Kinematics of the Roll-Tilt Mechanism in SHER-3.0

Botao Zhao

June 20, 2025

Contents

1 Introduction

The purpose of this document is to provide a quick and accessible understanding of the kinematics of the roll and tilt mechanism of SHER-3.0 (Eye Robot 3.0) for those who are new to the project. It is assumed that the reader is familiar with basic engineering kinematics, robotics, and linear algebra concepts. A separate document covering the kinematic derivation of the delta platform is also available at `https://github.com/zhaob5/delta_kinematics`. If you haven't read it yet, I strongly encourage you to review it before diving into this document, as some explanations here build upon concepts introduced there. Some related papers and scripts can be found on this github page: `https://github.com/zhaob5/roll_tilt_mechanism`

2 System Overview

The base of SHER-3.0 is a delta mechanism, which has been described in detail in our previous work. Mounted on top of the delta stage is an arm equipped with roll and tilt functionality. The tilt mechanism was designed with a four-bar linkage configuration, as illustrated in Fig.(2).

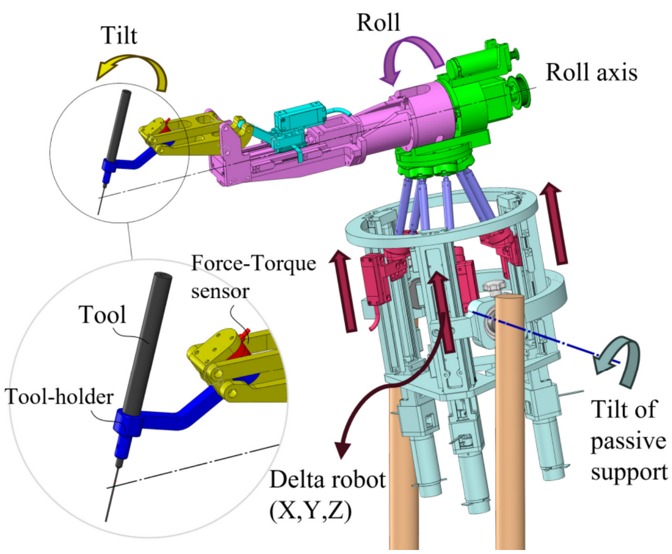

Figure 1: CAD models of the 5-DOF delta and roll-tilt mechanism

2.1 Tilting Mechanism

The structure of the tilt mechanism is depicted in Fig.(2). The joints A, B, C and D form a four-bar linkage mechanism. The joints A and B are fixed to the robot arm. This mechanism is actuated by an offset slider-crank mechanism with joints R, Q and A, where joint R is moved by a linear actuator. The rigid three-joint link QAD transfers the motion from the slider-crank to the four-bar. A tool-holder is mechanically fixed to the link CD and defines the offset towards the point P, which is the intended primary location for the RCM of the tool.

Since a four-bar mechanism **cannot provide a mechanically enforced RCM**, the point P shifts as the tool tilts. However, this motion can be actively compensated using the delta robot's degrees of freedom, thereby enabling a virtual RCM (**V-RCM**) despite the lack of a mechanically enforced pivot.

3 Four-Bar Linkage Mechanism

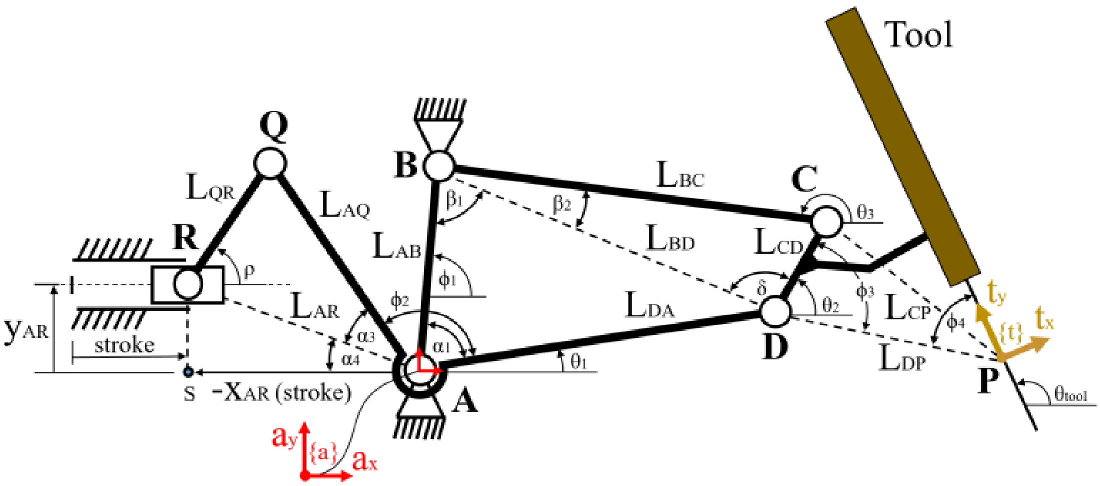

Figure 2: Side view of the tilt mechanism

3.1 Forward Kinematics

For the forward kinematics of this four-bar mechanism, the input is the linear actuator stroke length s, and the output is the tool angle θ_{tool}. Due to the nature of the four-bar linkage, the motion of the tool tip **P** involves both rotation and translation. However, these two motions are kinematically coupled — knowing one uniquely determines the other.

