Velocity Kinematics

Prof. Gerardo Flores

Oct 16, 2024

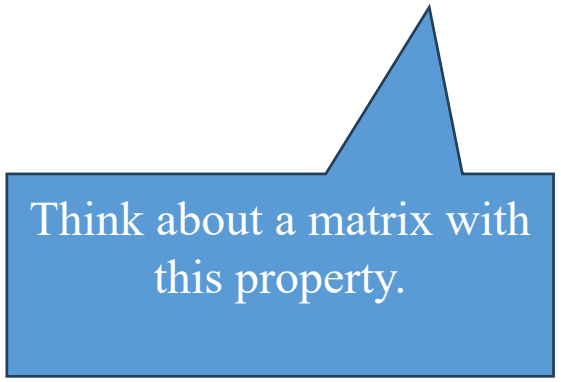
Robotics and Automation course,

TAMIU

Skew symmetric matrices

Definition 4.1 *An $n \times n$ matrix S is said to be* **skew symmetric** *if and only if*

$$S^T + S = 0 \tag{4.3}$$



Definition 4.1 *An $n \times n$ matrix S is said to be* **skew symmetric** *if and only if*

$$S^T + S = 0 \tag{4.3}$$

We denote the set of all 3×3 skew symmetric matrices by $so(3)$. If $S \in so(3)$ has components s_{ij}, $i, j = 1, 2, 3$ then Equation (4.3) is equivalent to the nine equations

$$s_{ij} + s_{ji} = 0 \qquad i, j = 1, 2, 3 \tag{4.4}$$

From Equation (4.4) we see that $s_{ii} = 0$; that is, the diagonal terms of S are zero and the off diagonal terms s_{ij}, $i \neq j$ satisfy $s_{ij} = -s_{ji}$. Thus S contains only three independent entries and every 3×3 skew symmetric matrix has the form

$$S = \begin{bmatrix} 0 & -s_3 & s_2 \\ s_3 & 0 & -s_1 \\ -s_2 & s_1 & 0 \end{bmatrix} \tag{4.5}$$

If $a = (a_x, a_y, a_z)^T$ is a 3-vector, we define the skew symmetric matrix $S(a)$ as

$$S(a) = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

Example

We denote by i, j and k the three unit basis coordinate vectors,

$$i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; \quad j = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}; \quad k = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The skew symmetric matrices $S(i)$, $S(j)$, and $S(k)$ are given by

$$S(i) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad S(j) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$S(k) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Code example: Generating a Skew-Symmetric Matrix from a 3D Vector

Write a code in Python that takes a vector $v = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$, where a,b,c are real numbers and returns the corresponding **Skew-symmetric matrix**. A skew-symmetric matrix A satisfies $A = -A^T$. The matrix for vector v will have the structure

$$A = \begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix}.$$

This function can be particularly useful in robotics and linear algebra for calculating cross products and rotations in 3D space.

```python
import numpy as np

def skew_symmetric(v):
    """
    Generates a skew-symmetric matrix from a vector v = [v1, v2, v3].
    """
    if len(v) != 3:
        raise ValueError("The vector must have exactly 3 elements.")

    # Construct the skew-symmetric matrix
    return np.array([
                        [0, -v[2], v[1]],
                        [v[2], 0, -v[0]],
                        [-v[1], v[0], 0]
    ])

# Example usage
vector = np.array([1, 2, 3])
skew_matrix = skew_symmetric(vector)

# Print the resulting skew-symmetric matrix
print("Skew-symmetric matrix:")
print(skew_matrix)
```

Properties of Skew Symmetric Matrices

1. The operator S is linear, i.e.,

$$S(\alpha a + \beta b) \quad = \quad \alpha S(a) + \beta S(b) \tag{4.6}$$

for any vectors a and b belonging to $\mathbb{R}^3$ and scalars α and β.

2. For any vectors a and p belonging to $\mathbb{R}^3$,

$$S(a)p \quad = \quad a \times p \tag{4.7}$$

where $a \times p$ denotes the vector cross product. Equation (4.7) can be verified by direct calculation.

3. If $R \in SO(3)$ and a, b are vectors in $\mathbb{R}^3$ it can also be shown by direct calculation that

$$R(a \times b) \quad = \quad Ra \times Rb \tag{4.8}$$

$$S(\mathbf{a}) = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

$$\mathbf{a} \times \mathbf{p} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ p_1 & p_2 & p_3 \end{vmatrix}$$

$$\mathbf{a} \times \mathbf{p} = \begin{bmatrix} a_2p_3 - a_3p_2 \\ a_3p_1 - a_1p_3 \\ a_1p_2 - a_2p_1 \end{bmatrix}$$

Equation (4.8) is not true in general unless R is orthogonal.

