Closed-Form Jacobian Inverse Derivation for SHER-3.0

Botao Zhao

August 2, 2025

Contents

1 Introduction

In my previous work, *Kinematics and Jacobian Analysis of SHER-3.0*, I demonstrated that the robot's Jacobian pseudo-inverse can be expressed as $\mathbf{J}^\dagger = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$, which is a common numerical method for computing the inverse. However, while numerical approaches are convenient, having an analytical (closed-form) Jacobian inverse is essential for achieving fast, stable, and insightful control — especially in real-time robotics applications.

For example, a robotic arm running at 500Hz. A numerical Jacobian inverse might take 1.8ms per control cycle - uncomfortably close to the 2ms timing constraint - whereas a symbolic expression, optimized in code, usually could reduce that time to under 0.5ms. Beyond performance, analytical forms also expose singularities, reduce numerical drift, and support optimization routines that rely on clean derivatives. They're not just elegant - they're highly practical!

This document walks through the step-by-step derivation of the analytical pseudo-inverse of the Jacobian for SHER-3.0 (Eye Robot 3.0). If you haven't read my earlier work - especially *Kinematics and Jacobian Analysis of SHER-3.0* and related documents, **I strongly encourage you to review them first**, as this work builds directly on those foundational concepts. Those related documents are avalible at: `https://github.com/zhaob5/sher3-kinematics`

2 SHER-3.0 at a Glance - and Why It's Different

As mentioned throughout previous work, SHER-3.0 is a hybrid delta-serial robot designed specifically for eye surgery, as illustrated in Fig.(1). Its unique structure, a delta platform coupled with a rotating four-bar linkage, sets it apart from most conventional ophthalmic surgical robots and presents significant challenges to traditional kinematic modeling approaches. My previous work has focused on deriving its kinematics in detail, and I hope those efforts will provide useful insights, or practical guidance for your own work.

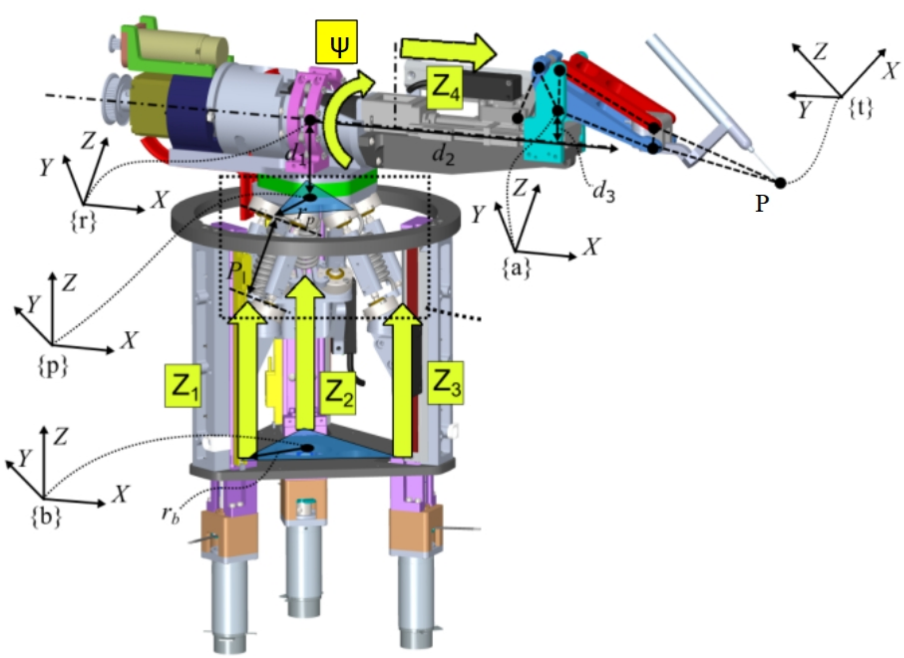

Figure 1: CAD models of the 5-DOF SHER-3.0 with assigned frames

3 Spatial Jacobian Inverse

Since we are not relying on numerical methods to compute the Jacobian inverse, it is important to first revisit its physical meaning. Suppose we want the tool tip to move with a desired spatial velocity $\mathbf{V}_{tool} = [\dot{x}, \dot{y}, \dot{z}, \dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z]^T$. To achieve this motion, the robot must drive its joints at specific velocities $\dot{\mathbf{q}} = [\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5]^T$. But how do we determine the required joint motions? This is where the Jacobian inverse comes in: it provides the mapping from task-space velocity to joint-space velocity. Our goal, therefore, is to derive this mapping from first principles, guided by physical intuition rather than black-box computation.

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{bmatrix}_{\{b\}} \tag{1}$$

For simplicity, all Jacobians discussed in this section refer to the spatial Jacobian, and velocities are expressed in base frame.

Recall from **Section-5.2** of *Kinematics of a 3-PUU Delta-Style Parallel Robot*, we derived the differential kinematics of the delta platform:

$$z_{l_i}\dot{q}_i = x_{l_i}\dot{x} + y_{l_i}\dot{y} + z_{l_i}\dot{z} \tag{2}$$

where $i = 1, 2, 3$, and $x_{l_i}, y_{l_i}, z_{l_i}$ are the x, y, z components of the vector along link $\mathbf{l}_i$. Here, $\dot{q}_i$ denotes the linear velocity of the i-th prismatic joint, and $\dot{x}, \dot{y}, \dot{z}$ represent the translational velocity of the delta platform in the base frame.