The forward kinematics is relatively straightforward to compute since the lengths of all the links are known. Link AB is fixed in space, and point P maintains a fixed position relative to link CD. For simplicity, we define point A as the origin of the coordinate system. As the stroke length $s \in [0, 50]$ is known:

$$x_{AR} = x_{AR_{max}} - s \tag{1}$$

where $x_{AR_{max}}$ is known based on the design parameter, and y_{AR} is also a fixed value, representing the height from the linear guide rail to the frame {a}.

Thus,

$$L_{AR} = \sqrt{x_{AR}^2 + y_{AR}^2} \tag{2}$$

$$\alpha_4 = 180^\circ - \arctan\left(y_{AR}, -x_{AR}\right) \tag{3}$$

As the lengths of AQ and QR are know, we can get α_3 from the law of cosine:

$$\alpha_3 = \arccos\left(\frac{L_{AR}^2 + L_{AQ}^2 - L_{QR}^2}{2L_{AR}L_{AQ}}\right) \tag{4}$$

Since ϕ_1 and ϕ_2 are angles with fixed values, and α_3, α_4 are known:

$$\theta_1 = 180^\circ - (\phi_2 + \alpha_3 + \alpha_4) \tag{5}$$

$$\alpha_1 = \phi_1 - \theta_1 \tag{6}$$

Form the law of cosine again:

$$L_{BD} = \sqrt{L_{AB}^2 + L_{DA}^2 - 2L_{AB}L_{DA}\cos(\alpha_1)} \tag{7}$$

$$\beta_1 = \arccos\left(\frac{L_{AB}^2 + L_{BD}^2 - L_{DA}^2}{2L_{AB}L_{BD}}\right) \tag{8}$$

$$\delta = \arccos\left(\frac{L_{BD}^2 + L_{CD}^2 - L_{BC}^2}{2L_{BD}L_{CD}}\right) \tag{9}$$

$$\theta_2 = 180^\circ - ((180^\circ - \beta_1 - \phi_1) + \delta) = \beta_1 + \phi_1 - \delta \tag{10}$$

Since ϕ_3 and ϕ_4 are also known:

$$\theta_{tool} = 180^\circ - \phi_4 - (\phi_3 - \theta_2) \tag{11}$$

As the lengths of AD and DP are constants, the x and y coordinates of point $\mathbf{P}$ in frame {a} can be directly computed:

$$P_x = \cos(\theta_1)L_{DA} + \cos(\theta_2 - \phi_3)L_{DP} \tag{12}$$

$$P_y = \sin(\theta_1)L_{DA} + \sin(\theta_2 - \phi_3)L_{DP} \tag{13}$$

To compute the transformation matrix from coordinate frame {a} to {t}, we can adopt a simplified approach by modeling links AD and DP as a 2-DOF serial robotic arm, but it is important to note that the joint angles in this representation are not independent.

Let $\theta = 90^\circ - \phi_4 - (\phi_3 - \theta_2)$, the angle between the x-axis of frame {a} and the x-axis of frame {t}:

$$T_{at} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & P_x \\ \sin(\theta) & \cos(\theta) & 0 & P_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

Thus, given the stroke length s, we can then determine both the tool's tilt angle and its position relative to frame {a}.

3.2 Inverse Kinematics

The inverse kinematics of this mechanism is somewhat more complex. Given the tool's tilt angle θ_{tool}, we can determine θ_2. However, if you substitute θ_2 back to the eq.(10) to compute θ_1, you will find that both β_1 and δ are unknown since the length of BD is also unknown. Solving for these variables would require handling a set of quadratic equations, which can be computationally expensive.

Thus, a more efficient approach is to derive the inverse kinematics directly from the geometry of the four-bar linkage. By translating edges AB and BC to form a new quadrilateral ABCE, as shown in Fig.(3):

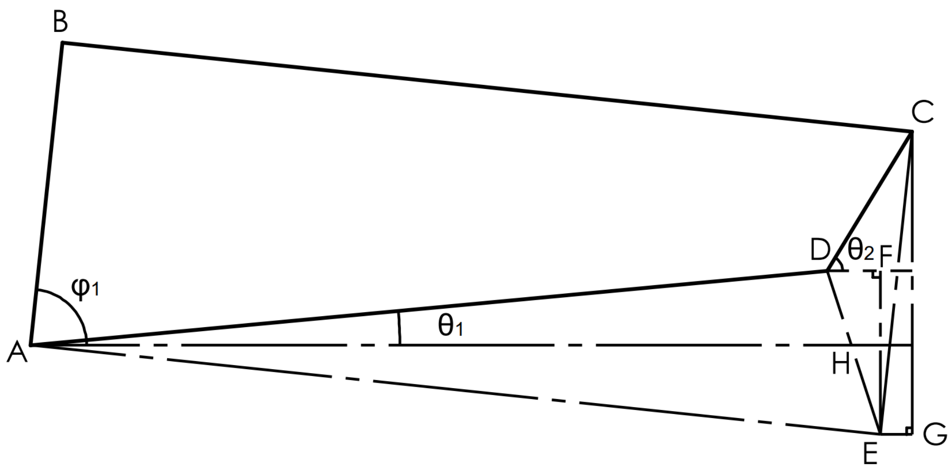

Figure 3: Close up of the four-bar mechanism

The goal is to determine θ_1 given the value of θ_2:

We know that $L_{CE} = L_{AB}$, $L_{AE} = L_{BC}$, $\angle CEG = \phi_1$

$$L_{DF} = \cos(\theta_2)L_{CD} - \cos(\phi_1)L_{AB} \tag{15}$$

$$L_{EF} = \sin(\phi_1)L_{AB} - \sin(\theta_2)L_{CD} \tag{16}$$

$$L_{DE} = \sqrt{L_{DF}^2 + L_{EF}^2} \tag{17}$$

Given the lengths of AD, AE, and DE, using the law of cosine:

$$\angle ADH = \arccos\left(\frac{L_{AD}^2 + L_{DE}^2 - L_{AE}^2}{2L_{AD}L_{DE}}\right) \tag{18}$$

$$\angle AHD = \angle FDH = \arctan(L_{EF}, L_{DF}) \tag{19}$$

$$\theta_1 = 180^\circ - \angle ADH - \angle AHD \tag{20}$$

Then, we can use θ_1 to calculate the stroke length s.

Let $\alpha = \alpha_3 + \alpha_4$:

$$\alpha = 180^\circ - \theta_1 - \phi_2 \tag{21}$$

$$x_{AR} = \cos(\alpha)L_{AQ} + \sqrt{L_{QR}^2 - (\sin(\alpha)L_{AQ} - y_{AR})^2} \tag{22}$$

$$s = x_{AR_{max}} - x_{AR} \tag{23}$$

3.3 Tool Tip Velocity

Calculating the tool tip velocity relative to frame {a} can also be quite challenging. If you attempt to take the time derivatives of eq.(12) and eq.(13), you'll find the process to be quite tedious, and the final expressions become unwieldy and difficult to interpret.

To address this, I use a technique known as the **Instantaneous Center of Velocity** method. If you've taken a course in mechanisms or dynamics, you may already be familiar with it. If not, the basic idea is that at any given instant, a rigid link in planar motion can be treated as rotating about a specific point in space, known as the instantaneous center. In our case, these points are labeled as **I** and **J**, as illustrated in Fig.(4).

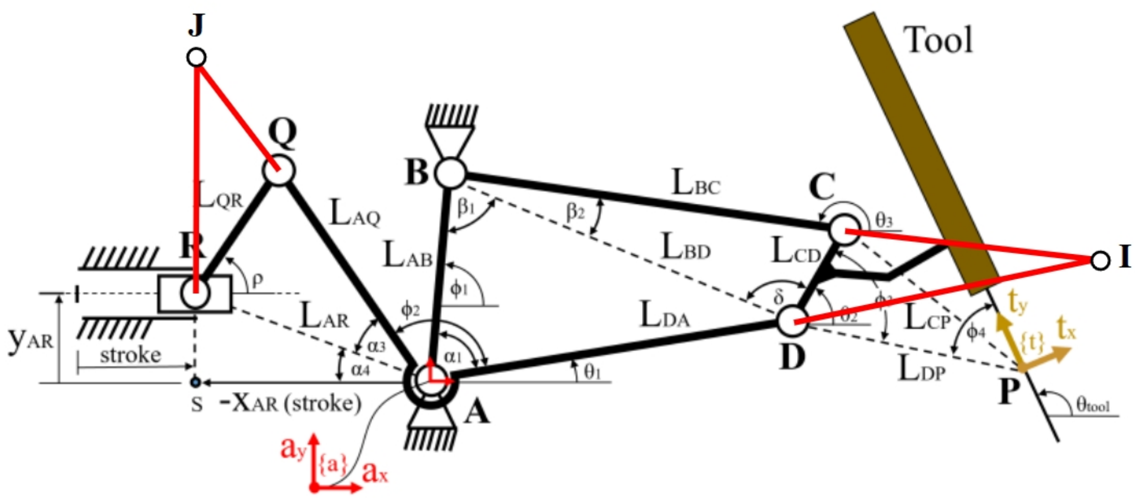

Figure 4: Velocity analysis for four-bar link and crank slider mechanism

For example, at any instant, link **QR** can be assumed to be in pure rotation with respect to the point **J**, where **QJ** is perpendicular to the velocity of point **Q** and **RJ** is perpendicular to the velocity of point **R**.

Then, we can get:

$$L_{JR} = \tan(\alpha) \cdot x_{AR} - y_{AR} \tag{24}$$
$$L_{JQ} = \sec(\alpha) \cdot x_{AR} - L_{AQ} \tag{25}$$

$$\dot{\rho} = \frac{\dot{s}}{L_{JR}} \tag{26}$$

$$L_{AQ} \cdot \dot{\alpha} = -L_{JQ} \cdot \dot{\rho} \tag{27}$$

$$\dot{\theta}_1 = \dot{\alpha} = -\frac{L_{JQ}}{L_{AQ}}\dot{\rho} \tag{28}$$

After determined $\dot{\theta}_1$, we want to find $\dot{\theta}_2$ and corresponding linear velocities $\dot{P}_x$ and $\dot{P}_y$.