Properties of Skew Symmetric Matrices

4. For $R \in SO(3)$ and $a \in \mathbb{R}^3$

$$RS(a)R^T = S(Ra) \tag{4.9}$$

Property

This property follows easily from Equations (4.7) and (4.8) as follows. Let $b \in \mathbb{R}^3$ be an arbitrary vector. Then

$$\begin{aligned} RS(a)R^T b &= R(a \times R^T b) \\ &= (Ra) \times (RR^T b) \\ &= (Ra) \times b \\ &= S(Ra)b \end{aligned}$$

$S(c)p = c \times p$

$R(c \times p) = Rc \times Rp$

$S(c)p = c \times p$

Similarity transformation

Proof

and the result follows.

Exercise

Corroborate that the properties 1 and 2 hold for the example of:

$a = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $\mathrm{b} = \begin{pmatrix} 0 \\ 4 \\ 6 \end{pmatrix}$, $\mathrm{p} = \begin{pmatrix} 1 \\ -5 \\ 2 \end{pmatrix}$, $\alpha = 2$, and $\beta = 1$.

The derivative of R: particular case

Suppose now that a rotation matrix R is a function of the single variable θ. Hence $R = R(\theta) \in SO(3)$ for every θ. Since R is orthogonal for all θ it follows that

$$R(\theta)R(\theta)^T = I \tag{4.10}$$

→ Identity matrix

Differentiating both sides of Equation (4.10) with respect to θ using the product rule gives

$$\frac{dR}{d\theta}R(\theta)^T + R(\theta)\frac{dR^T}{d\theta} = 0 \tag{4.11}$$

→ Chain rule

Let us define the matrix S as

$$S := \frac{dR}{d\theta}R(\theta)^T \tag{4.12}$$

→ Definition of S

Then the transpose of S is

$$S^T = \left(\frac{dR}{d\theta}R(\theta)^T\right)^T = R(\theta)\frac{dR^T}{d\theta} \tag{4.13}$$

→ Transpose of S

Equation (4.11) says therefore that

$$S + S^T = 0 \tag{4.14}$$

→ Skew-symmetric matrix

The derivative of R

In other words, the matrix S defined by Equation (4.12) is skew symmetric. Multiplying both sides of Equation (4.12) on the right by R and using the fact that $R^T R = I$ yields

$$\frac{dR}{d\theta} = SR(\theta) \tag{4.15}$$

Equation 4.15 is very important in robotics, it says that computing the derivative of a rotation matrix R is equivalent to a matrix multiplication by a skew symmetric matrix S.

Example

The derivative of R → $\frac{dR}{d\theta} = SR(\theta)$

Consider the following rotation matrix, $R_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$, using the derivative of R it follows that,

$$S = \frac{dR}{d\theta}R^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\sin\theta & -\cos\theta \\ 0 & \cos\theta & -\sin\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = S(i)$$

$$\frac{d}{d\theta}\sin(\theta) = \cos(\theta)$$

$$\frac{d}{d\theta}\cos(\theta) = -\sin(\theta)$$

Thus we have shown that

$$\frac{dR_{x,\theta}}{d\theta} = S(i)R_{x,\theta}$$

Exercise

Follow the same procedure to prove that:

$$\frac{dR_{y,\theta}}{d\theta} = S(j)R_{y,\theta} \qquad and \qquad \frac{dR_{z,\theta}}{d\theta} = S(k)R_{z,\theta}$$

where

$$R_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Angular velocity

We consider the general case of angular velocity about an arbitrary, possibly moving, axis. Suppose,

$$R = R(t) \in SO(3), \forall t \in \mathbb{R}$$

and assume that $R(t)$ is continuously differentiable as a function of t, then an using the argument as in the previous section, we get

$$\dot{R}(t) = S(t)R(t).$$

Where $S(t)$ is skew-symmetric, and thus, it can be represented by $S(\omega)$ for a unique vector $\omega(t)$.

$\omega(t)$ is the angular velocity of the rotating frame with respect to the fixed frame at time t.