From this, we can directly relate the desired platform velocity to the corresponding joint velocities as:

$$\dot{q}_i = \frac{x_{l_i}}{z_{l_i}}\dot{x} + \frac{y_{l_i}}{z_{l_i}}\dot{y} + \dot{z} \tag{3}$$

Now, focusing just on the delta platform - if we know the desired linear velocity of the platform, the corresponding joint velocities can be directly computed from it as:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}_{delta} = \begin{bmatrix} \frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1 \\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1 \\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{4}$$

This relationship effectively defines the top-left block of the Jacobian inverse.

Now let's do some visualization: Although the roll and tilt joints appear to provide only angular motion, the tool tip is not located along their axes of rotation (as illustrated in Fig.2). In fact, due to the four-bar linkage, the tilt motion is mechanically coupled with translation. As a result, both roll and tilt inevitably produce linear motion in the x, y, z directions.

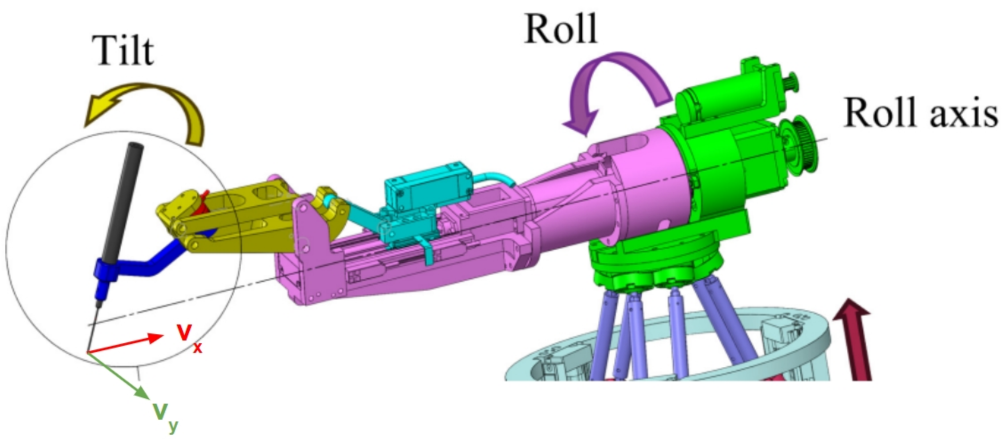

Figure 2: Roll-Tilt Mechanism and Induced Velocities v_x and v_y

Here comes the most challenging part to imagine: although angular velocities $\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z$ will cause nonzero $\dot{x}, \dot{y}, \dot{z}$, these induced translations can be computed precisely. So, to isolate the translation purely caused by the roll-tilt mechanism, we can simply compensate (by subtracting) these induced linear velocities with the delta platform.

How do we know the linear velocity of tool tip in frame $\{p\}$? If we just look the roll-tilt mechanism, as we derived previously in *Kinematics of the Roll-Tilt Mechanism in SHER-3.0*, the translational velocity induced by the roll-tilt mechanism can be expressed as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_{\{p\}} = \begin{bmatrix} \dot{P}_{x\{a\}} \\ -\sin(\psi)\dot{P}_{y\{a\}} - \cos(\psi)\dot{\psi}(P_{y\{a\}} + d_3) \\ \cos(\psi)\dot{P}_{y\{a\}} - \sin(\psi)\dot{\psi}(P_{y\{a\}} + d_3) \end{bmatrix} \tag{5}$$

where

$$\begin{aligned} \dot{P}_{x\{a\}} &= -\sin(\theta_1)L_{DA}\dot{\theta}_1 - \sin(\theta_2 - \phi_3)L_{DP}\dot{\theta}_2 \\ \dot{P}_{y\{a\}} &= \cos(\theta_1)L_{DA}\dot{\theta}_1 + \cos(\theta_2 - \phi_3)L_{DP}\dot{\theta}_2 \end{aligned} \tag{6}$$

and previously, we also derived the angular velocity of each link:

$$\begin{aligned} \dot{\theta}_1 &= -\frac{L_{ID}}{L_{DA}}\dot{\theta}_2 \\ \dot{\theta}_2 &= \frac{L_{DA}L_{JQ}}{L_{ID}L_{AQ}L_{JR}}\dot{s} \end{aligned} \tag{7}$$

Plug eq.(7) into eq.(6):

$$\begin{aligned} \dot{P}_{x\{a\}} &= \left(\sin(\theta_1)L_{ID} - \sin(\theta_2 - \phi_3)L_{DP}\right)\dot{\theta}_2 \\ \dot{P}_{y\{a\}} &= \left(-\cos(\theta_1)L_{ID} + \cos(\theta_2 - \phi_3)L_{DP}\right)\dot{\theta}_2 \end{aligned} \tag{8}$$