Similarly, link **CD** can be assumed to be in pure rotation with respect to the point **I**, where **CI** is perpendicular to the velocity of point **C** and **DI** is perpendicular to the velocity of point **D**.

$$\beta_2 = \arccos\left(\frac{L_{BC}^2 + L_{BD}^2 - L_{CD}^2}{2L_{BC}L_{BD}}\right) \tag{29}$$

Let $\beta = \beta_1 + \beta_2$:

$$\angle AIB = 180^\circ - \alpha_1 - \beta \tag{30}$$

Then, apply the law of sine:

$$\frac{L_{AB}}{\sin(\angle AIB)} = \frac{L_{IA}}{\sin(\beta)} = \frac{L_{IB}}{\sin(\alpha_1)} \tag{31}$$

Since $\sin(180^\circ - \alpha_1 - \beta) = \sin(\alpha_1 + \beta)$:

$$L_{IA} = \frac{L_{AB} \cdot \sin(\beta)}{\sin(\alpha_1 + \beta)} \tag{32}$$

$$L_{ID} = L_{IA} - L_{DA} \tag{33}$$

Using Instantaneous Center method:

$$\dot{\theta}_1 L_{DA} = -\dot{\theta}_2 L_{ID} \tag{34}$$

$$\dot{\theta}_{tool} = \dot{\theta}_2 = -\frac{L_{DA}}{L_{ID}}\dot{\theta}_1 \tag{35}$$

From eq.(28) and eq.(26), we can get:

$$\dot{\theta}_{tool} = \frac{L_{DA}L_{JQ}}{L_{ID}L_{AQ}L_{JR}}\dot{s} \tag{36}$$

Noted that $L_{JQ}(s)$, $L_{ID}(s)$, and $L_{JR}(s)$ are not constants.

Thus eq.(36) has the form of: $\dot{\theta}_{tool} = f(s)\dot{s}$

Then, taking derivative to eq.(12) and eq.(13), we can get the linear velocity of the tool tip:

$$\dot{P}_x = -\sin(\theta_1)L_{DA}\dot{\theta}_1 - \sin(\theta_2 - \phi_3)L_{DP}\dot{\theta}_2 \tag{37}$$

$$\dot{P}_y = \cos(\theta_1)L_{DA}\dot{\theta}_1 + \cos(\theta_2 - \phi_3)L_{DP}\dot{\theta}_2 \tag{38}$$

We now obtain a clear and compact expression for the velocity of point **P**.

Similarly, known the tool tip rotational velocity $\dot{\theta}_{tool}$ and the slider position s, we can find the slider velocity $\dot{s} = f(s)^{-1}\dot{\theta}_{tool}$

4 Roll Mechanism

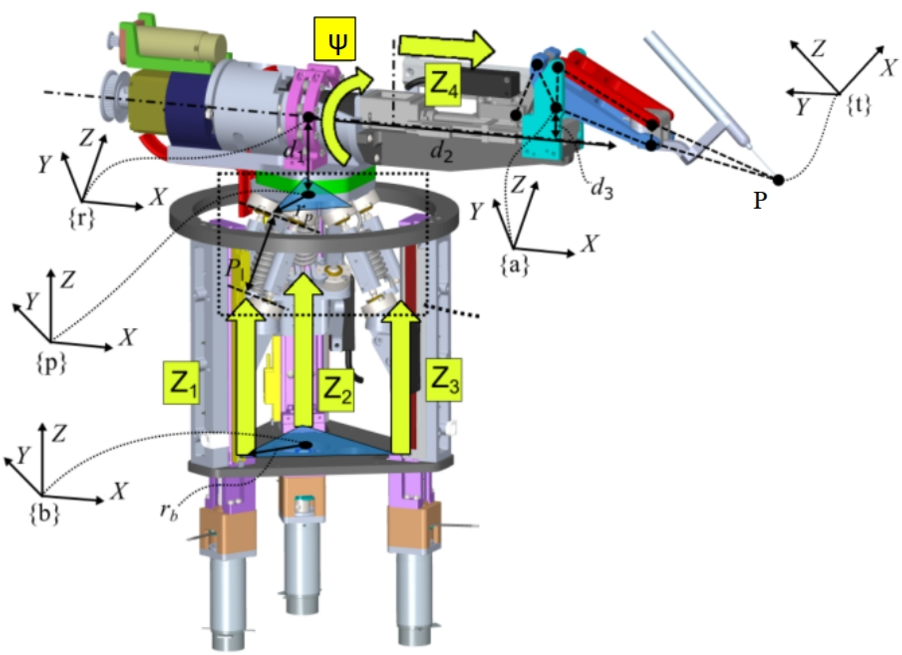

Figure 5: Kinematics diagram of the frames of SHER-3.0

The roll mechanism is relatively straightforward. As illustrated in Fig.(5), the roll angle ψ is defined with respect to the z-axis of frame {p}, and the rotational velocity $\dot{\psi}$ is directly contolled by the motor, with the positive direction defined along the x-axis of the base frame {b}. The roll axis is located d_1 mm above the delta platform {p}, and frame {a} is positioned d_3 mm above along the z-axis of frame {r}.