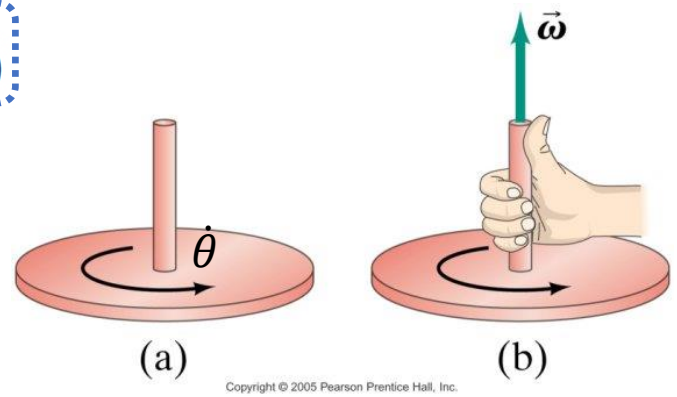
Therefore we have,

$$\dot{R}(t) = S\big(\omega(t)\big)R(t),$$

In which $\omega(t)$ is the angular velocity.

$$S(\alpha a + \beta b) \quad = \quad \alpha S(a) + \beta S(b)$$

Angular velocity: Example



Suppose that $R(t) = R_{x,\theta(t)}$. *Then* $\dot{R}(t)$ *is computed using the chain rule as*

$$\dot{R} = \frac{dR}{dt} = \frac{dR}{d\theta}\frac{d\theta}{dt} = \dot{\theta} S(i) R(t) = S(\omega(t)) R(t) \qquad (4.20)$$

in which $\omega = i\dot{\theta}$ *is the* **angular velocity**. *Note, here* $i = (1, 0, 0)^T$.

◇

$$S(i) \quad = \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

Recall from a previous example that, $\dfrac{dR_{x,\theta}}{d\theta} \quad = \quad S(i) R_{x,\theta}$

- The **angular velocity vector** ω is defined as:

$$\omega = i\,\dot{\theta} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \dot{\theta} = \begin{bmatrix} \dot{\theta} \\ 0 \\ 0 \end{bmatrix}$$

- Now, the **skew-symmetric matrix** associated with ω is:

$$S(\omega) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\dot{\theta} \\ 0 & \dot{\theta} & 0 \end{bmatrix}$$

This matrix $S(\omega)$ is simply $S(i)$ **scaled by** $\dot{\theta}$. Therefore:

$$\dot{\theta}\, S(i) = S(\omega)$$

Addition of angular velocities

$\omega_{0,1}^{0} + \omega_{0,1}^{1} = ????$

- Goal: find the expressions for the composition of angular velocities of two moving frames $o_1x_1y_1z_1$ and $o_2x_2y_2z_2$ relative to the fixed frame $o_0x_0y_0z_0$.

Finding the resultant angular velocity due to the relative rotation of several coordinate frames

Let the relative orientations of the frames $o_1x_1y_1z_1$ and $o_2x_2y_2z_2$ be given by $R_1^0(t)$ and $R_2^1(t)$, and then:

$$R_2^0(t) = R_1^0(t)\, R_2^1(t), \quad \rightarrow \text{Composition of rotations}$$

$$\dot{R}_2^0(t) = \dot{R}_1^0\, R_2^1 + R_1^0\, \dot{R}_2^1, \quad \rightarrow \text{Derivative}$$

Remember that $\dot{R}(t) = S\big(\omega(t)\big)R(t)$, then using this expression for the derivative of $R_2^0(t)$, it results in

$$\dot{R}_2^0(t) = S(\omega_{0,2}^0)R_2^0$$

$\omega_{0,2}^0$ is the angular velocity experienced by frame $o_2x_2y_2z_2$ that results from changing $R_2^0 = R_1^0 R_2^1$ [combined rotations]. Such angular velocity is expressed relative to $o_0x_0y_0z_0$.