Let:

$$\begin{aligned} C &= \sin(\theta_1)L_{ID} - \sin(\theta_2 - \phi_3)L_{DP} \\ D &= -\cos(\theta_1)L_{ID} + \cos(\theta_2 - \phi_3)L_{DP} \end{aligned} \tag{9}$$

Then

$$\dot{P}_{x\{a\}} = C\dot{\theta}_2 \\ \dot{P}_{y\{a\}} = D\dot{\theta}_2 \tag{10}$$

By plugging eq.(5) into eq.(4), we obtain the joint velocities required to compensate for the induced linear motion:

$$\begin{aligned}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix} &= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}_{\{p\}} \\ &= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1\end{bmatrix}\begin{bmatrix}\dot{P}_{x\{a\}}\\ -\sin(\psi)\dot{P}_{y\{a\}} - \cos(\psi)\dot{\psi}(P_{y\{a\}} + d_3)\\ \cos(\psi)\dot{P}_{y\{a\}} - \sin(\psi)\dot{\psi}(P_{y\{a\}} + d_3)\end{bmatrix}\end{aligned} \tag{11}$$

Since we're interested in understanding how the roll and tilt mechanisms contribute to this induced velocity, we aim to express eq.(11) explicitly in terms of the roll and tilt rates $\dot{\psi}$ and $\dot{\theta}_2$. Substituting eq.(10) into eq.(11) yields:

$$\begin{aligned}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix} &= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1\end{bmatrix}\begin{bmatrix}C\dot{\theta}_2\\ -\sin(\psi)D\dot{\theta}_2 - \cos(\psi)(P_{y\{a\}} + d_3)\dot{\psi}\\ \cos(\psi)D\dot{\theta}_2 - \sin(\psi)(P_{y\{a\}} + d_3)\dot{\psi}\end{bmatrix} \\ &= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1\end{bmatrix}\begin{bmatrix}0 & C\\ -\cos(\psi)(P_{y\{a\}} + d_3) & -\sin(\psi)D\\ -\sin(\psi)(P_{y\{a\}} + d_3) & \cos(\psi)D\end{bmatrix}\begin{bmatrix}\dot{\psi}\\ \dot{\theta}_2\end{bmatrix} \\ &= \begin{bmatrix}-\frac{y_{l_1}}{z_{l_1}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_1}}{z_{l_1}}C - \frac{y_{l_1}}{z_{l_1}}\sin(\psi)D + \cos(\psi)D\\ -\frac{y_{l_2}}{z_{l_2}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_2}}{z_{l_2}}C - \frac{y_{l_2}}{z_{l_2}}\sin(\psi)D + \cos(\psi)D\\ -\frac{y_{l_3}}{z_{l_3}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_3}}{z_{l_3}}C - \frac{y_{l_3}}{z_{l_3}}\sin(\psi)D + \cos(\psi)D\end{bmatrix}\begin{bmatrix}\dot{\psi}\\ \dot{\theta}_2\end{bmatrix}\end{aligned} \tag{12}$$

Now we have established the relationship between the joint velocities required to compensate for the induced linear motion and the angular velocities of the roll and tilt mechanisms.

However, we want to express these angular velocities in the base frame as $[\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z]^T$. To do so, let's recall the expression for the tool tip's angular velocity previously derived in *Kinematics of the Roll-Tilt Mechanism in SHER-3.0*:

$$\begin{bmatrix}\dot{\theta}_x\\ \dot{\theta}_y\\ \dot{\theta}_z\end{bmatrix}_{\{p\}} = \begin{bmatrix}\dot{\psi}\\ -\cos(\psi)\dot{\theta}_2\\ -\sin(\psi)\dot{\theta}_2\end{bmatrix} \tag{13}$$

Since there's no rotational transformation between base frame $\{b\}$ and delta platform frame $\{p\}$, it's easy to figure out $[\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z]^T_{\{p\}} = [\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z]^T_{\{b\}}$.

From eq.(13), it's clear that although $\dot{\theta}_y$ and $\dot{\theta}_z$ appear to be independent, they are actually coupled through ψ and $\dot{\theta_2}$. In other words, specifying $\dot{\theta}_y$ determines a unique corresponding $\dot{\theta}_z$ and vice versa.

Since $\dot{\psi}$ represents the rotational velocity about the base frame's x axis, and we already have the relationship between $\dot{\theta}_y$ and $\dot{\theta}_2$ from eq.(13), we can substitute $\dot{\psi} = \dot{\theta}_x$ and $\dot{\theta}_2 = -\frac{1}{\cos(\psi)}\dot{\theta}_y$ into eq.(12) to obtain:

$$\begin{aligned}
\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} &= \begin{bmatrix} -\frac{y_{l_1}}{z_{l_1}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_1}}{z_{l_1}}C - \frac{y_{l_1}}{z_{l_1}}\sin(\psi)D + \cos(\psi)D \\ -\frac{y_{l_2}}{z_{l_2}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_2}}{z_{l_2}}C - \frac{y_{l_2}}{z_{l_2}}\sin(\psi)D + \cos(\psi)D \\ -\frac{y_{l_3}}{z_{l_3}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & \frac{x_{l_3}}{z_{l_3}}C - \frac{y_{l_3}}{z_{l_3}}\sin(\psi)D + \cos(\psi)D \end{bmatrix} \begin{bmatrix} \dot{\theta}_x \\ -\frac{1}{\cos(\psi)}\dot{\theta}_y \end{bmatrix} \\
&= \begin{bmatrix} -\frac{y_{l_1}}{z_{l_1}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_1}}{z_{l_1}\cos(\psi)}C + \frac{y_{l_1}}{z_{l_1}}\tan(\psi)D - D \\ -\frac{y_{l_2}}{z_{l_2}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_2}}{z_{l_2}\cos(\psi)}C + \frac{y_{l_2}}{z_{l_2}}\tan(\psi)D - D \\ -\frac{y_{l_3}}{z_{l_3}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_3}}{z_{l_3}\cos(\psi)}C + \frac{y_{l_3}}{z_{l_3}}\tan(\psi)D - D \end{bmatrix} \begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \end{bmatrix}
\end{aligned} \tag{14}$$

Let

$$\mathbf{M_r} = \begin{bmatrix} -\frac{y_{l_1}}{z_{l_1}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_1}}{z_{l_1}\cos(\psi)}C + \frac{y_{l_1}}{z_{l_1}}\tan(\psi)D - D \\ -\frac{y_{l_2}}{z_{l_2}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_2}}{z_{l_2}\cos(\psi)}C + \frac{y_{l_2}}{z_{l_2}}\tan(\psi)D - D \\ -\frac{y_{l_3}}{z_{l_3}}\cos(\psi)(P_{y\{a\}} + d_3) - \sin(\psi)(P_{y\{a\}} + d_3) & -\frac{x_{l_3}}{z_{l_3}\cos(\psi)}C + \frac{y_{l_3}}{z_{l_3}}\tan(\psi)D - D \end{bmatrix} \tag{15}$$

Then

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}_{induced} = \mathbf{M_r} \begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \end{bmatrix}_{\{b\}} \tag{16}$$

In this case, it's clear that $\dot{\theta}_z$ has no effect on $\dot{q}_1, \dot{q}_2, \dot{q}_3$. However, to maintain consistency with the full Jacobian inverse expression introduced later, we'll add $\dot{\theta}_z$ to the right-hand side of eq.(16) to make it consistancy with later Jacobian Inverse expression:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}_{induced} = [\mathbf{M_r}\ \mathbf{0}] \begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{bmatrix}_{\{b\}} \tag{17}$$

Now, let's combine eq.(4) and eq.(17). Note that the joint velocities from eq.(17) represent the linear velocity induced by roll and tilt, which is exactly the term that needs to be subtracted in our final Jacobian inverse formulation.

Thus:

$$\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix} = \begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix}_{delta} - \begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix}_{induced}$$

$$= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} - [\mathbf{M_r}\ \mathbf{0}]\begin{bmatrix}\dot{\theta}_x\\ \dot{\theta}_y\\ \dot{\theta}_z\end{bmatrix} \tag{18}$$

$$= \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1 & -\mathbf{M_r}_{[1,1]} & -\mathbf{M_r}_{[1,2]} & 0\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1 & -\mathbf{M_r}_{[2,1]} & -\mathbf{M_r}_{[2,2]} & 0\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1 & -\mathbf{M_r}_{[3,1]} & -\mathbf{M_r}_{[3,2]} & 0\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\theta}_x\\ \dot{\theta}_y\\ \dot{\theta}_z\end{bmatrix}_{\{b\}}$$

From eq.(7), since $\dot{s} = \dot{q}_5$ and $\dot{\theta}_2 = -\frac{1}{\cos(\psi)}\dot{\theta}_y$, we can get the relationship between $\dot{q}_5$ and $\dot{\theta}_y$:

$$\begin{aligned}\dot{\theta}_2 =& \frac{L_{DA}L_{JQ}}{L_{ID}L_{AQ}L_{JR}}\dot{s}\\ -\frac{1}{\cos(\psi)}\dot{\theta}_y =& \frac{L_{DA}L_{JQ}}{L_{ID}L_{AQ}L_{JR}}\dot{q}_5\end{aligned} \tag{19}$$

Recall ψ is just q_4. Rearrange eq.(19), and we will get:

$$\dot{q}_5 = -\frac{L_{ID}L_{AQ}L_{JR}}{\cos(q_4)L_{DA}L_{JQ}}\dot{\theta}_y \tag{20}$$

Since $\dot{q}_4 = \dot{\psi}$ represents the rotational velocity about the base frame's x axis,

$$\dot{q}_4 = \dot{\theta}_x \tag{21}$$

Thus, we now have the final two components needed to construct the Jacobian inverse. By stacking eq.(20) and eq.(21) to eq.(18), we obtain:

$$\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\\ \dot{q}_4\\ \dot{q}_5\end{bmatrix} = \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1 & -\mathbf{M_r}_{[1,1]} & -\mathbf{M_r}_{[1,2]} & 0\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1 & -\mathbf{M_r}_{[2,1]} & -\mathbf{M_r}_{[2,2]} & 0\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1 & -\mathbf{M_r}_{[3,1]} & -\mathbf{M_r}_{[3,2]} & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & -\frac{L_{ID}L_{AQ}L_{JR}}{\cos(q_4)L_{DA}L_{JQ}} & 0\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\theta}_x\\ \dot{\theta}_y\\ \dot{\theta}_z\end{bmatrix} \tag{22}$$