However, it is important to note that the orientations of frames {a} and {r} are different. In the home configuration, the vertical axis of frame {a} is y-axis, whereas the vertical axis of frame {r} is z-axis. Additional geometric details and frame assignments are shown in Fig.(6).

5 Tool Tip Velocity in Delta Platform Frame

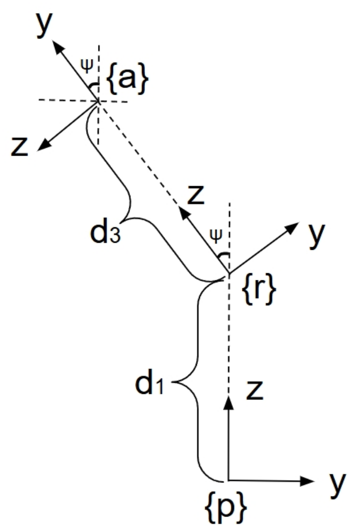

Figure 6: Front view of frames {p}, {r}, {a} of SHER-3.0

Recall that all previous kinematic derivations for the tool tip were expressed in frame {a}. To express the tool tip velocity in the delta platform frame {p}, we need to apply the following coordinate transformations:

First, define the homogeneous transformation matrix T_{pr}, which represents the transformation from frame frame {p} to frame {r}:

$$T_{pr} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) & 0 \\ 0 & \sin(\psi) & \cos(\psi) & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39}$$

Similarly, define the homogeneous transformation matrix T_{ra}, which represents the transformation from frame {r} to frame {a}:

$$T_{ra} = \begin{bmatrix} 1 & 0 & 0 & d_2 \\ 0 & \cos(90°) & -\sin(90°) & 0 \\ 0 & \sin(90°) & \cos(90°) & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{40}$$

Then, by multiplying the two matrices, we obtain the transformation matrix T_{pa}:

$$T_{pa} = T_{pr}T_{ra} = \begin{bmatrix} 1 & 0 & 0 & d_2 \\ 0 & -\sin(\psi) & -\cos(\psi) & -\sin(\psi)d_3 \\ 0 & \cos(\psi) & -\sin(\psi) & \cos(\psi)d_3 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{41}$$

Given the tool tip position $\mathbf{P}$ expressed in frame {a} as $\mathbf{P}_{\{a\}}$, we can compute its coordinates in frame{p}, denoted as $\mathbf{P}_{\{p\}} = T_{pa}\mathbf{P}_{\{a\}}$:

Since the four-bar linkage is a planar mechanism, the z-component of the tool tip position in frame {a}, $P_{z\{a\}}$ is always zero.

$$\begin{bmatrix} P_{x\{p\}} \\ P_{y\{p\}} \\ P_{z\{p\}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_2 \\ 0 & -\sin(\psi) & -\cos(\psi) & -\sin(\psi)d_3 \\ 0 & \cos(\psi) & -\sin(\psi) & \cos(\psi)d_3 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{x\{a\}} \\ P_{y\{a\}} \\ 0 \\ 1 \end{bmatrix} \tag{42}$$

$$\begin{bmatrix} P_{x\{p\}} \\ P_{y\{p\}} \\ P_{z\{p\}} \end{bmatrix} = \begin{bmatrix} P_{x\{a\}} + d_2 \\ -\sin(\psi)P_{y\{a\}} - \sin(\psi)d_3 \\ \cos(\psi)P_{y\{a\}} + \cos(\psi)d_3 + d_1 \end{bmatrix} \tag{43}$$

By taking the time derivative of eq.(43), we obtain the expression for the tool tip velocity in frame {p}:

$$\begin{bmatrix} \dot{P}_{x\{p\}} \\ \dot{P}_{y\{p\}} \\ \dot{P}_{z\{p\}} \end{bmatrix} = \begin{bmatrix} \dot{P}_{x\{a\}} \\ -\sin(\psi)\dot{P}_{y\{a\}} - \cos(\psi)\dot{\psi}(P_{y\{a\}} + d_3) \\ \cos(\psi)\dot{P}_{y\{a\}} - \sin(\psi)\dot{\psi}(P_{y\{a\}} + d_3) \end{bmatrix} \tag{44}$$

For the angular velocity, the components can be easily identified:

$$\begin{bmatrix} \dot{\theta}_{x\{p\}} \\ \dot{\theta}_{y\{p\}} \\ \dot{\theta}_{z\{p\}} \end{bmatrix} = \begin{bmatrix} \dot{\theta}_{x\{a\}} \\ \cos(\psi)\dot{\theta}_{z\{a\}} \\ -\sin(\psi)\dot{\theta}_{z\{a\}} \end{bmatrix} \tag{45}$$

where $\dot{\theta}_{x\{a\}}$ is the roll velocity and $\dot{\theta}_{z\{a\}}$ is the tilt velocity.

6 Kinematics Simulation and Visualization

The visualization script is available at `https://github.com/zhaob5/roll_tilt_mechanism`. This simulation focuses on the four-bar linkage mechanism, with all coordinates defined in frame {a}. The tool tip velocity and trajectory are also visualized, currently limited to 2-D planar space. Feel free to follow the page for updates or contribute if you're interested.