Recall our previous equation:

$$\dot{R}_2^0(t) = \dot{R}_1^0 R_2^1 + R_1^0 \dot{R}_2^1$$

$\dot{R}(t) = S\big(\omega(t)\big)R(t)$ ← $\dot{R}_1^0 R_2^1 = S(\omega_{0,1}^0)R_1^0 R_2^1$

$$R_1^0 \dot{R}_2^1 = R_1^0 S(\omega_{1,2}^1) R_2^1$$
$$= R_1^0 S(\omega_{1,2}^1)[(R_1^0)^T R_1^0] R_2^1$$
$$= S(R_1^0 \omega_{1,2}^1)\, R_2^0$$

Similarity transformation

Recall our previous equation:

$$\dot{R}_2^0(t) = \dot{R}_1^0\, R_2^1 + R_1^0\, \dot{R}_2^1$$

$\dot{R}_1^0\, R_2^1 = S(\omega_{0,1}^0) R_1^0 R_2^1$

$\dot{R}(t) = S(\omega(t))R(t)$

$R_1^0 \dot{R}_2^1 = R_1^0 S(\omega_{1,2}^1) R_2^1$

$= R_1^0 S(\omega_{1,2}^1)[(R_1^0)^T R_1^0] R_2^1$

$= S(R_1^0 \omega_{1,2}^1)\, R_2^0$

Similarity transformation

Then,

$$\dot{R}_2^0(t) = S(\omega_{0,1}^0) R_2^0 + S(R_1^0 \omega_{1,2}^1)\, R_2^0$$

$$S(\omega_{0,2}^0)\, R_2^0 = [S(\omega_{0,1}^0) + S(R_1^0 \omega_{1,2}^1)]\, R_2^0$$

Since $S(a) + S(b) = S(a + b)$, then

$$\omega_{0,2}^0 = \omega_{0,1}^0 + R_1^0 \omega_{1,2}^1$$

The angular velocities can be added once they are expressed relative to the same coordinate frame, in this case $o_0 x_0 y_0 z_0$.

The above reasoning can be extended to any number of coordinate systems. In particular, suppose that we are given

$$R_n^0 = R_1^0\ R_2^1 \cdots R_n^{n-1}$$

and then,

$$\omega_{0,n}^0 = \omega_{0,1}^0 + \omega_{1,2}^0 + \omega_{2,3}^0 + \omega_{3,4}^0 + \cdots + \omega_{n-1,n}^0$$

The angular velocities can be added once they are expressed relative to the same coordinate frame, in this case $o_0x_0y_0z_0$.

Linear velocity of a point attached to a moving frame

We now consider the linear velocity of a point that is rigidly attached to a moving frame. Suppose the point p is rigidly attached to the frame $o_1x_1y_1z_1$, and that $o_1x_1y_1z_1$ is rotating relative to the frame $o_0x_0y_0z_0$. Then the coordinates of p with respect to the frame $o_0x_0y_0z_0$ are given by

$$p^0 = R_1^0(t)p^1. \tag{4.33}$$

The velocity $\dot{p}^0$ is then given by the product rule for differentiation as

$$\dot{p}^0 = \dot{R}_1^0(t)p^1 + R_1^0(t)\dot{p}^1 \tag{4.34}$$

$$= S(\omega^0)R_1^0(t)p^1 \tag{4.35}$$

$$= S(\omega^0)p^0 = \omega^0 \times p^0 \tag{4.36}$$

p is rigidly attached to frame one, and then its coordinates relative to frame one do not change, giving $\dot{p}^1 = 0$.

Linear velocity of a point attached to a moving frame

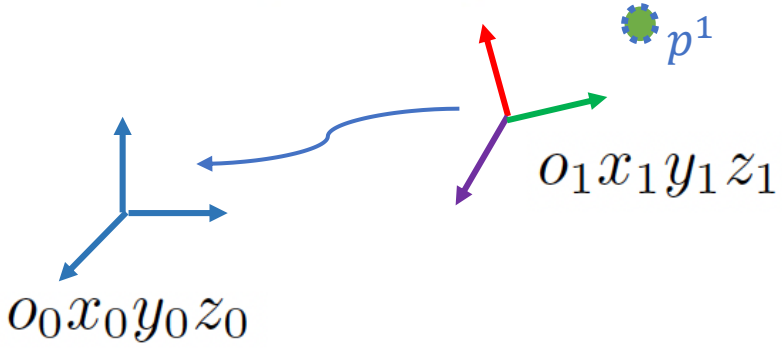
Now suppose that the motion of the frame $o_1x_1y_1z_1$ relative to $o_0x_0y_0z_0$ is more general. Suppose that the homogeneous transformation relating the two frames is time-dependent, so that

$$H_1^0(t) = \begin{bmatrix} R_1^0(t) & o_1^0(t) \\ 0 & 1 \end{bmatrix} \tag{4.37}$$

For simplicity we omit the argument t and the subscripts and superscripts on R_1^0 and o_1^0, and write