Finally, the analytical expression of Spatial Jacobian Inverse is:

$$\mathbf{J_s}^{-1} = \begin{bmatrix}\frac{x_{l_1}}{z_{l_1}} & \frac{y_{l_1}}{z_{l_1}} & 1 & -\mathbf{M_r}_{[1,1]} & -\mathbf{M_r}_{[1,2]} & 0\\ \frac{x_{l_2}}{z_{l_2}} & \frac{y_{l_2}}{z_{l_2}} & 1 & -\mathbf{M_r}_{[2,1]} & -\mathbf{M_r}_{[2,2]} & 0\\ \frac{x_{l_3}}{z_{l_3}} & \frac{y_{l_3}}{z_{l_3}} & 1 & -\mathbf{M_r}_{[3,1]} & -\mathbf{M_r}_{[3,2]} & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & -\frac{L_{ID}L_{AQ}L_{JR}}{\cos(q_4)L_{DA}L_{JQ}} & 0\end{bmatrix} \tag{23}$$

4 Body Jacobian Inverse

In *Kinematics and Jacobian Analysis of SHER-3.0*, we defined a simplified adjoint-like mapping $\mathbf{Ad}^{\dagger}_{(\mathbf{T_{bt}})}$ using only the rotation matrix:

$$\mathbf{Ad}^{\dagger}_{(\mathbf{T_{bt}})} = \begin{bmatrix} \mathbf{R_{bt}} & 0 \\ 0 & \mathbf{R_{bt}} \end{bmatrix} \tag{24}$$

where $\mathbf{T_{bt}}$ is the homogeneous transformation from base frame $\{b\}$ to tool frame $\{t\}$, and rotation matrix $\mathbf{R_{bt}} = \mathbf{T_{bt}}[:\mathbf{3}, :\mathbf{3}]$.

Recall the relationship between Spatial and Body Jacobians:

$$\mathbf{J_b} = {\mathbf{Ad}^{\dagger}_{(\mathbf{T_{bt}})}}^{-1} \mathbf{J_s} \tag{25}$$

Thus, we get the expression of the **Body Jacobian Inverse**:

$$\mathbf{J_b}^{-1} = \mathbf{J_s}^{-1} \mathbf{Ad}^{\dagger}_{(\mathbf{T_{bt}})} \tag{26}$$

5 Jacobian Inverse Validation

In *Kinematics and Jacobian Analysis of SHER-3.0*, we previously verified the correctness of the forward Jacobian. Therefore, we will use that validated Jacobian as the ground truth $\mathbf{J_{true}}(\mathbf{q})$ in this section to verify the analytical Jacobian Inverse. The Python script used for verification is provided in the appendix.

To perform the verification, we first assign arbitrary joint positions $\mathbf{q}$ and joint velocities $\dot{\mathbf{q}}_{\mathbf{d}}$. Using the verified Jacobian, we compute the corresponding tool tip velocities. These velocities are then fed into the Jacobian Inverse model. If the output joint velocities match the originally assigned (desired) values, the Jacobian Inverse is considered correct.

The verification workflow is as follows:

$$\begin{aligned} \mathbf{V}_{tool} &= \mathbf{J_{true}}(\mathbf{q})\dot{\mathbf{q}}_{\mathbf{d}} \\ \dot{\mathbf{q}} &= \mathbf{J}(\mathbf{q})^{-1}\mathbf{V}_{tool} \end{aligned} \tag{27}$$

If $\dot{\mathbf{q}} = \dot{\mathbf{q}}_{\mathbf{d}}$, then the Jacobian Inverse is correct.

Consider the following random joint positions and velocities

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} = \begin{bmatrix} 1\ mm \\ 2\ mm \\ 3\ mm \\ \frac{31\pi}{180}\ rad \\ 11\ mm \end{bmatrix}, \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \end{bmatrix} = \begin{bmatrix} 7\ mm/s \\ 8\ mm/s \\ 9\ mm/s \\ \frac{10\pi}{180}\ rad/s \\ 15\ mm/s \end{bmatrix}$$

The results are shown below:

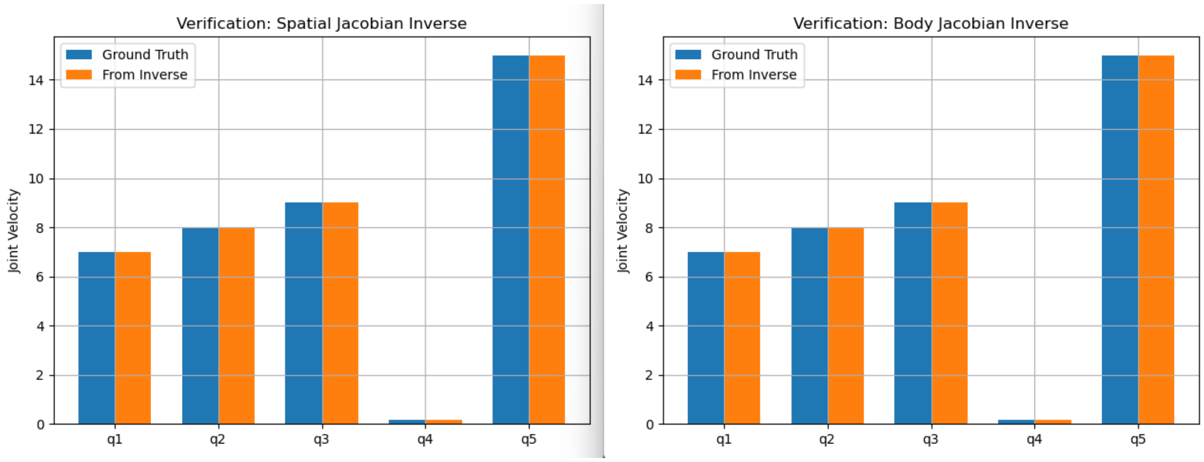

Figure 3: Joint Velocities Comparison

The joint velocities reconstructed using the analytical Jacobian inverse match the ground truth values across all joints. Thus, it confirms that the transformation from task-space velocities to joint-space velocities is accurate under the current kinematic model.

6 Disclaimer

The derivation process for the closed-form Jacobian inverse presented in this document is extremely tedious and prone to algebraic complexity. While every effort has been made to ensure the accuracy of the results, readers are strongly encouraged to focus on understanding the methodology and reasoning behind each step rather than relying solely on the final equations. If you plan to use these results in any critical applications, please independently verify all expressions and calculations for your specific use case. If you spot any errors, have suggestions, or would like to discuss the topic further, feel free to reach out to me at: bzhao17@alumni.jh.edu.

7 Appendix: Python Scripts

7.1 Jacobian Inverse Validation

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 30 13:17:00 2025

@author: Botao Zhao
"""

import numpy as np
from sympy import Matrix, zeros, pprint
import matplotlib.pyplot as plt

################## Adjoint-like Mapping ##########################
def adjoint_pseudo(T):
    R = T[:3, :3]
    return Matrix.vstack(
        Matrix.hstack(R, Matrix.zeros(3)),
        Matrix.hstack(Matrix.zeros(3), R)
    )

################## Joint Parameters: #############################
# Set desired Joint Positions Here:
q1 = 1  # Prismatic
q2 = 2  # Prismatic
q3 = 3  # Prismatic
q4 = 31*np.pi/180  # Revolute
q5 = 11  # Prismatic

################## Joint Velocities: #############################
# Set desired Joint Velocities Here:
q1_dot = 7  # mm/s
q2_dot = 8  # mm/s
q3_dot = 9  # mm/s
q4_dot = 10*np.pi/180  # rad/s
q5_dot = 15  # mm/s
q_dot = Matrix([
        [q1_dot],
        [q2_dot],
        [q3_dot],
        [q4_dot],
        [q5_dot]
])

############################ Delta Platform: ##############################
# Robot parameters
rp = 34.4773 # mm platform radius
rb = 60.6927 # mm base radius
l = 68  # mm rod length

# Base joint angles
theta1 = np.pi / 3
theta2 = np.pi
theta3 = 5 * np.pi / 3

# Base joint positions (fixed)
rb1 = np.array([np.cos(theta1)*rb, np.sin(theta1)*rb, 0])
```

```python
rb2 = np.array([np.cos(theta2)*rb, np.sin(theta2)*rb, 0])
rb3 = np.array([np.cos(theta3)*rb, np.sin(theta3)*rb, 0])

# Platform offsets (relative to center)
rp1 = np.array([np.cos(theta1)*rp, np.sin(theta1)*rp, 0])
rp2 = np.array([np.cos(theta2)*rp, np.sin(theta2)*rp, 0])
rp3 = np.array([np.cos(theta3)*rp, np.sin(theta3)*rp, 0])


L1 = np.array([0,0,q1])
L2 = np.array([0,0,q2])
L3 = np.array([0,0,q3])

e1 = L1 + rb1 - rp1
e2 = L2 + rb2 - rp2
e3 = L3 + rb3 - rp3

h1 = ((e2[0]**2 + e2[1]**2 + e2[2]**2) - (e1[0]**2 + e1[1]**2 + e1[2]**2))/2
h2 = ((e3[0]**2 + e3[1]**2 + e3[2]**2) - (e1[0]**2 + e1[1]**2 + e1[2]**2))/2

x21 = (e2 - e1)[0]
y21 = (e2 - e1)[1]
z21 = (e2 - e1)[2]

x31 = (e3 - e1)[0]
y31 = (e3 - e1)[1]
z31 = (e3 - e1)[2]

k1 = (y31*z21/y21 - z31)/(x31 - y31*x21/y21)
k2 = (h2 - y31*h1/y21)/(x31 - y31*x21/y21)
k3 = (x31*z21/x21 - z31)/(y31 - x31*y21/x21)
k4 = (h2 - x31*h1/x21)/(y31 - x31*y21/x21)