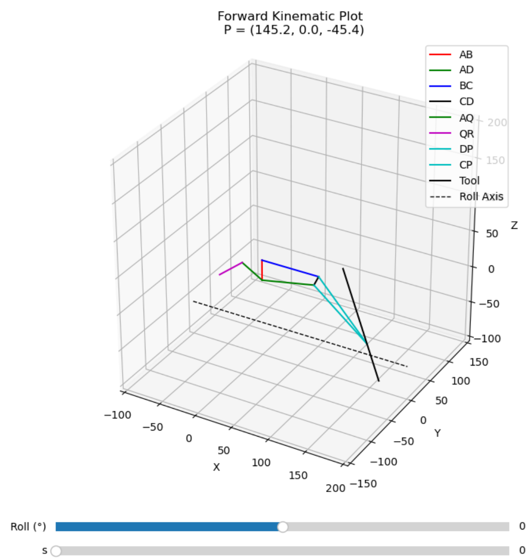

Figure 7: Forward Kinematics Visualization

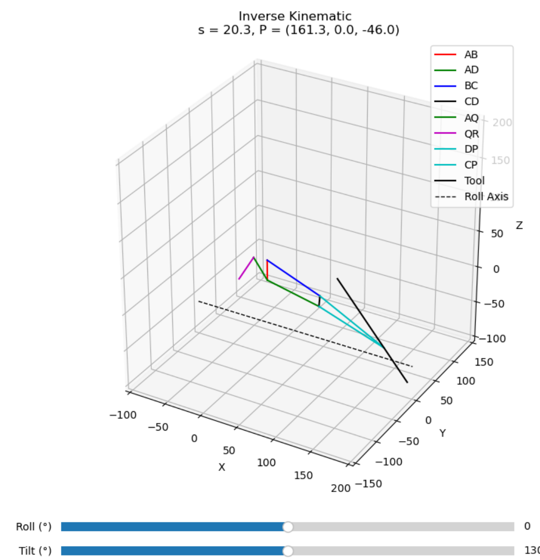

Figure 8: Inverse Kinematics Visualization

7 Disclaimer

The calculations and methods presented in this document are based on my personal understanding and interpretation of the roll and tilt mechanism of SHER-3.0. I hope the derivations provide useful insights and inspiration for your own work. Please note that if you define coordinate frames or the home configuration differently than described here, your results may differ. Therefore, it is more important to understand the principles of the derivation than to directly copy the final expressions into your work. While I've made every effort to ensure accuracy, there may still be errors or oversights. I strongly encourage you to verify the results independently before applying them in any critical context. If you have suggestions, corrections, or would like to discuss the topic further, feel free to reach out to me at: bzhao17@alumni.jh.edu.