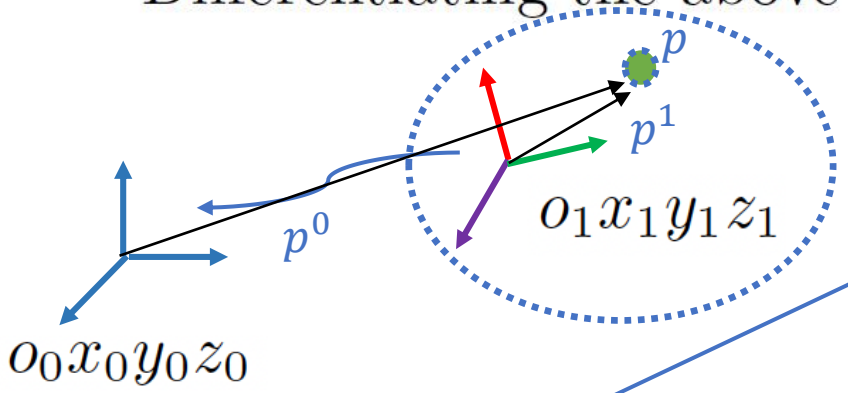
$$p^0 = Rp^1 + o \tag{4.38}$$

We compute $H_1^0 \begin{pmatrix} p^1 \\ 1 \end{pmatrix}$



$$p^0 = Rp^1 + o$$

Differentiating the above expression using the product rule gives



$$\begin{aligned} \dot{p}^0 &= \dot{R}p^1 + \dot{o} \\ &= S(\omega)Rp^1 + \dot{o} \\ &= \omega \times r + v \end{aligned} \tag{4.39}$$

where $r = Rp^1$ is the vector from o_1 to p expressed in the orientation of the frame $o_0x_0y_0z_0$, and v is the rate at which the origin o_1 is moving.

If the point p is moving relative to the frame $o_1x_1y_1z_1$, then we must add to the term v the term $R(t)\dot{p}^1$, which is the rate of change of the coordinates p^1 expressed in the frame $o_0x_0y_0z_0$.

Usually, p is attached to frame one, and then p^1 is a constant vector.

Exercises [Extra points!!]

4-9 Given the Euler angle transformation

$$R \quad = \quad R_{z,\psi} R_{y,\theta} R_{z,\phi}$$

show that $\frac{d}{dt} R = S(\omega) R$ where

$$\omega \quad = \quad \{c_\psi s_\theta \dot{\phi} - s_\psi \dot{\theta}\} i + \{s_\psi s_\theta \dot{\phi} + c_\psi \dot{\theta}\} j + \{\dot{[} s i + c_\theta \dot{\phi}\} k.$$

The components of i, j, k, respectively, are called the **nutation**, **spin**, and **precession**.

For the Euler angle transformation, we have

$$R = R_{z,\psi} R_{y,\theta} R_{z,\phi}.$$

From Equation (4.18), we know that

$$\frac{dR_{z,\theta}}{d\theta} = S(k) R_{z,\theta}$$

$$\frac{dR}{d\theta} = S(k) R.$$

By the chain rule for differentiation, we have

$$\dot{R} = \frac{dR}{dt} = \frac{dR}{d\theta}\frac{d\theta}{dt} = S(k)R\dot{\theta}.$$

Applying the product rule for differentiation to the Euler angle transformation, we have

$$\begin{aligned}
\dot{R} &= \dot{R}_z R_y R_z + R_z \dot{R}_y R_z + R_z R_y \dot{R}_z \\
&= [S(\dot{\psi}k) R_{z,\psi}] R_y R_z + R_z [S(\dot{\theta}j) R_{y,\theta}] R_z + R_z R_y [\dot{\phi}k) R_{z,\phi}] \\
&= S(\dot{\psi}k) R_z R_y R_z + S(R_{z,\theta}\dot{\theta}j) R_z R_y R_z + S(R_z R_y \dot{\phi}k) R_z R_y R_z \\
&= [S(\dot{\psi}k) + S(R_z\dot{\theta}j) + S(R_z R_y \dot{\phi}k)] R \\
&= S(\omega) R.
\end{aligned}$$

So

$$\begin{aligned}
\omega &= \dot{\psi}k + R_z\dot{\theta}j + R_z R_y \dot{\phi}k \\
&= (c_\psi s_\theta \dot{\phi} - s_\psi \dot{\theta}) i + (s_\psi s_\theta \dot{\phi} + c_\psi \dot{\theta}) j + (\dot{\psi} + c_\theta \dot{\phi}) k.
\end{aligned}$$

Derivation of the Jacobian

The goal of this section is to relate the linear and angular velocity of the end-effector to the vector of joint velocities $\dot{q}(t)$.