T1 = k1**2 + k3**2 + 1
T2 = k1*k2 + k3*k4 - e1[0]*k1 - e1[1]*k3 - e1[2]
T3 = k2**2 + k4**2 -2*e1[0]*k2 - 2*e1[1]*k4 - l**2 + e1[0]**2 + e1[1]**2 + e1[2]**2

z = (-T2 + np.sqrt(T2**2 - T1*T3))/T1
x = k1*z + k2
y = k3*z + k4

r = np.array([x, y, z])

l1 = -L1 - rb1 + r + rp1
l2 = -L2 - rb2 + r + rp2
l3 = -L3 - rb3 + r + rp3


######################### Roll-Tilt Mechanism: ##########################

s = q5 # slider linear actuator
s_dot = q5_dot # slider velocity
psi = q4 # roll angle

# Fixed Parameters
x_AR_max = 60.48
y_AR = -10
L_AQ = 32.5
L_QR = 42
```

```python
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 = 56.01 # Distance between the center frame of the delta platform to point A
d2 = 192.7
d3 = 25.5

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

# Position Analysis
x_AR = x_AR_max - 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)
alpha = alpha3 + alpha4

theta1 = np.pi - (phi2 + alpha3 + alpha4)
alpha_1 = phi1 - theta1
L_BD = np.sqrt(L_AB**2 + L_DA**2 - 2 * L_AB * L_DA * np.cos(alpha_1))
beta_1 = 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 + beta_1 - delta

# Velocity Analysis
L_JR = np.tan(alpha)*x_AR - y_AR
L_JQ = np.sqrt((L_JR+y_AR)**2 + x_AR**2) - L_AQ

rho_dot = s_dot / L_JR
theta1_dot = -L_JQ / L_AQ * rho_dot

beta_2 = np.arccos((L_BC**2 + L_BD**2 - L_CD**2) / (2 * L_BC * L_BD))
L_IA = L_AB * np.sin(beta_1 + beta_2) / np.sin(alpha_1 + beta_1 + beta_2)
L_ID = L_IA - L_DA

theta2_dot = -L_DA / L_ID * theta1_dot

# Position in {a} frame:
px_a = np.cos(theta1)*L_DA + np.cos(theta2 - phi3)*L_DP
py_a = np.sin(theta1)*L_DA + np.sin(theta2 - phi3)*L_DP

# Position in {p} frame:
px_p = px_a + d2
py_p = -np.sin(psi)*py_a - np.sin(psi)*d3
pz_p = np.cos(psi)*py_a + np.cos(psi)*d3 + d1

# Position in {b} frame:
px_b = px_a + d2 + x
```

```python
py_b = -np.sin(psi)*py_a - np.sin(psi)*d3 + y
pz_b = np.cos(psi)*py_a + np.cos(psi)*d3 + d1 + z

# print([px_p, py_p, pz_p])



########## Homogeneous Transformation from Base frame to Tool frame ###########
# Tbt
theta = np.pi/2 - phi4 - phi3 + theta2
Tbt = Matrix([
    [np.cos(theta), -np.sin(theta), 0, px_a+d2+x],
    [-np.sin(theta)*np.sin(psi), -np.cos(theta)*np.sin(psi), -np.cos(psi), -np.sin(psi)*py_a-np.sin(psi)*d3+y],
    [np.sin(theta)*np.cos(psi), np.cos(theta)*np.cos(psi), -np.sin(psi), np.cos(psi)*py_a+np.cos(psi)*d3+d1+z],
    [0, 0, 0, 1]
    ])  # 4x4 transformation from base to tool
Rbt = Tbt[:3, :3]

# Compute Ad_Tbt^1
Ttb = Tbt.inv()
Ad_Ttb = adjoint_pseudo(Ttb)


################## Spatial Jacobian Inverse (Analytical) #################

JacobianInvTip = zeros(5,6)

C = np.sin(theta1)*L_ID - np.sin(theta2-phi3)*L_DP
D = -np.cos(theta1)*L_ID + np.cos(theta2-phi3)*L_DP

M_r = zeros(3,2)
M_r[0,0] = -l1[1]/l1[2] * np.cos(psi) * (py_a + d3) - np.sin(psi)*(py_a + d3)
M_r[0,1] = -l1[0]/(l1[2]*np.cos(psi)) * C + l1[1]/l1[2]*np.tan(psi)*D - D

M_r[1,0] = -l2[1]/l2[2] * np.cos(psi) * (py_a + d3) - np.sin(psi)*(py_a + d3)
M_r[1,1] = -l2[0]/(l2[2]*np.cos(psi)) * C + l2[1]/l2[2]*np.tan(psi)*D - D

M_r[2,0] = -l3[1]/l3[2] * np.cos(psi) * (py_a + d3) - np.sin(psi)*(py_a + d3)
M_r[2,1] = -l3[0]/(l3[2]*np.cos(psi)) * C + l3[1]/l3[2]*np.tan(psi)*D - D

JacobianInvTip[0,0] = l1[0]/l1[2]
JacobianInvTip[1,0] = l2[0]/l2[2]
JacobianInvTip[2,0] = l3[0]/l3[2]

JacobianInvTip[0,1] = l1[1]/l1[2]
JacobianInvTip[1,1] = l2[1]/l2[2]
JacobianInvTip[2,1] = l3[1]/l3[2]

