_c + 130\,\cos(a) \\ 0 \end{bmatrix} \tag{46}$$

At this stage, it is possible to calculate the points PP1, PP2 and PP3 however, to calculate $\theta_i$ we need to infer the value of the intermediate angle $\phi_i$, which is the angle of attachment relative to the platform's orientation, such that:

$$\phi_1 = a + \frac{\pi}{6} \tag{47}$$

$$\phi_2 = a + \frac{5\pi}{6} \tag{48}$$

$$\phi_3 = a + \frac{9\pi}{6} \tag{49}$$

Note that $\frac{\pi}{6}$ comes from the geometry of the equilateral triangle and the offsets $\frac{5\pi}{6}$ and $\frac{9\pi}{6}$ come from an increase in the angle of 120° and 240° due to the relative position of each serial arm with respect to the platform.

Therefore, using $\phi_i$ we can calculate $\theta_i$ by using the law of cosines:

$$c_i = Atan2\,(PB_{iy} - y_c - r_{\text{plat}}\sin(\phi_i), PB_{ix} - x_c - r_{\text{plat}}\cos(\phi_i)) \tag{50}$$

$$d_i = \arccos\left(\frac{S^2 - L^2 + (PB_{ix} - x_c - r_{\text{plat}}\cos(\phi_i))^2 + (PB_{iy} - y_c - r_{\text{plat}}\sin(\phi_i))^2}{2S\sqrt{(PB_{ix} - x_c - r_{\text{plat}}\cos(\phi_i))^2 + (PB_{iy} - y_c - r_{\text{plat}}\sin(\phi_i))^2}}\right) \tag{51}$$

$$\theta_i = c_i \pm d_i \tag{52}$$

For each leg, we apply the above equations, making sure that we account for the quadrant of each serial arm. This means that $\phi_i$ and $\theta_i$ in the equation above is offset by $\pi$ and $2\pi$ for arm 2 and 3 respectively (arm 1 does not need an offset).

We validate the value of $\theta_i$ above by calculating the points $\overrightarrow{PB_iM_i}$ and intermediate angle $\psi_i$. These can be derived geometrically as:

$$\overrightarrow{PB_1M_1} = \begin{bmatrix} S\sin(\theta_1) \\ S\cos(\theta_1) \\ 0 \end{bmatrix} \tag{53}$$

$$\overrightarrow{PB_2M_2} = \begin{bmatrix} S\cos(\theta_2) + 502.5 \\ S\sin(\theta_2) \\ 0 \end{bmatrix} \tag{54}$$

$$\overrightarrow{PB_3M_3} = \begin{bmatrix} 145\,\sqrt{3} + S\,\cos(\theta_3) \\ S\,\sin(\theta_3) + 435 \\ 0 \end{bmatrix} \tag{55}$$

$$\psi_i = Atan2(PP_{iy} - M_{iy}, PP_{ix} - M_{ix}) \tag{56}$$

Hence, if we have calculated $\theta_i$ correctly, the vectors $\overrightarrow{PB_iM_i}$ and $\overrightarrow{PP_iM_i}$ will intercept at the exact point. We demonstrate this in our results below, where Figures 9, 10 and 11 show different configurations of the parallel robot for varying values of $a$ and $y_c$, $x_c$.

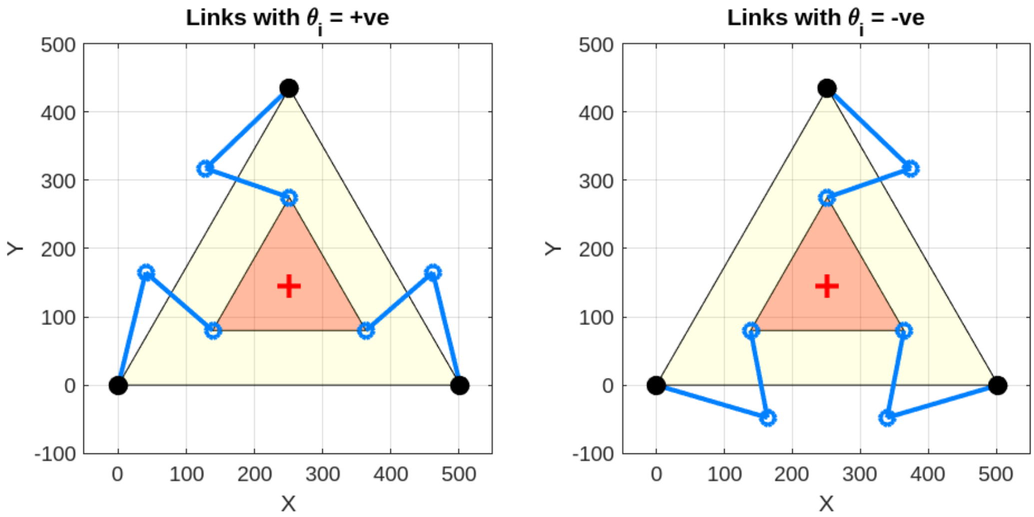


Figure 9: $a = 0$ and {C}={B}

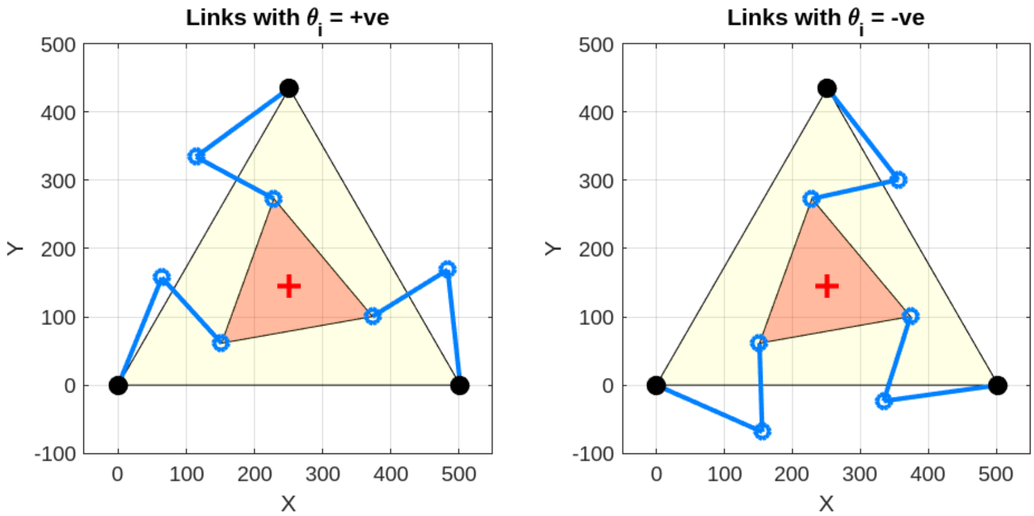


Figure 10: $a = \pi/18$ and {C}={B}

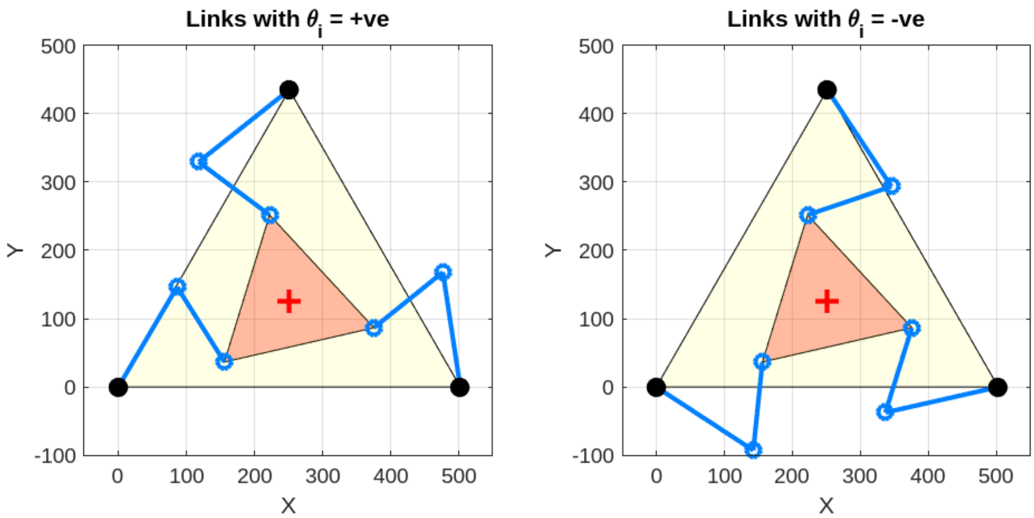


Figure 11: $a = \pi/14$ and $\{C\} = \{B\} - [0, -20, -0]^T$

## 5.2 Parallel Workspace

Firstly, we create a function to solve for the inverse kinematics of the system as shown in Algorithm 2. The purpose of this function is to calculate all **real angle** solutions that exist for given values of $x_c$, $y_c$ and $a$.