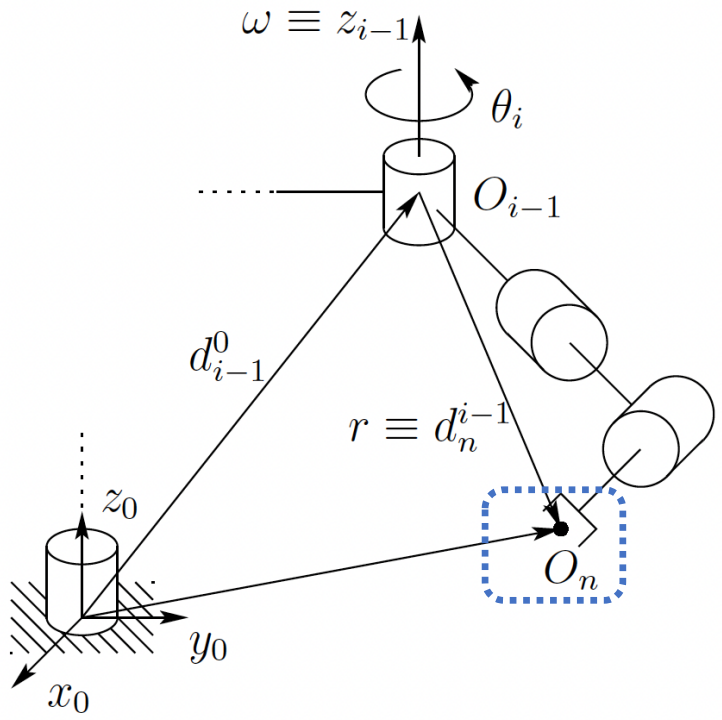


Fig. 4.1 Motion of the end-effector due to link i.

From the well-known equation $\dot{R}^0_n = S(\omega^0_n)R^0_n$, we define the angular velocity vector ω^0_n of the end effector as:

$$S(\omega^0_n) = \dot{R}^0_n \, (R^0_n)^T$$

And let

$$v^0_n = \dot{o}^0_n$$

Denote the linear velocity of the end-effector.
We seek expressions of the form:

$$\begin{matrix} v^0_n = J_v\dot{q} \\ \omega^0_n = J_\omega\dot{q} \end{matrix} \quad \xi = J\dot{q}$$

Where $J_v, J_\omega \in 3 \times n$

$$\xi = \begin{pmatrix} v^0_n \\ \omega^0_n \end{pmatrix}, \qquad J = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix}$$

Body velocity

Jacobian

Angular velocity

The matrices refer to the **0-frame**, also known as the **base frame**, which serves as the primary reference coordinate system.

Recall this equation:

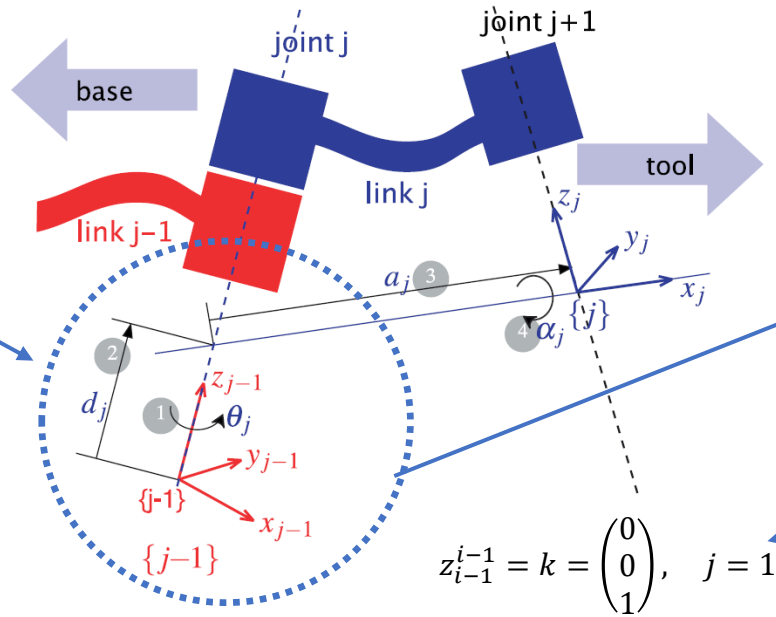
$$\omega_{0,n}^{0} = \omega_{0,1}^{0} + R_1^0\omega_{1,2}^{1} + R_2^0\omega_{2,3}^{2} + R_3^0\omega_{3,4}^{3} + \cdots + R_{n-1}^0\omega_{n-1,n}^{n-1}$$
$$= \omega_{0,1}^{0} + \omega_{1,2}^{0} + \omega_{2,3}^{0} + \omega_{3,4}^{0} + \cdots + \omega_{n-1,n}^{0}$$