8 Appendix: Python Scripts

8.1 Forward Kinematics Visualization

```python
# -*- coding: utf-8 -*-
"""
SHER-3.0 Tilt-Roll Forward Kinematics Visualization

Created on Thu Jun 19 13:26:00 2025

@author: Botao Zhao
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider

def ROTz(theta):
    return np.array([[np.cos(theta), -np.sin(theta), 0],
                     [np.sin(theta),  np.cos(theta), 0],
                     [0,              0,             1]])

def Pos(theta, L):
    return np.array([[L*np.cos(theta)],
                     [L*np.sin(theta)],
                     [0]])

def Trans(R, p):
    T = np.vstack((np.hstack((R, p)), [0, 0, 0, 1]))
    return T

def apply_roll_rotation(points, angle_deg, offset_z):
    angle_rad = np.deg2rad(angle_deg)
    R = np.array([[1, 0, 0],
                  [0, np.cos(angle_rad), -np.sin(angle_rad)],
                  [0, np.sin(angle_rad),  np.cos(angle_rad)]])
    rotated = []
    for p in points:
        p_shifted = p - np.array([0, 0, offset_z])
        p_rotated = R @ p_shifted + np.array([0, 0, offset_z])
        rotated.append(p_rotated)
    return rotated

# Fixed Parameters
x_AR_l = -60.48
y_AR = -10
L_AQ = 32.5
L_QR = 42
L_AB = 28
L_BC = 78.5
L_CD = 15
L_DA = 73.5
phi1 = np.pi / 2
phi2 = np.pi * 138 / 180
```

```
L_DP = 94.49
phi3 = np.pi * 103.93 / 180
phi4 = np.pi * 31.06 / 180
L_T = 50
d1 = -25.5 # Distance between the center frame of the delta platform to point A

# Create figure and axes
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
fig.subplots_adjust(bottom=0.15)

# Slider axes
slider_ax_s = fig.add_axes([0.2, 0.02, 0.6, 0.03])
slider_ax_roll = fig.add_axes([0.2, 0.06, 0.6, 0.03])

# Create sliders
s_slider = Slider(slider_ax_s, 's', 0, 50, valinit=0)
roll_slider = Slider(slider_ax_roll, 'Roll (°)', -90, 90, valinit=0)

def update_plot(_=None):
    s = s_slider.val
    roll = roll_slider.val
    ax.cla()

    x_AR = x_AR_l + s
    L_AR = np.sqrt(x_AR**2 + y_AR**2)
    alpha3 = np.arccos((L_AR**2 + L_AQ**2 - L_QR**2) / (2 * L_AR * L_AQ))
    alpha4 = np.pi - np.arctan2(y_AR, x_AR)

    theta1 = np.pi - (phi2 + alpha3 + alpha4)
    alpha1 = phi1 - theta1
    L_BD = np.sqrt(L_AB**2 + L_DA**2 - 2 * L_AB * L_DA * np.cos(alpha1))
    beta1 = np.arccos((L_AB**2 + L_BD**2 - L_DA**2) / (2 * L_AB * L_BD))

    delta = np.arccos((L_CD**2 + L_BD**2 - L_BC**2) / (2 * L_CD * L_BD))
    theta2 = phi1 + beta1 - delta

    T_AD = Trans(ROTz(theta1), Pos(theta1, L_DA))
    T_DP = Trans(ROTz(theta2 - phi3 - theta1), Pos(theta2 - phi3 - theta1, L_DP))
    T_PT = Trans(ROTz(np.pi / 2 - phi4 - (theta2 - phi3 - theta1)), Pos(np.pi / 2 - phi4 - (theta2 - phi
    T_AT = T_AD @ T_DP @ T_PT

    thetat = -np.arccos(T_AT[0][0])
    pt = np.array([T_AT[0][3], 0, T_AT[1][3]])
    p_tip = pt + np.array([np.cos(thetat)*L_T, 0, np.sin(thetat)*L_T])
    p_end = pt + np.array([-np.cos(thetat)*L_T*2, 0, -np.sin(thetat)*L_T*2])

    # Define original points
    A = np.array([0, 0, 0])
    B = A + np.array([0, 0, L_AB])
    D = A + np.array([np.cos(theta1) * L_DA, 0, np.sin(theta1) * L_DA])
    C = D + np.array([np.cos(theta2) * L_CD, 0, np.sin(theta2) * L_CD])
    Q = A + np.array([-np.cos(alpha3 + alpha4) * L_AQ, 0, np.sin(alpha3 + alpha4) * L_AQ])
    R = np.array([x_AR, 0, y_AR])
```

```python
    P = D + np.array([np.cos(phi3 - theta2) * L_DP, 0, -np.sin(phi3 - theta2) * L_DP])

    # Apply roll rotation to all points
    raw_points = [A, B, C, D, Q, R, P, p_tip, p_end]
    A, B, C, D, Q, R, P, p_tip, p_end = apply_roll_rotation(raw_points, roll, d1)

    # Plot links
    ax.plot([A[0], B[0]], [A[1], B[1]], [A[2], B[2]], 'r', label='AB')
    ax.plot([A[0], D[0]], [A[1], D[1]], [A[2], D[2]], 'g', label='AD')
    ax.plot([B[0], C[0]], [B[1], C[1]], [B[2], C[2]], 'b', label='BC')
    ax.plot([C[0], D[0]], [C[1], D[1]], [C[2], D[2]], 'k', label='CD')
    ax.plot([A[0], Q[0]], [A[1], Q[1]], [A[2], Q[2]], 'g', label='AQ')
    ax.plot([Q[0], R[0]], [Q[1], R[1]], [Q[2], R[2]], 'm', label='QR')
    ax.plot([D[0], P[0]], [D[1], P[1]], [D[2], P[2]], 'c', label='DP')
    ax.plot([C[0], P[0]], [C[1], P[1]], [C[2], P[2]], 'c', label='CP')
    ax.plot([p_tip[0], p_end[0]], [p_tip[1], p_end[1]], [p_tip[2], p_end[2]], 'k', label='Tool')

    # Plot the roll axis (dashed black line)
    x_range = np.array([-100, 200])
    y_fixed = np.array([0, 0])
    z_fixed = np.array([-60, -60])
    ax.plot(x_range, y_fixed, z_fixed, 'k--', linewidth=1, label='Roll Axis')

    # Axis settings
    ax.set_xlim([-100, 200])
    ax.set_ylim([-150, 150])
    ax.set_zlim([-100, 200])
    ax.set_box_aspect([1, 1, 1])
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.set_title(f"Forward Kinematic Plot \n P = ({P[0]:.1f}, {P[1]:.1f}, {P[2]:.1f})")
    ax.legend()

# Initial draw
update_plot()

# Connect sliders
s_slider.on_changed(update_plot)
roll_slider.on_changed(update_plot)

plt.show()
```