**Algorithm 2** Solve IK and Determine feasibility

```
Using inverse kinematics equations:
θ1 = c1 + d1,  −θ1 = c1 − d1
θ2 = π − (c2 + d2),  −θ2 = π − (c2 − d2)
θ3 = 2π − (c3 + d3),  −θ3 = 2π − (c3 − d3)

function SOLVEIK(xTry, yTry, a)
    Calculate joint angles using inverse kinematics equations
    Compute possible configurations:
    combos = [3 × 8]
    for each configuration in combos do
        if configuration is real and feasible then
            Store solution
        else
            Break
        end if
    end for
    return joint angles and feasibility
end function
```

Algorithm 3 is then used to calculate the workspace of the parallel robot for a given value of $a$. A crucial step in this process is the **filtering of the search grid**, which ensures that the workspace is confined within the robot's base. This refinement is justified by the requirements of medical applications, where the workspace is typically constrained to the base. Such a limitation facilitates better control and precise configuration of the end-effector, enhancing the system's safety and reliability (Merlet 2006).

**Algorithm 3** Calculate Workspace

```
Step 1: Choose a values for a
Step 2: Define search grid
x_min, x_max, y_min, y_max ← bounding box of base triangle
x_vals, y_vals ← 2D grid points within bounds
Filter points inside the base triangle using polygon checks
Step 3: Compute workspace
for each point (xTry, yTry) in grid do
    [θ_sol, feasible] ← SOLVEIK(xTry, yTry, a)
    if feasible then
        Store feasible point (xTry, yTry)
    end if
end for
Step 4: Plot workspace
```

Figures 12 and 13 show the results of the implementation for values of $a = 0$ and $a = \pi/14$ respectively. Note here is that we iterated over 625 points, and only plotted the points that have a real solution, where the end-effector can reach with at least one degree of freedom.

These results show that with increasing $a$, the workspace changes noticeably, indicating that platform rotations must be carefully optimised to ensure full worspace coverage. Balancing workspace,

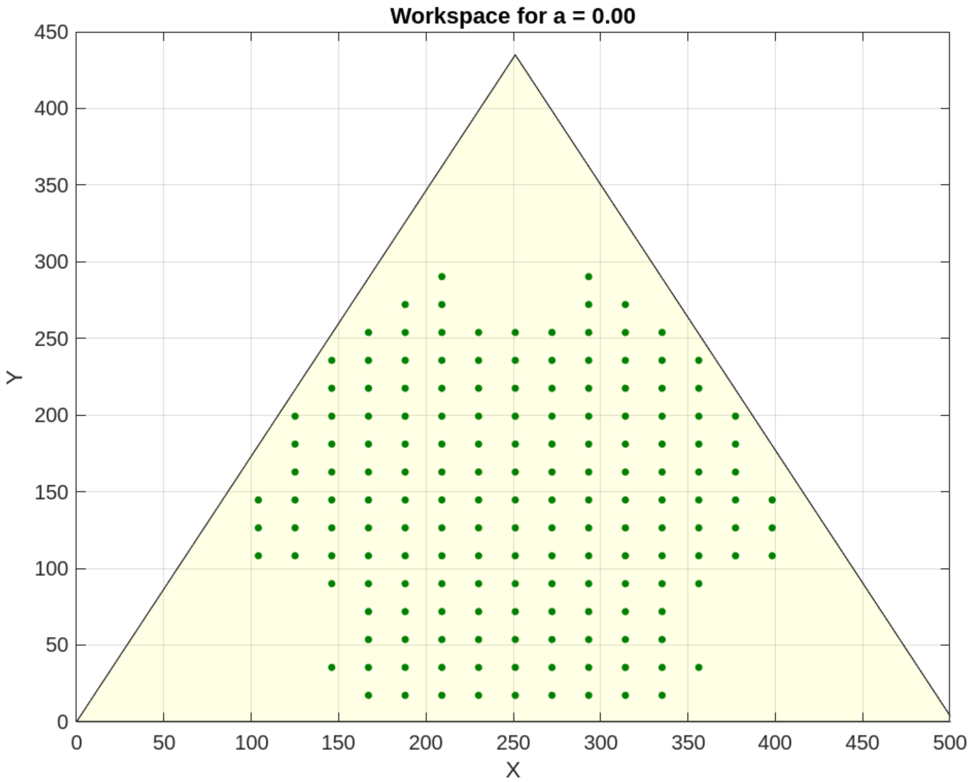


Figure 12: No rotation of platform

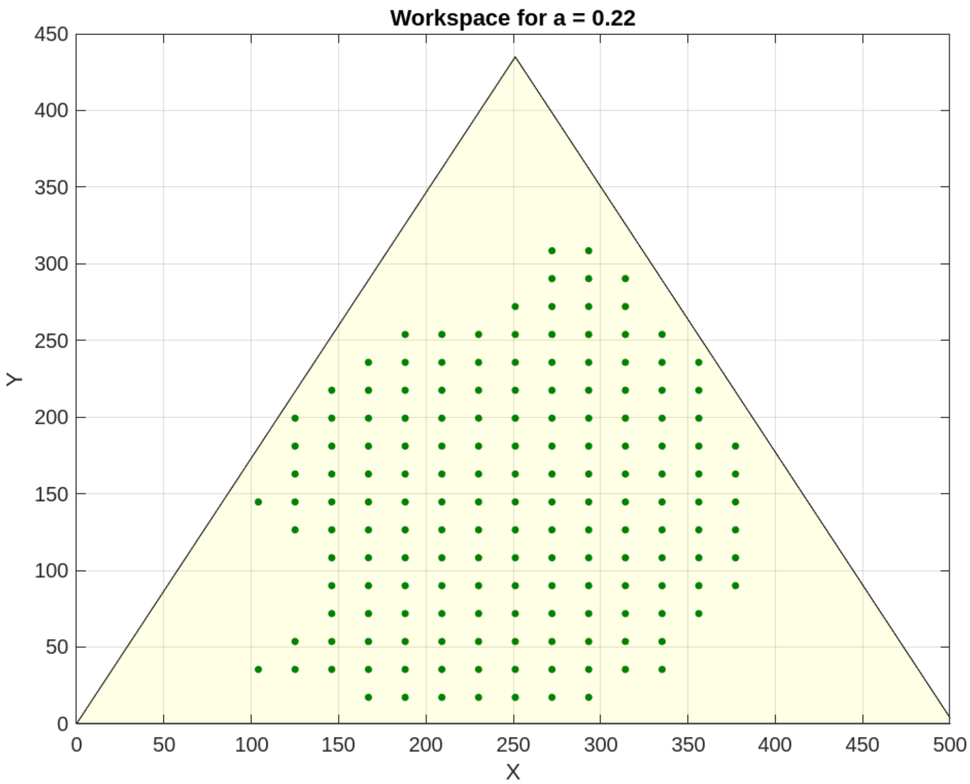


Figure 13: Slight rotation of platform

stiffness, and dexterity in the manipulator's design will depend significantly on the link lengths $S$ and $L$. While longer links may increase the reachable area, they could also reduce stiffness and make the system harder to control, potentially compromising precision.

## 6. Lynxmotion Dynamics

We use the recursive Newton-Euler method for the Lynxmotion arm under the influence of **gravity**. We would like to know (i) torque required to move the manipulator while it supports the **weight of a standard pint of beer, approximately 0.9kg**, and (ii) what a real world actuator for the task would look like, given the torque requirements.

We make the following assumptions:

- We have relaxed the problem to a planar 3R system, assuming no rotation of end-effector and independent of $\theta_1$, see Figure 14 that shows the new co-ordinate system.
- The centre of mass of the is located at vector $P_{C_i}$ for each link, which are assumed to be at the **midpoint of** $L_i$.
- Using Table 2, we assume each joint accelerations and velocities for **light weight manipulators** (Siciliano 2010).
- The beer can be thought of a point of mass acting on end-effector.
- We neglect friction forces and elasticity of the links.