JacobianInvTip[0,2] = 1
JacobianInvTip[1,2] = 1
JacobianInvTip[2,2] = 1

JacobianInvTip[0:3, 3:5] = -M_r

JacobianInvTip[3,3] = 1

JacobianInvTip[4,4] = -L_ID*L_AQ*L_JR/(np.cos(q4)*L_DA*L_JQ)

JacobianInvTip[4,5] = 0 #
```

```python

Js_inv = JacobianInvTip # 5x6 Spatial Jacobian Inverse



################ Body Jacobian Inverse  (Analytical) ################
Ad_Tbt = adjoint_pseudo(Ttb).T

# Compute body Jacobian
Jb_inv = Js_inv * Ad_Tbt # 5x6 Body Jacobian Inverse


################ Spatial Jacobian ##############
J_l = zeros(3,3)

J_l[0,0] = l1[0]
J_l[0,1] = l1[1]
J_l[0,2] = l1[2]
J_l[1,0] = l2[0]
J_l[1,1] = l2[1]
J_l[1,2] = l2[2]
J_l[2,0] = l3[0]
J_l[2,1] = l3[1]
J_l[2,2] = l3[2]

J_z = zeros(3,3)

J_z[0,0] = l1[2]
J_z[1,1] = l2[2]
J_z[2,2] = l3[2]

J_d = J_l.inv() * J_z

A = np.sin(theta1)*L_DA*L_JQ/(L_AQ*L_JR) - np.sin(theta2-phi3)*L_DP*L_DA*L_JQ/(L_ID*L_AQ*L_JR)
B = -np.cos(theta1)*L_DA*L_JQ/(L_AQ*L_JR) + np.cos(theta2-phi3)*L_DP*L_DA*L_JQ/(L_ID*L_AQ*L_JR)

JacobianTip = zeros(6,5)

JacobianTip[:3, :3] = J_d

JacobianTip[1,3] = -np.cos(q4)*(py_a + d3)
JacobianTip[2,3] = -np.sin(q4)*(py_a + d3)
JacobianTip[3,3] = 1

JacobianTip[0,4] = A
JacobianTip[1,4] = -np.sin(q4)*B
JacobianTip[2,4] = np.cos(q4)*B
JacobianTip[4,4] = -np.cos(q4)*L_DA*L_JQ/(L_ID*L_AQ*L_JR)
JacobianTip[5,4] = -np.sin(q4)*L_DA*L_JQ/(L_ID*L_AQ*L_JR)

Js = JacobianTip        # 6x5 spatial Jacobian



################ Body Jacobian ##################

# Tbt is known

# Compute body Jacobian
```

```
Jb = Ad_Ttb * Js


################ Tool Tip Velocity #################
V_tool_t = Jb*q_dot
V_tool_b = Js*q_dot

# print("\nIn base frame:\n")
# print("x_dot_b = ", V_tool_b[0], "mm/s")
# print("y_dot_b = ", V_tool_b[1], "mm/s")
# print("z_dot_b = ", V_tool_b[2], "mm/s")
# print("theta_x_dot_b = ", V_tool_b[3]*180/np.pi, "deg/s")
# print("theta_y_dot_b = ", V_tool_b[4]*180/np.pi, "deg/s")
# print("theta_z_dot_b = ", V_tool_b[5]*180/np.pi, "deg/s\n")


# print("\nIn tool frame:\n")
# print("x_dot_t = ", V_tool_t[0], "mm/s")
# print("y_dot_t = ", V_tool_t[1], "mm/s")
# print("z_dot_t = ", V_tool_t[2], "mm/s")
# print("theta_x_dot_t = ", V_tool_t[3]*180/np.pi, "deg/s")
# print("theta_y_dot_t = ", V_tool_t[4]*180/np.pi, "deg/s")
# print("theta_z_dot_t = ", V_tool_t[5]*180/np.pi, "deg/s\n")



################ Jacobian Inverse Verification #################

q_dot_verify_s = Js_inv * V_tool_b
q_dot_verify_b = Jb_inv * V_tool_t

# print("Joint Velocity from Jacobian Pseudo-Inverse:")
# print("q_dot_s =")
# pprint(q_dot_verify_s) # The results should match your q_dot values
# print("q_dot_b =")
# pprint(q_dot_verify_b) # The results should match your q_dot values

################## Visualization #############################

def visualize_joint_velocity_comparison(original, computed, title):
    original_np = np.array(original).astype(np.float64).flatten()
    computed_np = np.array(computed).astype(np.float64).flatten()

    joint_labels = [f'q{i+1}' for i in range(len(original_np))]
    x = np.arange(len(joint_labels))
    width = 0.35

    fig, ax = plt.subplots()
    ax.bar(x - width/2, original_np, width, label='Ground Truth')
    ax.bar(x + width/2, computed_np, width, label='From Inverse')

    ax.set_ylabel('Joint Velocity')
    ax.set_title(title)
    ax.set_xticks(x)
    ax.set_xticklabels(joint_labels)
    ax.legend()
    ax.grid(True)

    plt.tight_layout()
```

```python
    plt.show()

# Visualize results
visualize_joint_velocity_comparison(q_dot, q_dot_verify_s, "Verification: Spatial Jacobian Inverse")
visualize_joint_velocity_comparison(q_dot, q_dot_verify_b, "Verification: Body Jacobian Inverse")
```