8.2 Inverse Kinematics Visualization

```python
# -*- coding: utf-8 -*-
"""
SHER-3.0 Tilt-Roll Inverse Kinematics Visualization

Created on Thu Jun 19 11:27:24 2025

@author: Botao Zhao
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider

# offset_z is the distance between the center frame of the delta platform to point A on the tilt mechan
def apply_roll_rotation(points, angle_deg, offset_z):
    angle_rad = np.deg2rad(angle_deg)
    R = np.array([[1, 0, 0],
                  [0, np.cos(angle_rad), -np.sin(angle_rad)],
                  [0, np.sin(angle_rad),  np.cos(angle_rad)]])
    rotated = []
    for p in points:
        p_shifted = p - np.array([0, 0, offset_z])
        p_rotated = R @ p_shifted + np.array([0, 0, offset_z])
        rotated.append(p_rotated)
    return rotated

# Constants
AB = 28
BC = 78.5
CD = 15
DA = 73.5
DP = 94.49
AQ = 32.5
QR = 42
x_AR_l = -60.48
y_AR = -10
L_T = 50
d1 = -25.5 # Distance between the center frame of the delta platform to point A

# Angles in radians
phi1 = np.pi / 2
phi2 = np.pi * 138 / 180
phi3 = np.pi * 103.93 / 180
phi4 = np.pi * 31.06 / 180

#print("theta2 = ", float(theta2*180/np.pi))

theat_tool_max = 150 # deg
theat_tool_min = 110 # deg
theat_tool_init = 130 # deg

# Create figure and axes
fig = plt.figure(figsize=(10, 8))
```

```python
ax = fig.add_subplot(111, projection='3d')
fig.subplots_adjust(bottom=0.15)

# Slider axes
slider_ax_theta_tool = fig.add_axes([0.2, 0.02, 0.6, 0.03])
slider_ax_roll = fig.add_axes([0.2, 0.06, 0.6, 0.03])

# Create sliders
theta_tool_slider = Slider(slider_ax_theta_tool, 'Tilt (°)', theat_tool_min, theat_tool_max, valinit=the
roll_slider = Slider(slider_ax_roll, 'Roll (°)', -90, 90, valinit=0)

def update_plot(_=None):
    theta_tool = theta_tool_slider.val * np.pi/180
    roll = roll_slider.val
    ax.cla()

    theta2 = phi3 - np.pi + phi4 + theta_tool
    a = CD*np.cos(theta2) - AB*np.cos(phi1)
    b = - CD*np.sin(theta2) + AB*np.sin(phi1)
    gamma = np.arctan2(a, b)
    angle_adc = np.arccos((DA**2+a**2+b**2-BC**2)/(2*DA*np.sqrt(a**2+b**2)))
    theta1 = np.pi/2 - angle_adc + gamma

    Px = np.cos(theta1)*DA + np.cos(phi3 - theta2)*DP
    Py = np.sin(theta1)*DA - np.sin(phi3 - theta2)*DP

    # Now find the slide bar position s:
    alpha = np.pi - theta1 - phi2 # alpha = alpha3 + alpha4

    x_AR = -np.cos(alpha)*AQ - np.sqrt(QR**2 - (np.sin(alpha)*AQ - y_AR)**2)

    s = x_AR - x_AR_l

    # Define points
    A = np.array([0, 0, 0])
    B = A + np.array([0, 0, AB])
    D = A + np.array([np.cos(theta1) * DA, 0, np.sin(theta1) * DA])
    C = D + np.array([np.cos(theta2) * CD, 0, np.sin(theta2) * CD])
    Q = A + np.array([-np.cos(alpha) * AQ, 0, np.sin(alpha) * AQ])
    R = np.array([x_AR, 0, y_AR])
    P = np.array([Px, 0, Py])

    # Tool shaft
    p_tip = P + np.array([np.cos(theta_tool)*L_T*2, 0, np.sin(theta_tool)*L_T*2])
    p_end = P + np.array([-np.cos(theta_tool)*L_T, 0, -np.sin(theta_tool)*L_T])

    # Apply roll rotation to all points
    raw_points = [A, B, C, D, Q, R, P, p_tip, p_end]
    A, B, C, D, Q, R, P, p_tip, p_end = apply_roll_rotation(raw_points, roll, d1)

    # Plot links
    ax.plot([A[0], B[0]], [A[1], B[1]], [A[2], B[2]], 'r', label='AB')
    ax.plot([A[0], D[0]], [A[1], D[1]], [A[2], D[2]], 'g', label='AD')
    ax.plot([B[0], C[0]], [B[1], C[1]], [B[2], C[2]], 'b', label='BC')
```

```python
    ax.plot([C[0], D[0]], [C[1], D[1]], [C[2], D[2]], 'k', label='CD')
    ax.plot([A[0], Q[0]], [A[1], Q[1]], [A[2], Q[2]], 'g', label='AQ')
    ax.plot([Q[0], R[0]], [Q[1], R[1]], [Q[2], R[2]], 'm', label='QR')
    ax.plot([D[0], P[0]], [D[1], P[1]], [D[2], P[2]], 'c', label='DP')
    ax.plot([C[0], P[0]], [C[1], P[1]], [C[2], P[2]], 'c', label='CP')
    ax.plot([p_tip[0], p_end[0]], [p_tip[1], p_end[1]], [p_tip[2], p_end[2]], 'k', label='Tool')

    # Plot the roll axis (dashed black line)
    x_range = np.array([-100, 200])
    y_fixed = np.array([0, 0])
    z_fixed = np.array([-60, -60])
    ax.plot(x_range, y_fixed, z_fixed, 'k--', linewidth=1, label='Roll Axis')

    # Axis settings
    ax.set_xlim([-100, 200])
    ax.set_ylim([-150, 150])
    ax.set_zlim([-100, 200])
    ax.set_box_aspect([1, 1, 1])
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.set_title(f"Inverse Kinematic \n s = {s:.1f}, P = ({P[0]:.1f}, {P[1]:.1f}, {P[2]:.1f})")
    ax.legend()

# Initial draw
update_plot()

# Connect sliders
theta_tool_slider.on_changed(update_plot)
roll_slider.on_changed(update_plot)

plt.show()
```