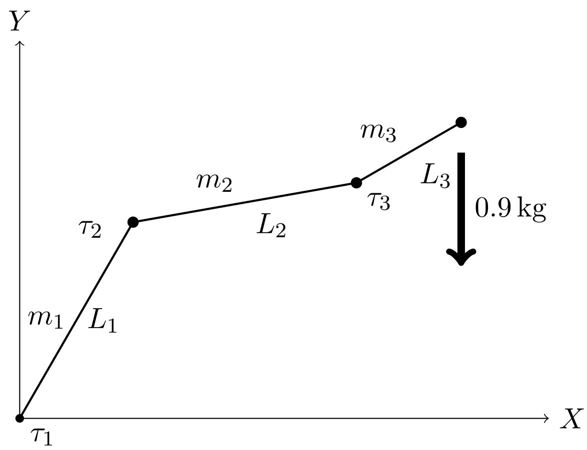


Figure 14: Planar system configuration

Table 2: Assumed Parameters

| Variable | Link 1 | Link 2 | Link 3 | Comments |
|---|---|---|---|---|
| Mass | 2 kg | 2 kg | 1 kg | Actual lynxmotion arms are lighter, but we assume a more robust build |
| Moment of Inertia | 0.5 $kgm^2$ | 0.5 $kgm^2$ | 0.25 $kgm^2$ | For planar cylinder |
| Joint Velocity | 2 $rad/s$ | 2 $rad/s$ | 2 $rad/s$ | (Siciliano 2010) |
| Joint Acceleration | 7 $rad/s^2$ | 7 $rad/s^2$ | 7 $rad/s^2$ | (Siciliano 2010) |

From (Craig 2018), we use the outward pass to calculate the linear and angular velocities and accelerations of each link starting from the base:

$$
\begin{aligned}
{}^{i+1}\boldsymbol{\omega}_{i+1} &=_i^{i+1} R\,\boldsymbol{\omega}_i + \dot{\theta}_{i+1}\,{}^{i+1}\hat{\mathbf{Z}}_{i+1}, && (7.10)\\
{}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} &=_i^{i+1} R\,\dot{\boldsymbol{\omega}}_i +_i^{i+1} R\,\boldsymbol{\omega}_i \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{\mathbf{Z}}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{\mathbf{Z}}_{i+1}, && (7.11)\\
{}^{i+1}\dot{\mathbf{v}}_{i+1} &=_i^{i+1} R\,(\dot{\boldsymbol{\omega}}_i \times \mathbf{P}_{i+1} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \mathbf{P}_{i+1})) + \mathbf{v}_i, && (7.12)\\
{}^{i+1}\dot{\mathbf{v}}_{C_{i+1}} &=^{i+1} \dot{\boldsymbol{\omega}}_{i+1} \times \mathbf{P}_{C_{i+1}} +^{i+1} \boldsymbol{\omega}_{i+1} \times \left({}^{i+1}\boldsymbol{\omega}_{i+1} \times \mathbf{P}_{C_{i+1}}\right) +^{i+1} \mathbf{v}_{i+1}. && (7.13)\\
{}^{i+1}\mathbf{F}_{i+1} &= m_{i+1}\,{}^{i+1}\mathbf{v}_{C_{i+1}}, && (7.14)\\
{}^{i+1}\mathbf{N}_{i+1} &= \mathbf{C}_{i+1}\mathbf{I}_{i+1}\,{}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} +^{i+1} \boldsymbol{\omega}_{i+1} \times \left(\mathbf{I}_{i+1}\,{}^{i+1}\boldsymbol{\omega}_{i+1}\right). && (7.15)
\end{aligned}
$$

Followed by the inward pass, that calculates the forces and torques acting on each link, starting from the end-effector:

$$
\begin{aligned}
{}^{i}\mathbf{f}_i &=_i^{i+1} R\,{}^{i+1}\mathbf{f}_{i+1} +^{i+1} \mathbf{F}_{i+1}, && (7.16)\\
{}^{i}\mathbf{n}_i &=_i^{i+1} R\,{}^{i+1}\mathbf{n}_{i+1} + \mathbf{P}_{C_{i+1}} \times^{i+1} \mathbf{F}_{i+1} + \mathbf{P}_{i+1} \times\, {}_i^{i+1}R\,{}^{i+1}\mathbf{f}_{i+1}, && (7.17)\\
\tau_i &=^i \mathbf{n}_i^T\,{}^i\hat{\mathbf{z}}_i. && (7.18)
\end{aligned}
$$

Refer to (ibid.) for a full description of these parameters.

Based on our assumptions, we can locate the centre of mass for each link which is given by:

$$P_{C_1} = \frac{L_1\hat{X}_1}{2} \quad (7.19)$$

$$P_{C_2} = \frac{L_2\hat{X}_2}{2} \quad (7.20)$$

$$P_{C_3} = \frac{L_3\hat{X}_3}{2} \quad (7.21)$$

The forces acting on the end-effector comes from the mass of the beer:

$$f_e = 0.9g\hat{Y}_e(N) \quad (7.22)$$

$$n_e = 0 \quad (7.23)$$

Initial conditions for the base:

$$\omega_0 = 0 \quad (7.24)$$

$$\dot{\omega_0} = 0 \quad (7.25)$$

$${}^0\dot{v_0} = g\hat{Y}_0 \quad (7.26)$$

Hence, for the outward pass, note that gravity is positive as it is an acceleration that the first link is experiencing "upwards":

$$
\begin{aligned}
{}^1\boldsymbol{\omega}_1 &= \dot{\theta}_1\,{}^1\hat{\mathbf{Z}}_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix}, \\
{}^1\dot{\boldsymbol{\omega}}_1 &= \ddot{\theta}_1\,{}^1\hat{\mathbf{Z}}_1 = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 \end{bmatrix}, \\
{}^1\mathbf{v}_1 &= \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} = \begin{bmatrix} gs_1 \\ gc_1 \\ 0 \end{bmatrix}, \\
{}^1\mathbf{v}_{C_1} &= \begin{bmatrix} 0 \\ 0.5L_1\dot{\theta}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} -0.5L_1\dot{\theta}_1^2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} gs_1 \\ gc_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5L_1\dot{\theta}_1^2 + gs_1 \\ 0.5L_1\ddot{\theta}_1 + gc_1 \\ 0 \end{bmatrix}, \\
{}^1\mathbf{F}_1 &= \begin{bmatrix} -0.5m_1L_1\dot{\theta}_1^2 + m_1gs_1 \\ 0.5m_1L_1\ddot{\theta}_1 + m_1gc_1 \\ 0 \end{bmatrix}, \\
{}^1\mathbf{N}_1 &= \begin{bmatrix} 0 \\ 0 \\ I_1\ddot{\theta}_1 \end{bmatrix}.
\end{aligned} \tag{7.27}
$$

Link 2:

$$
{}^2\boldsymbol{\omega}_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix},
$$

$$
{}^2\dot{\boldsymbol{\omega}}_2 = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 + \ddot{\theta}_2 \end{bmatrix},
$$

$$
{}^2\mathbf{v}_2 = \begin{bmatrix} s_2\sigma_2 - c_2\sigma_1 \\ s_2\sigma_1 + c_2\sigma_2 \\ 0 \end{bmatrix},
$$

where

$$
\begin{aligned}
\sigma_1 &= L_2(\dot{\theta}_1 + \dot{\theta}_2)^2 - s_1(L_1\ddot{\theta}_1 + gc_1) + c_1(L_1\dot{\theta_1}^2 - gs_1) \\
\sigma_2 &= c_2(L_1\ddot{\theta}_1 + gc_1) + s_1(L_1\dot{\theta_1}^2 - gs_1) + L_2(\ddot{\theta}_1 + \ddot{\theta}_2)
\end{aligned}
$$

$$
{}^2\mathbf{v}_{C_2} = \begin{bmatrix} s_1(L_1\theta_1 \ddot{+} gc_1) - 0.5L_2(\dot{\theta}_1 + \dot{\theta}_2)^2 - c_1(L_1\dot{\theta_1}^2 - gs_1) \\ c_1(L_1\theta_1 \ddot{+} gc_1) + 0.5L_2(\ddot{\theta}_1 + \ddot{\theta}_2) + s_1(L_1\dot{\theta_1}^2 - gs_1) \\ 0 \end{bmatrix},
$$

$$
{}^2\mathbf{F}_2 = m_2\,{}^2\mathbf{v}_{C_2},
$$

$$
{}^2\mathbf{N}_2 = \begin{bmatrix} 0 \\ 0 \\ I_2(\ddot{\theta}_1 + \ddot{\theta}_2) \end{bmatrix}. \tag{7.28}
$$

Link 3:

$$
{}^{3}\boldsymbol{\omega}_{3} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix},
$$

$$
{}^{3}\dot{\boldsymbol{\omega}}_{3} = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3 \end{bmatrix},
$$

$$
{}^{3}\mathbf{v}_{3} = \begin{bmatrix} s_3\,\sigma_2 - s_3\,\sigma_1 \\ s_3\,\sigma_1 + s_3\,\sigma_2 \\ 0 \end{bmatrix},
$$

where:

$$
\sigma_1 = s_2\,\sigma_4 - s_2\,\sigma_3 + L_3\,(\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3)^2,
$$

$$
\sigma_2 = s_2\,\sigma_4 + L_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) + c_2\,\sigma_3,
$$

$$
\sigma_3 = c_1\,\big(L_1\,\ddot{\theta}_1 + gc_1 + s_1\,\big(L_1\,\dot{\theta}_1^2 - gs_1\big) + L_2\,(\ddot{\theta}_1 + \ddot{\theta}_2),
$$

$$
\sigma_4 = L_2\,(\dot{\theta}_1 + \dot{\theta}_2)^2 - s_1\,\big(L_1\,\ddot{\theta}_1 + gc_1 + c_1\,\big(L_1\,\dot{\theta}_1^2 - gs_1\big),
$$

$$
{}^{3}\mathbf{v}_{C_3} = \begin{bmatrix} s_2\,\sigma_6 - c_2\,\sigma_5 - 0.5\,L_3\,(\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3)^2 \\ s_2\,\sigma_5 + 0.5\,L_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) + c_2\,\sigma_6 \\ 0 \end{bmatrix},
$$

where:

$$
\sigma_5 = L_2\,(\dot{\theta}_1 + \dot{\theta}_2)^2 - s_1\,\big(L_1\,\ddot{\theta}_1 + gc_1\big) + c_1\,\big(L_1\,\dot{\theta}_1^2 - gs_1\big),
$$

$$
\sigma_6 = c_1\,\big(L_1\,\ddot{\theta}_1 + gc_1\big) + s_1\,\big(L_1\,\dot{\theta}_1^2 - gs_1\big) + L_2\,(\ddot{\theta}_1 + \ddot{\theta}_2),
$$

$$
{}^{3}\mathbf{F}_{3} = m_3\,{}^{3}\mathbf{v}_{C_3},
$$

$$
{}^{3}\mathbf{N}_{3} = \begin{bmatrix} 0 \\ 0 \\ I_3(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) \end{bmatrix}. \tag{7.29}
$$