Thus, we can determine the angular velocity of the end-effector relative to the base by expressing the angular velocity contributed by each joint in the orientation of the base frame and then summing these.

From the DH table, we know that if joint i is revolute, then the i joint variable $q_i = \theta_i$ and the axis of rotation is z_{i-1}.

If the joint i is prismatic, then the motion of the frame i relative to frame $i-1$ is translation and

$$\omega_i^{i-1} = 0$$



ω_i^{i-1} represents the angular velocity of link i that is imparted by the rotation of joint i, expressed relative to frame $o_{i-1}x_{i-1}y_{i-1}z_{i-1}$. Thus,

$$\omega_i^{i-1} = \dot{q}_i z_{i-1}^{i-1} = \dot{q}_i k$$

$$\omega_{0,n}^{0} = \omega_{0,1}^{0} + R_1^0\omega_{1,2}^{1} + R_2^0\omega_{2,3}^{2} + R_3^0\omega_{3,4}^{3} + \cdots + R_{n-1}^0\omega_{n-1,n}^{n-1}$$
$$= \omega_{0,1}^{0} + \omega_{1,2}^{0} + \omega_{2,3}^{0} + \omega_{3,4}^{0} + \cdots + \omega_{n-1,n}^{0}$$

$\omega_i^{i-1} = \dot{q}_i z_{i-1}^{i-1} = \dot{q}_i k$ ⟶ revolute

$\omega_i^{i-1} = 0$ ⟶ prismatic

Therefore, the overall angular velocity of the end effector, ω_n^0 in the base frame, is determined by:

$$\omega_n^0 = \rho_1\dot{q}_1 k + \rho_2\dot{q}_2 R_1^0 k + \cdots + \rho_n\dot{q}_n R_{n-1}^0 k = \sum_{i-1}^{n} \rho_i\dot{q}_i z_{i-1}^0$$

$$\rho_i = \begin{cases} 1, \text{ if joint } i \text{ is revolute} \\ 0, \text{ if joint } i \text{ is prismatic} \end{cases}$$

$$R_{n-1}^0 = \begin{pmatrix} x_{n-1}^0 & | & y_{n-1}^0 & | & z_{n-1}^0 \end{pmatrix}, \quad k = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad R_{n-1}^0 k = z_{n-1}^0$$

Then, $\omega_n^0 = \sum_{i-1}^{n} \rho_i\dot{q}_i z_{i-1}^0$ and since we want expressions of the form $\omega_n^0 = J_\omega\dot{q}$ it follows that,

$$J_\omega = [\rho_1 z_0 \ \cdots \ \rho_n z_{n-1}] \in 3 \times n$$

Note that in this equation, we have omitted the superscripts for the unit vectors along the z-axes, since these are all referenced to the world frame.

Linear velocity

$$H_n^0(t) = \begin{bmatrix} R_n^0(t) & o_n^0 \\ 0 & 1 \end{bmatrix}$$

The linear velocity of the end effector is just

$$\frac{d}{dt} o_n^0 = \sum_{i=1}^{n} \frac{\partial o_n^0}{\partial q_i} \dot{q}_i$$

Thus, we see that the column i of J_v, denoted by J_{v_i} is given by

$$J_{v_i} = \frac{\partial o_n^0}{\partial q_i}$$

We consider two cases: prismatic and revolute joints separately.

Linear velocity [Prismatic joints]

If joint i is prismatic, then $q_i = d_i$ and it imparts a pure translation to the end-effector.

Let us start by using the homogenous transformation T_n^0 from frame n to frame 0:

$$\begin{aligned}
\begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} &= T_n^0 \\
&= T_{i-1}^0 T_i^{i-1} T_n^i \\
&= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ 0 & 1 \end{bmatrix},
\end{aligned}$$

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0$$

Since only joint i is allowed to move those terms are constant.