For the inward iteration we equate the force exerted from the pint glass $f_e$ over the end effector:

$$
{}^{3}f_3 = {}^{3}F_3 + \begin{bmatrix} 0 \\ 0.9g \\ 0 \end{bmatrix} \tag{7.30}
$$

The results for $\tau_i$ are as follows:

$$\tau_3 = 0.9\,L_3\,g + I_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)$$

$$+\,0.5\,L_3\,m_3\left[s_2\left(L_2\,(\dot{\theta}_1 + \dot{\theta}_2)^2 - s_1\,\sigma_8 + c_1\,\sigma_7\right) + 0.5\,L_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)\right.$$

$$\left.+\,c_2\left(c_1\,\sigma_8 + s_1\,\sigma_7 + L_2\,(\ddot{\theta}_1 + \ddot{\theta}_2)\right)\right],$$

where:

$$\sigma_7 = L_1\,\dot{\theta}_1^2 - g\,s_1,$$

$$\sigma_8 = L_1\,\ddot{\theta}_1 + g\,c_1,$$

$$\tau_2 = 0.9\,L_3\,g + I_2\,(\ddot{\theta}_1 + \ddot{\theta}_2) + I_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)$$

$$+\,0.9\,L_2\,g\,c_{23} + 0.25\,L_2^2\,m_2\,(\ddot{\theta}_1 + \ddot{\theta}_2) + L_2^2\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2)$$

$$+\,0.25\,L_3^2\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) + 0.5\,L_3\,g\,m_3\,\cos(2\theta_1 + \theta_2) + 0.5\,L_2\,g\,m_2\,\cos(2\theta_1) + L_2\,g\,m_3\,\cos(2\theta_1)$$

$$+\,0.5\,L_1\,L_2\,m_2\,\dot{\theta}_1^2\,s_1 + L_1\,L_2\,m_3\,\dot{\theta}_1^2\,s_1 - 0.5\,L_2\,L_3\,m_3\,\dot{\theta}_3^2\,s_2$$

$$+\,0.5\,L_1\,L_3\,m_3\,\ddot{\theta}_1\,c_{12} + 0.5\,L_1\,L_2\,m_2\,\ddot{\theta}_1\,c_1 + L_1\,L_2\,m_3\,\ddot{\theta}_1\,c_1$$

$$+\,L_2\,L_3\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2)\,c_2 + 0.5\,L_2\,L_3\,m_3\,\ddot{\theta}_3\,c_2$$

$$+\,0.5\,L_1\,L_3\,m_3\,\dot{\theta}_1^2\,s_{12} - L_2\,L_3\,m_3\,(\dot{\theta}_1\,\dot{\theta}_3 + \dot{\theta}_2\,\dot{\theta}_3)\,s_2,$$

$$\tau_1 = 0.9\,L_3\,g + I_1\,\ddot{\theta}_1 + I_2\,(\ddot{\theta}_1 + \ddot{\theta}_2) + I_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)$$

$$+\,0.9\,L_1\,g\,c_{123} + 0.9\,L_2\,g\,c_{23}$$

$$+\,0.25\,L_1^2\,m_1\,\ddot{\theta}_1 + L_1^2\,(m_2 + m_3)\,\ddot{\theta}_1 + 0.25\,L_2^2\,m_2\,(\ddot{\theta}_1 + \ddot{\theta}_2) + L_2^2\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2)$$

$$+\,0.25\,L_3^2\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) + 0.5\,L_1\,g\,(m_1 + 2m_2 + 2m_3)\,c_1 + 0.5\,L_3\,g\,m_3\,\cos(2\theta_1 + \theta_2)$$

$$+\,L_1\,L_2\,(m_2 + 2m_3)\,\ddot{\theta}_1\,c_1 + 0.5\,L_1\,L_3\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)\,c_{12} + L_2\,L_3\,m_3\,(\ddot{\theta}_1 + \ddot{\theta}_2)\,c_2$$

$$-\,0.5\,L_1\,L_2\,m_2\,\dot{\theta}_2^2\,s_1 - L_1\,L_2\,m_3\,\dot{\theta}_2^2\,s_1 - 0.5\,L_2\,L_3\,m_3\,\dot{\theta}_3^2\,s_2$$

$$-\,L_1\,L_3\,m_3\,(\dot{\theta}_1\,\dot{\theta}_2 + \dot{\theta}_1\,\dot{\theta}_3 + \dot{\theta}_2\,\dot{\theta}_3)\,s_{12}$$

$$-\,L_1\,L_2\,(m_2 + 2m_3)\,\dot{\theta}_1\,\dot{\theta}_2\,s_1 - L_2\,L_3\,m_3\,(\dot{\theta}_1\,\dot{\theta}_3 + \dot{\theta}_2\,\dot{\theta}_3)\,s_2. \tag{7.31}$$

Craig 2018, explains the terms in these equations as:

- **Inertial forces**: Terms with $\ddot{\theta}_i$ (e.g., $I_3(\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)$), are components of the $M$ matrix.
- **Gravitational forces**: Terms with $g$ (e.g., $0.9\,L_3\,g$), account for the weight of the links.
- **Centrifugal/Coriolis forces**: Terms with $\dot{\theta}_i^2$ or $\dot{\theta}_i\,\dot{\theta}_j$ (e.g., $L_2L_3m_3(\dot{\theta}_1+\dot{\theta}_2)$), represent velocity effects.

We then substitute values into $\tau_i$, at the fully stretched configuration $\theta_i = 0$, at which the arm experiences maximum torque due to gravity. At this singularity, the actuator selection meets worst-case requirements (Siciliano 2010). Using MATLAB we substitute into (7.31) and obtain the following max torques:

$$\tau_1 \approx 9.26\,(Nm) \tag{7.32}$$
$$\tau_2 \approx 40.28\,(Nm) \tag{7.33}$$
$$\tau_3 \approx 82\,(Nm) \tag{7.34}$$

Finally, we validate our results by exploring commercially available actuators capable of delivering the required torque. If the size and cost of these actuators are reasonable and practical, it provides greater confidence in the accuracy of our calculations.

Commercial actuators from (Pololu 2025) can satisfy the torque requirements, see example in Figure 15. It is possible to use gearboxes to increase the torque of the actuator by reducing speed. We noticed that at the velocities and accelerations specified in Table 2 only high-end industrial actuators can meet this requirement. Therefore, we conclude that our calculation is valid if we relax our velocity and acceleration assumptions.


Figure 15: Pololu 10(Nm) Actuator

## References

Craig, John J. (2018). *Introduction to robotics: mechanics and control*. Fourth edition. Section: viii, 438 pages : illustrations ; 24 cm. NY, NY: Pearson. viii, 438. ISBN: 978-0-13-348979-8.

Merlet, J.-P. (2006). *Parallel robots*. 2nd ed. Solid mechanics and its applications v. 128. Section: xix, 394 pages : illustrations ; 25 cm. Dordrecht, Netherlands: Springer. xix, 394. ISBN: 978-1-4020-4132-7 978-1-4020-4133-4.

Pololu (Jan. 15, 2025). *Pololu - 150:1 Metal Gearmotor 37Dx57L mm 24V (Helical Pinion)*. URL: `https://www.pololu.com/product/4687/specs` (visited on 01/15/2025).

Siciliano, Bruno (2010). *Robotics: modelling, planning and control*. Advanced textbooks in control and signal processing. Section: xxiv, 632 pages : illustrations ; 24 cm. London: Springer. xxiv, 632. ISBN: 978-1-84996-634-4 978-1-84628-641-4.

# 7. Appendix

```
syms theta1 theta2 theta3 theta4 theta5 d1 L1 L2 L3
T01 = dh_proximal(theta1, d1, 0, 0)
T12 = dh_proximal(theta2, 0, 0, pi/2)
T23 = dh_proximal(theta3, 0, L1, 0)
T34 = dh_proximal(theta4, 0, L2, 0)
T45 = dh_proximal(theta5, L3, 0, -pi/2)
%calculate forward kinematic matrix
T15 = simplify(T12 * T23 * T34 * T45)
T05 = simplify(T01 * T12 * T23 * T34 * T45)
% test
theta1_val = 0;
theta2_val = 0;
theta3_val = 0;
theta4_val = 0;
theta5_val = 0;
d1_val = 0.2;
L1_val = 0.5;
L2_val = 0.5;
L3_val = 0.2;
%test
T05_evaluated = subs(T05, {theta1, theta2, theta3, theta4, theta5, d1, L1,
    L2, L3},...
    {theta1_val, theta2_val, theta3_val, theta4_val, theta5_val, d1_val,
        L1_val, L2_val, L3_val})
```

Listing 1: Part1: FK Code

```
%joint limits
theta1_lim = linspace(-pi, pi, 36);
theta2_lim = linspace(-pi/2, pi/2, 12);
theta3_lim = linspace(-pi/2, pi/2, 12);
theta4_lim = linspace(-pi/2, pi/2, 12);

%link parameters
d1 = 0.2;
L1 = 0.5;
L2 = 0.5;
L3 = 0.2;

num_points = length(theta1_lim) * length(theta2_lim) * length(theta3_lim) *
    length(theta4_lim);
workspace_points = zeros(num_points, 3);
idx = 1;

%calculate positions iterate
for theta1 = theta1_lim
    for theta2 = theta2_lim
        for theta3 = theta3_lim
            for theta4 = theta4_lim

                x = cos(theta1)*(L1*cos(theta2+theta3)+L1*cos(theta2)-L3*sin
                    (theta2+theta3+theta4));
                y = sin(theta1)*(L2*cos(theta2+theta3)+L1*cos(theta2)-L3*sin
                    (theta2+theta3+theta4));
                z = d1+L2*sin(theta2+theta3)+L1*sin(theta2)+L3*cos(theta2+
                    theta3+theta4);
                workspace_points(idx, :) = [x, y, z];
                idx = idx + 1;

```

```
            end
        end
    end
end

figure;
scatter3(workspace_points(:,1), workspace_points(:,2), workspace_points(:,3), '.', 'MarkerEdgeColor', [0, 0.5, 0]);
title('3D Workspace of Lynxmotion');
xlabel('X'); ylabel('Y'); zlabel('Z');
filename = 'workspace.png';
exportgraphics(gcf, filename, 'ContentType', 'vector');
```

Listing 2: Part1: WS Code

```
syms theta1 theta2 theta3 theta4 theta5 d1 L1 L2 L3
T01 = dh_proximal(theta1, d1, 0, 0);
T12 = dh_proximal(theta2, 0, 0, pi/2);
T23 = dh_proximal(theta3, 0, L1, 0);
T34 = dh_proximal(theta4, 0, L2, 0);
T45 = dh_proximal(theta5, L3, 0, -pi/2);
T05 = simplify(T01 * T12 * T23 * T34 * T45)

%known position

theta1_val = 0;
theta2_val = pi;
theta3_val = pi/4;
theta4_val = 0;
theta5_val = 0; %have to fix this at 0 to calculate sin(psi)
d1_val = 0.2;
L1_val = 0.5;
L2_val = 0.5;
L3_val = 0.2; %TRY L3=0 (ALL OTHER AS 0)

T05_evaluated = subs(T05, {theta1, theta2, theta3, theta4, theta5, d1, L1, L2, L3},...
    {theta1_val, theta2_val, theta3_val, theta4_val, theta5_val, d1_val, L1_val, L2_val, L3_val})

%IK theta1 validation, should be pi/4, as per above
theta1_res = atan2(T05_evaluated(2,4), T05_evaluated(1,4))

%IK theta3 validation
psi = double(acos(T05_evaluated(3,3)))

%handle pos_location, total 4 different configurations
if T05_evaluated(1,3) < 0
    if T05_evaluated(3,3) > 0
        psi_offset = double(psi + pi/2)
    else
        psi_offset = double(psi - pi/2)
    end
else
    if T05_evaluated(3,3) < 0
        psi_offset = double(psi + pi/2)
    else
        psi_offset = double(psi - pi/2)
    end
end

```

```
zw = double( T05_evaluated(3,4) - (L3_val * [1 -1]*sqrt(1 - cos(psi_offset)
    ^2)) )
rw = double(sqrt(T05_evaluated(1,4)^2 + T05_evaluated(2,4)^2) - (L3_val *
    cos(psi_offset)))
top_part = ((zw - d1_val).^2) + (rw^2) - (L1_val^2) - (L2_val^2)
bot_part = 2 * L1_val * L2_val
cos_theta3 = top_part./bot_part

%initialize sin_theta3 as an array
sin_theta3 = NaN(size(cos_theta3));

%iterate through each cos_theta3 value
for i = 1:length(cos_theta3)
    cos_theta3_val = cos_theta3(i);
    if cos_theta3_val >= -1 && cos_theta3_val <= 1
        % sin_theta3 if cos_theta3 is valid
        sin_theta3(i) = sqrt(1 - cos_theta3_val^2);
    else
        fprintf('cos_theta3 at %d is out of bounds: %.2f\n', i, cos_theta3);
    end
end

%theta3 using atan2
theta3_pos = atan2(sin_theta3, cos_theta3); % positive solution
theta3_neg = atan2(-sin_theta3, cos_theta3); % negative solution

disp('theta3 positive solution (in radians):');
disp(double(theta3_pos));

disp('theta3 negative solution (in radians):');
disp(double(theta3_neg));

disp('thould be (in radians):');
disp(double(theta3_val));
```

Listing 3: Part1:IK Code

```
l1 = 1.0;
l2 = 1.0;
l3 = 0.5;


M_points = [
    0, 0, 0.5
    0, 5, 0.5;
    0, 2.5, 3;
    0, 5, 5;
    0, 0, 5
];

tb = 0.5;
tf = 2.0;
dt = 0.01;
time_vector = 0:dt:tf;


for i = 1:size(M_points, 1) - 1
    start_point = M_points(i, :);
    end_point = M_points(i + 1, :);


```

```
    theta0 = caljoints(start_point, l1, l2, l3);
    thetaf = caljoints(end_point, l1, l2, l3);

    fprintf('\nSegment %d: From M Point %d to %d\n', i, i, i + 1);
    fprintf('Joints      Start  0  ( ( 1 ,  2 ,  3 ,  4 ,  5 );       End f  ( 1 ,  2 ,  3 ,  4 ,  5 )\n');

    for joint = 1:5

        theta_start = theta0(joint);
        theta_end = thetaf(joint);
        theta_dot = (theta_end - theta_start) / (tf - tb);
        theta_ddot = theta_dot / tb;


        theta_traj = parabolic_blend(theta_start, theta_end, theta_dot, theta_ddot, tb, tf, dt);


        blend1_start = theta_traj(1);                     % Start of Blend 1
        blend1_end = theta_traj(find(time_vector == tb, 1)); % End of Blend 1
        linear_start = blend1_end;                        % Start of Linear
        linear_end = theta_traj(find(time_vector == (tf - tb), 1)); % End of Linear
        blend2_start = linear_end;                        % Start of Blend 2
        blend2_end = theta_traj(end);                     % End of Blend 2


        fprintf('  Joint %d   (%6.4f, %6.4f, %6.4f)        (%6.4f, %6.4f, %6.4f)\n', ...
            joint, blend1_start, linear_start, blend2_start, blend1_end, linear_end, blend2_end);
    end
end


function theta_traj = parabolic_blend(theta0, thetaf, theta_dot, theta_ddot, tb, tf, dt)
    t = 0:dt:tf;
    theta_traj = zeros(size(t));
    for i = 1:length(t)
        if t(i) <= tb

            theta_traj(i) = theta0 + 0.5 * theta_ddot * t(i)^2;
        elseif t(i) <= tf - tb

            theta_traj(i) = theta0 + theta_dot * (t(i) - tb / 2);
        else

            theta_traj(i) = thetaf - 0.5 * theta_ddot * (tf - t(i))^2;
        end
    end
end

function theta = caljoints(point, l1, l2, l3)
    x = point(1);
    y = point(2);
    z = point(3);


```

```
    theta1 = atan2(y, x);


    r = sqrt(x^2 + y^2);


    cos_theta2 = (r^2 + z^2 - l1^2 - l2^2) / (2 * l1 * l2);
    cos_theta2 = min(max(cos_theta2, -1), 1); % Clamp to valid range
    theta2 = acos(cos_theta2);
    theta3 = atan2(z, r) - theta2;


    theta4 = 0;
    theta5 = 0;
    theta = [theta1, theta2, theta3, theta4, theta5];
end
```

Listing 4: Part1: Trajectories

```
global x_c y_c a S L
syms L S theta1 theta2 theta3 a x_c y_c

r_base = 290; %(mm)
r_plat = 130;

%B centr of base
B = [r_base*sqrt(3)/2; r_base/2; 0]

%C centre of platform
C = [x_c; y_c; 0]


%test variables
S_val = 170;
L_val = 130;
x_c_val = r_base*sqrt(3)/2;
y_c_val = (r_base/2);
a_val = 0;

%FIRST LEG
%calculate the point M1
PB1 = [0;0;0];

M1 = PB1 + [S*cos(theta1);S*sin(theta1);0]

%BPP1 = Rbc*CPP1 + BC
CPP1 = [-r_plat*cos(pi/6);-r_plat*sin(pi/6);0];
BC = C-B;
%rot
Rbc = [cos(a) -sin(a) 0;sin(a) cos(a) 0;0 0 1];
BPP1 = Rbc*CPP1 + BC;

%calculate point PP1 (%BPP1-BPB1)
%PP1 = BPP1 - (PB1-B)
PP1 = BPP1 + B

%psi
psi1 = simplify(atan2(PP1(2,1)-M1(2,1), PP1(1,1)- M1(1,1)))

%phi
phi1 = a + pi/6;
```

```
c1 = atan2(y_c-r_plat*sin(phi1),x_c-r_plat*cos(phi1));
acos_arg1 = (S^2 - L^2 + (x_c - r_plat * cos(phi1))^2 + (y_c - r_plat * sin(phi1))^2) / ...
            (2 * S * sqrt((x_c - r_plat * cos(phi1))^2 + (y_c - r_plat * sin(phi1))^2));
d1 = acos(acos_arg1);

%phi
phi1_val = subs(phi1, a, a_val);

%theta1
theta1_1 = subs(c1+d1,[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
theta1_2 = subs(c1-d1,[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
double(theta1_1)
double(theta1_2)

%psi
psi_1 = subs(psi1, [x_c y_c a S L theta1],[x_c_val y_c_val a_val S_val L_val theta1_1]);
psi_2 = subs(psi1, [x_c y_c a S L theta1],[x_c_val y_c_val a_val S_val L_val theta1_2]);
double(psi_1)
double(psi_2)

%PP1
PP1_val = subs(PP1, [x_c y_c a], [x_c_val y_c_val a_val]);

%SECOND LEG
%calculate the point M1
PB2 = [r_base*sqrt(3);0;0];

M2 = PB2 + [S*cos(theta2);S*sin(theta2);0]

%BPP2 = Rbc*CPP2 + BC
CPP2 = [-r_plat*cos(pi/6 + 2*pi/3);-r_plat*sin(pi/6 + 2*pi/3);0]; %offset of 120 degrees
BC2 = C-B;
%rot
Rbc2 = [cos(a) -sin(a) 0;sin(a) cos(a) 0;0 0 1];
BPP2 = Rbc2*CPP2 + BC2;

%calculate point PP2 (%BPP2-BPB2)
PP2 = BPP2 + B

%psi
psi2 = simplify(atan2(PP2(2,1)-M2(2,1), PP2(1,1) - M2(1,1)))

%because of point chage relative to centre, we need to deduct pi
%phi
phi2 = a + 5*pi/6;
c2 = atan2(y_c-r_plat*sin(pi - phi2), r_base*sqrt(3)-x_c-r_plat*cos(pi - phi2));
acos_arg2 = (S^2 - L^2 + (r_base*sqrt(3)-x_c-r_plat*cos(pi - phi2))^2 + (y_c - r_plat * sin(pi - phi2))^2) / ...
            (2 * S * sqrt((r_base*sqrt(3)-x_c-r_plat*cos(pi - phi2))^2 + (y_c - r_plat * sin(pi - phi2))^2));
d2 = acos(acos_arg2);

%theta2 (off set of pi due to quadrant)
theta2_1 = subs(pi-(c2+d2),[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
```

```
theta2_2 = subs(pi-(c2-d2),[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
double(theta2_1)
double(theta2_2)

%psi2
psi2_1 = subs(pi-psi2, [x_c y_c a S L theta2],[x_c_val y_c_val a_val S_val L_val theta2_1]);
psi2_2 = subs(pi-psi2, [x_c y_c a S L theta2],[x_c_val y_c_val a_val S_val L_val theta2_2]);
double(psi2_1)
double(psi2_2)

%PP2
PP2_val = subs(PP2, [x_c y_c a], [x_c_val y_c_val a_val]);

%Third LEG
%calculate the point M3
PB3 = [r_base*sqrt(3)/2;r_base*3/2;0];

M3 = PB3 + [S*cos(theta3);S*sin(theta3);0]

%BPP3 = Rbc*CPP3 + BC
CPP3 = [-r_plat*cos(pi/6 + pi*4/3);-r_plat*sin(pi/6 + pi*4/3);0];
BC3 = C-B;
%rot
Rbc3 = [cos(a) -sin(a) 0;sin(a) cos(a) 0;0 0 1];
BPP3 = Rbc3*CPP3 + BC3

%calculate point PP3 (%BPP3-BPB3)
%PP3 = BPP3 - (PB3-B)
PP3 = simplify(BPP3 + B)

%psi3
psi3 = simplify(atan2(PP3(2,1)-M3(2,1), PP3(1,1) - M3(1,1)))

%phi (phi1 + 240 degrees due to gemotry)
phi3 = a + 9*pi/6;

c3 = atan2(r_base*3/2 - y_c- r_plat*sin(2*pi - phi3), r_base*sqrt(3)/2 - x_c - r_plat*cos(2*pi - phi3));
acos_arg2 = (S^2 - L^2 + (r_base*sqrt(3)/2 - x_c- r_plat*cos(2*pi - phi3))^2 + (r_base*3/2 - y_c - r_plat*sin(2*pi - phi3))^2) / ...
            (2 * S * sqrt((r_base*sqrt(3)/2 - x_c- r_plat*cos(2*pi - phi3))^2 + (r_base*3/2 - y_c - r_plat*sin(2*pi - phi3))^2));
d3 = acos(acos_arg2);

%test d3
%double(subs(d3,[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]))

%theta3 (off set of 2pi due to quadrant)
theta3_1 = subs(2*pi-(c3+d3),[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
theta3_2 = subs(2*pi-(c3-d3),[x_c y_c a S L],[x_c_val y_c_val a_val S_val L_val]);
double(theta3_1)
double(theta3_2)

%psi3
psi3_1 = subs(2*pi-psi3, [x_c y_c a S L theta3],[x_c_val y_c_val a_val S_val L_val theta3_1]);
```

```
psi3_2 = subs(2*pi-psi3, [x_c y_c a S L theta3],[x_c_val y_c_val a_val S_val
     L_val theta3_2]);
double(psi3_1)
double(psi3_2)

%PP3
PP3_val = subs(PP3, [x_c y_c a], [x_c_val y_c_val a_val]);

%end effector equilateral triangle
p_length = r_plat * sqrt(3);
p = nsidedpoly(3, 'Center', [x_c_val, y_c_val], 'SideLength', p_length);
p_rotated=rotate(p,rad2deg(a_val),[x_c_val, y_c_val]);

%validate
p_rotated.Vertices
double(PP1_val')
double(PP2_val')
double(PP3_val')


%calculate endpoints for both angles for M
x_end1 = S_val * cos(theta1_1);
y_end1 = S_val * sin(theta1_1);
x_end2 = S_val * cos(theta1_2);
y_end2 = S_val * sin(theta1_2);

x2_end1 = r_base*sqrt(3) - (S_val * cos(pi - theta2_1));
y2_end1 = S_val * sin(theta2_1);
x2_end2 = r_base*sqrt(3) - (S_val * cos(pi - theta2_2));
y2_end2 = S_val * sin(theta2_2);

x3_end1 = r_base*sqrt(3)/2 + (S_val * cos(2*pi - theta3_1));
y3_end1 = r_base*3/2 + S_val * sin(theta3_1);
x3_end2 = r_base*sqrt(3)/2 + (S_val * cos(2*pi - theta3_2));
y3_end2 = r_base*3/2 + S_val * sin(theta3_2);

%calculate startpoints for PP
x_pp_end1 = PP1_val(1,1) - L_val * cos(psi_1);
y_pp_end1 = PP1_val(2,1) - L_val * sin(psi_1);
x_pp_end2 = PP1_val(1,1) - L_val * cos(psi_2);
y_pp_end2 = PP1_val(2,1) - L_val * sin(psi_2);

x_pp2_end1 = PP2_val(1,1) + L_val * cos(psi2_1);
y_pp2_end1 = PP2_val(2,1) - L_val * sin(psi2_1);
x_pp2_end2 = PP2_val(1,1) + L_val * cos(psi2_2);
y_pp2_end2 = PP2_val(2,1) - L_val * sin(psi2_2);

x_pp3_end1 = PP3_val(1,1) - L_val * cos(psi3_1);
y_pp3_end1 = PP3_val(2,1) + L_val * sin(psi3_1);
x_pp3_end2 = PP3_val(1,1) - L_val * cos(psi3_2);
y_pp3_end2 = PP3_val(2,1) + L_val * sin(psi3_2);

%VALIDATE RESULTS all should equal the same
double(subs(M3, [S theta3], [S_val theta3_1]))
double([x3_end1; y3_end1])
double([x_pp3_end1; y_pp3_end1])
double(subs(M3, [S theta3], [S_val theta3_2]))
double([x3_end2; y3_end2])
double([x_pp3_end2; y_pp3_end2])
```

```
%base
b_length = r_base * sqrt(3);
b = nsidedpoly(3, 'Center', [B(1,1), B(2,1)], 'SideLength', b_length);

%plots
figure;

%axis limits
x_lim = [-50, 550];
y_lim = [-100, 500];

%bubplot 1: Link with theta1_1
subplot(1, 2, 1); % 1 row, 2 columns, first subplot
fill(p_rotated.Vertices(:, 1), p_rotated.Vertices(:, 2), 'red', 'FaceAlpha', 0.3);
hold on;
fill(b.Vertices(:, 1), b.Vertices(:, 2), 'yellow', 'FaceAlpha', 0.1);
plot(x_c_val, y_c_val, 'r+', 'MarkerSize', 10, 'LineWidth', 2); % Plot the center point
plot([0, x_end1], [0, y_end1], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 1-1
plot([x_pp_end1, PP1_val(1,1)], [y_pp_end1, PP1_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 1-2
plot([r_base*sqrt(3), x2_end1], [0, y2_end1], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 2-1
plot([x_pp2_end1, PP2_val(1,1)], [y_pp2_end1, PP2_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 2-2
plot([r_base*sqrt(3)/2, x3_end1], [r_base*3/2, y3_end1], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 3-1
plot([x_pp3_end1, PP3_val(1,1)], [y_pp3_end1, PP3_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 3-2
plot(0, 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); %origin points
plot(r_base*sqrt(3), 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k');
plot(r_base*sqrt(3)/2, r_base*3/2, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k');
title('Links with \theta_i = +ve');
xlabel('X');
ylabel('Y');
axis equal;
xlim(x_lim);
ylim(y_lim);
grid on;

%subplot 2: Link with theta1_2
subplot(1, 2, 2); % 1 row, 2 columns, second subplot
fill(p_rotated.Vertices(:, 1), p_rotated.Vertices(:, 2), 'red', 'FaceAlpha', 0.3);
hold on;
fill(b.Vertices(:, 1), b.Vertices(:, 2), 'yellow', 'FaceAlpha', 0.1);
plot(x_c_val, y_c_val, 'r+', 'MarkerSize', 10, 'LineWidth', 2); % Plot the center point
plot([0, x_end2], [0, y_end2], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 1
plot([x_pp_end2, PP1_val(1,1)], [y_pp_end2, PP1_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 2
plot([r_base*sqrt(3), x2_end2], [0, y2_end2], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 2-1
plot([x_pp2_end2, PP2_val(1,1)], [y_pp2_end2, PP2_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 2-2
plot([r_base*sqrt(3)/2, x3_end2], [r_base*3/2, y3_end2], '-o', 'LineWidth',
```

```
    2, 'Color', [0, 0.5, 1]); % Link 3-1
plot([x_pp3_end2, PP3_val(1,1)], [y_pp3_end2, PP3_val(2,1)], '-o', 'LineWidth', 2, 'Color', [0, 0.5, 1]); % Link 3-2
plot(0, 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); %origin points
plot(r_base*sqrt(3), 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); %origin points
plot(r_base*sqrt(3)/2, r_base*3/2, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k');
title('Links with \theta_i = -ve');
xlabel('X');
ylabel('Y');
axis equal;
xlim(x_lim);
ylim(y_lim);
grid on;

%filename = 'parallel-1.png';
%exportgraphics(gcf, filename, 'ContentType', 'vector');

%test validate answers
double([x_pp_end1 y_pp_end1])
double(subs(M1, [S theta1], [S_val theta1_1])')

double([x_pp2_end1 y_pp2_end1])
double(subs(M2, [S theta2], [S_val theta2_1])')

double([x_pp3_end1 y_pp3_end1])
double(subs(M3, [S theta3], [S_val theta3_1])')


%SOLVE IK
theta1_1_expr = c1+d1;
theta1_2_expr = c1-d1;
theta2_1_expr = pi-(c2+d2);
theta2_2_expr = pi-(c2-d2);
theta3_1_expr = 2*pi-(c3+d3);
theta3_2_expr = 2*pi-(c3-d3);

function [thetaSol, feasible] = solveIK(xTry, yTry, a_val, S_val, L_val, ...
                                        theta1_1_expr, theta1_2_expr, ...
                                        theta2_1_expr, theta2_2_expr, ...
                                        theta3_1_expr, theta3_2_expr)
    global x_c y_c a S L

    %calculate thetas
    t1_1 = subs(theta1_1_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);
    t1_2 = subs(theta1_2_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);

    t2_1 = subs(theta2_1_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);
    t2_2 = subs(theta2_2_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);

    t3_1 = subs(theta3_1_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);
    t3_2 = subs(theta3_2_expr, [x_c y_c a S L], [xTry yTry a_val S_val L_val]);

    %to double
```

```
    t1_1 = double(t1_1);
    t1_2 = double(t1_2);

    t2_1 = double(t2_1);
    t2_2 = double(t2_2);

    t3_1 = double(t3_1);
    t3_2 = double(t3_2);

    %check all possible configurations

    combos = [
       t1_1, t2_1, t3_1;
       t1_1, t2_1, t3_2;
       t1_1, t2_2, t3_1;
       t1_1, t2_2, t3_2;
       t1_2, t2_1, t3_1;
       t1_2, t2_1, t3_2;
       t1_2, t2_2, t3_1;
       t1_2, t2_2, t3_2
    ];

    feasible = false;
    thetaSol = [NaN, NaN, NaN];

    for i = 1:size(combos,1)
        tCandidate = combos(i,:);

        %check for imaginary parts
        if ~isreal(tCandidate) || any(imag(tCandidate) ~= 0)
            %skip not feasible
            continue;
        end

        %else it is possible config
        feasible = true;
        thetaSol = tCandidate;  %store solution
        break;                  %only first valid sol
    end

end


%WORKSPACE calculate
%STEP 1 calculate the search grid, in this case the base triangle
b.Vertices
%bounding box that covers the triangle fully
x_min = min(b.Vertices(:,1)) - 1;
x_max = max(b.Vertices(:,1)) + 1;
y_min = min(b.Vertices(:,2)) - 1;
y_max = max(b.Vertices(:,2)) + 1;

%2D grid
numSteps = 25;  %detail
x_vals = linspace(x_min, x_max, numSteps);
y_vals = linspace(y_min, y_max, numSteps);

%check if each point is in base triangle
[Xgrid, Ygrid] = meshgrid(x_vals, y_vals);
[in, on] = inpolygon(Xgrid, Ygrid, b.Vertices(:,1), b.Vertices(:,2));

```

```
%get points that are inside base platform
insideIdx = (in | on);  %boolean mask
xInside = Xgrid(insideIdx);
yInside = Ygrid(insideIdx);

%STEP 2 calculate IK for mesh points - IF THEY EXIST! (exclude imaginary)
a_val = pi/14;
%a_val = 0;

feasiblePoints = [];

for i = 1:numel(xInside)
    xTry = xInside(i);
    yTry = yInside(i);

    [thetaSol, feasible] = solveIK(xTry, yTry, a_val, S_val, L_val, ...
                                    theta1_1_expr, theta1_2_expr, ...
                                    theta2_1_expr, theta2_2_expr, ...
                                    theta3_1_expr, theta3_2_expr);
    if feasible
        feasiblePoints = [feasiblePoints; xTry, yTry];
    end
end

x_lim = [0, 500];
y_lim = [0, 450];

%STEP 3 plot workspace
figure;
axis equal;

fill(b.Vertices(:, 1), b.Vertices(:, 2), 'yellow', 'FaceAlpha', 0.1);
hold on;
plot(feasiblePoints(:,1), feasiblePoints(:,2), '.','MarkerEdgeColor', [0,
    0.5, 0], 'MarkerSize', 10);

xlabel('X'); ylabel('Y');
title(sprintf('Workspace for a = %.2f', a_val));
xlim(x_lim);
ylim(y_lim);
grid on;
filename = 'parallel-ws-2.png';
exportgraphics(gcf, filename, 'ContentType', 'vector');
```

Listing 5: Part2:Parallel Robot

```

%%%%%%%%%% OUTWARD PASS %%%%%%%%%%%%%%%%

function R_i1 = getRot_i1(theta)
    R_i1 = [
                cos(theta) sin(theta) 0;
                -sin(theta) cos(theta) 0;
                0           0          1;
    ];
end

function omega_i = calcAngularVelocity(omega_i, R_im1_i, dq_i, z_i)
    omega_i = R_im1_i * omega_i + dq_i * z_i;
end

```

```
function domega_i = calcAngularAcceleration(domega_im1, R_im1_i, ddq_i, z_i,
    dq_i, omega_i)
    domega_i = R_im1_i * domega_im1 ...
               + cross(R_im1_i * omega_i, dq_i * z_i) ...
              + ddq_i  * z_i ;
end

function v_i = calcLinearAcceleration(v_im1, R_im1_i, p_im1_i, domega_im1,
   omega_im1)
    v_i = R_im1_i * ( ...
            v_im1 ...
          + cross(domega_im1, p_im1_i) ...
          + cross(omega_im1, cross(omega_im1, p_im1_i)) ...
        );
end

function v_ci = calcMassAcceleration(v_i, domega_i, P_i_ci, omega_i)
    v_ci = v_i ...
         + cross(domega_i, P_i_ci) ...
         + cross(omega_i, cross(omega_i, P_i_ci));
end

function F_i = calcInertialForce(mi, v_ci)
    F_i = mi * v_ci;
end

function N_i = calcInertialTorque(I_i, omega_i, domega_i)
    N_i = I_i*domega_i...
        + cross(omega_i, I_i*omega_i);
end

%%%%%%%%%% INWARD PASS %%%%%%%%%%%%%%%%

function R_i = getRot(theta)
    R_i = [
                cos(theta) -sin(theta) 0;
                sin(theta) cos(theta) 0;
                0             0         1;
    ];
end

function f_i = calcForce(R_i, f_ip1, F_i)

    f_i = R_i * f_ip1 + F_i;
end

function n_i = calcTorque(N_i, R_i, n_ip1, P_c_i, F_i, P_i, f_ip1)

    n_i = N_i + R_i * n_ip1...
        + cross(P_c_i, F_i)...
        + cross(P_i, R_i * f_ip1);
end


syms theta1 theta2 theta3 theta1_dot theta2_dot theta3_dot theta1_dot_dot
   theta2_dot_dot theta3_dot_dot...
    I_1 I_2 I_3 m_1 m_2 m_3 L_1 L_2 L_3 g
n = 3; %n of links
q = [theta1; theta2; theta3];
dq = [theta1_dot ; theta2_dot; theta3_dot];
ddq = [theta1_dot_dot; theta2_dot_dot; theta3_dot_dot];
```

```
gravity = [0; g; 0];

v         = zeros(3, n);
omega     = zeros(3, n);
dv        = zeros(3, n);
domega    = zeros(3, n);

% assumption is only true if the base is not moving.
v0        = zeros(3,1) + gravity;
omega0    = zeros(3,1);
dv0       = zeros(3,1);
domega0 = zeros(3,1);

%outward Link 1
P12 = [L_1; 0; 0] %mass along x_axis {1} to {2}
Pc1 = [L_1/2; 0; 0]; %mass along the x-axis of frame {1}
z_1 = [0; 0; 1]; %revolute joint along z-axis
R_0 = getRot_i1(0); %there is no rotation between 0 and 1
R_1 = getRot_i1(theta1)
omega1 = calcAngularVelocity(omega0, R_1, dq(1), z_1)
domega1 = calcAngularAcceleration(domega0, R_1, ddq(1), z_1, dq(1), omega1)
v01 = calcLinearAcceleration(v0, R_1, [0;0;0], domega0, omega0) %0,0,0
    because {0} = {1}
v11 = calcLinearAcceleration(v01, R_1, P12, domega1, omega1)
vc1 = calcMassAcceleration(v01, domega1, Pc1, omega1)
F1 = calcInertialForce(m_1, vc1)
N1 = calcInertialTorque(I_1, omega1, domega1)

%test
subs(F1, [m_1 theta1 theta1_dot theta1_dot_dot L_1 g], [25 0 2 10 0.7 9.81])

%outward link 2
P23 = [L_2;0;0];
Pc2 = [L_2/2; 0; 0]; %mass along the x-axis of frame {2}
z_2 = [0; 0; 1]; %revolute joint along z-axis
R_2 = getRot_i1(theta2)
omega2 = calcAngularVelocity(omega1, R_2, dq(2), z_2)
domega2 = calcAngularAcceleration(domega1, R_2, ddq(2), z_2, dq(2), omega2)
v22 = simplify(calcLinearAcceleration(v11,R_2, P23, domega2, omega2))
vc2 = calcMassAcceleration(v11, domega2, Pc2, omega2)
F2 = calcInertialForce(m_2, vc2)
N2 = calcInertialTorque(I_2, omega2, domega2)

% Debugging v22
% subs(v22, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5])
% % Debugging vc2
% subs(vc2, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5])
% F2 = calcInertialForce(m_2, vc2);
% subs(F2, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5]) %WORKING

%outward link 3
P3E = [L_3;0;0];
Pc3 = [L_3/2; 0; 0]; %mass along the x-axis of frame {2}
z_3 = [0; 0; 1]; %revolute joint along z-axis
R_3 = getRot_i1(theta3);
```

```
omega3 = calcAngularVelocity(omega2, R_3, dq(3), z_3)
domega3 = calcAngularAcceleration(domega2, R_3, ddq(3), z_3, dq(3), omega3)
v33 = simplify(simplify(calcLinearAcceleration(v22,R_3, P3E, domega3, omega3
    )))
vc3 = simplify(calcMassAcceleration(v22, domega3, Pc3, omega3))
F3 = calcInertialForce(m_3, vc3);
N3 = calcInertialTorque(I_3, omega3, domega3)

% % Debugging v33
% subs(v33, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5])
% % Debugging v33
% subs(vc3, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5])
% F2 = calcInertialForce(m_2, vc2);
% subs(F3, [m_2, theta1, theta1_dot, theta1_dot_dot, L_1, L_2, theta2,
    theta2_dot, theta2_dot_dot], ...
%      [15, 0, 2, 10, 0.7, 0.6, 0, -4, 5]) %WORKING

%inward link 3
%in this case f3 = fe
fe = [0; 0.9*g; 0]; %sings flippled! for forces to equate to 0
Re = getRot(0);
R3 = getRot(theta3);
f3 = calcForce(R3, fe, F3);
n3 = simplify(calcTorque(N3, Re, [0;0;0], Pc3, F3, P3E, fe)); %[0,0,0] there
     is no n+1

%inward link 2
R2 = getRot(theta2);
f2 = calcForce(R2, f3, F2);
n2 = simplify(calcTorque(N2, R2, n3, Pc2, F2, P23, f3));

%inward link 3
R1 = getRot(theta1);
f1 = calcForce(R1, f2, F1);
n1 = simplify(calcTorque(N1, R1, n2, Pc1, F1, P12, f2));


%extract torque action on joints on z axis:
t1 = simplify(n1(3,1))
t2 = simplify(n2(3,1))
t3 = simplify(n3(3,1))

t3_torque = double(subs(t3, ...
    [theta1 theta2 theta3 theta1_dot theta2_dot theta3_dot theta1_dot_dot
       theta2_dot_dot theta3_dot_dot...
    I_1 I_2 I_3 m_1 m_2 m_3 L_1 L_2 L_3 g], ...
    [0, 0, 0, 2, 2, 2, 7, 7, 7, 0.5, 0.5, 0.25, 2, 2, 1, 0.5, 0.5, 0.2,
       9.81]))

t2_torque = double(subs(t2, ...
    [theta1 theta2 theta3 theta1_dot theta2_dot theta3_dot theta1_dot_dot
       theta2_dot_dot theta3_dot_dot...
    I_1 I_2 I_3 m_1 m_2 m_3 L_1 L_2 L_3 g], ...
    [0, 0, 0, 2, 2, 2, 7, 7, 7, 0.5, 0.5, 0.25, 2, 2, 1, 0.5, 0.5, 0.2,
       9.81]))

t1_torque = double(subs(t1, ...
```

```
    [theta1 theta2 theta3 theta1_dot theta2_dot theta3_dot theta1_dot_dot
        theta2_dot_dot theta3_dot_dot...
    I_1 I_2 I_3 m_1 m_2 m_3 L_1 L_2 L_3 g], ...
    [0, 0, 0, 2, 2, 2, 7, 7, 7, 0.5, 0.5, 0.25, 2, 2, 1, 0.5, 0.5, 0.2,
        9.81]))



% double(subs(f1, [theta1, theta2, theta1_dot, theta2_dot, theta1_dot_dot,
    theta2_dot_dot, ...
%     c1, c2, I_1, I_2, m_1, m_2, L_1, L_2], ...
%     [0, 0, 2, -4, 10, 5, ...
%     0.5, 0.3, 0.5, 0.2, 25, 15, 0.7, 0.6])) %working
```

Listing 6: Part3: Dynamics