Since joint i is prismatic, then R_{I-1}^0 is also constant.

Remember, from the DH convention:

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

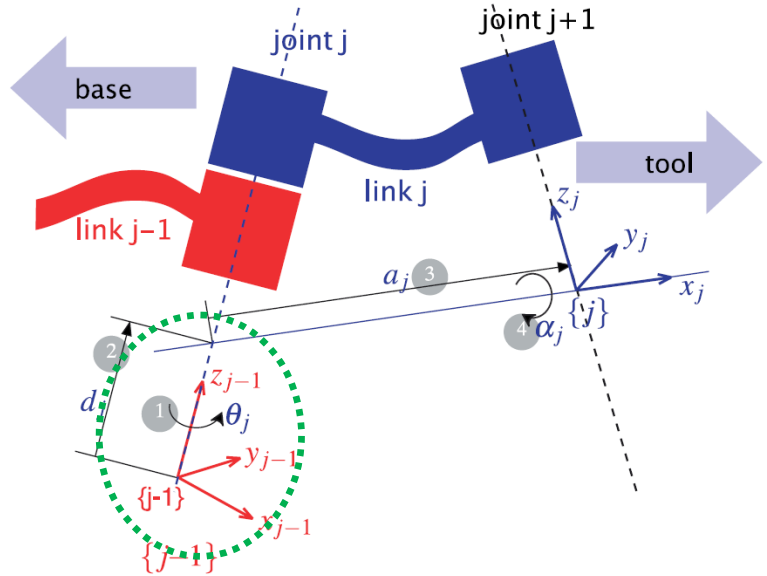
o_i^{i-1}

Then, we get

$$\begin{aligned}
J_{v_i} = \frac{\partial o_n^0}{\partial q_i} &= \frac{\partial}{\partial d_i} R_{i-1}^0 o_i^{i-1} \\
&= R_{i-1}^0 \frac{\partial}{\partial d_i} \begin{bmatrix} a_i c_i \\ a_i s_i \\ d_i \end{bmatrix} \\
&= R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
&= z_{i-1}^0,
\end{aligned}$$

$$J_{v_i} = z_{i-1}$$

Linear velocity [revolute joints]



If joint i is revolute, then $q_i = \theta_i$. Consider the previous origin equation but this time with a time varying $R_i^0(t)$ because we have a revolute joint:

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0$$

And we compute

$$\begin{aligned}
\frac{\partial}{\partial\theta_i} o_n^0 &= \frac{\partial}{\partial\theta_i}\left[R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0\right] \\
&= \frac{\partial}{\partial\theta_i} R_i^0 o_n^i + R_{i-1}^0 \frac{\partial}{\partial\theta_i} o_i^{i-1} \\
&= \dot\theta_i S(z_{i-1}^0) R_i^0 o_n^i + \dot\theta_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \\
&= \dot\theta_i S(z_{i-1}^0)\left[R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}\right] \\
&= \dot\theta_i S(z_{i-1}^0)(o_n^0 - o_{i-1}^0) \\
&= \dot\theta_i z_{i-1}^0 \times (o_n^0 - o_{i-1}^0)
\end{aligned}$$

$$\begin{aligned}
R_{i-1}^0 \frac{\partial}{\partial\theta_i}\begin{bmatrix} a_i c_i \\ a_i s_i \\ d_i \end{bmatrix} &= R_{i-1}^0 \begin{bmatrix} -a_i s_i \\ a_i c_i \\ 0 \end{bmatrix}\dot\theta_i \\
&= R_{i-1}^0 S(k\dot\theta_i) o_i^{i-1} \\
&= R_{i-1}^0 S(k\dot\theta_i)\left(R_{i-1}^0\right)^T R_{i-1}^0 o_i^{i-1} \\
&= S(R_{i-1}^0 k\dot\theta_i) R_{i-1}^0 o_i^{i-1} \\
&= \dot\theta_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1}
\end{aligned}$$

$$J_{v_i} = z_{i-1} \times (o_n - o_{i-1})$$

Exercises

Exercise 1: Skew-Symmetric Matrix Calculation

Given the angular velocity $\omega = [1, 2, 3]^T$, calculate the skew-symmetric matrix associated with it.

Instructions:

1. Use the following formula for the skew-symmetric matrix:

$$\text{Skew}(\omega) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

2. Plug in the values $\omega_x = 1$, $\omega_y = 2$, and $\omega_z = 3$ to obtain the matrix.

Solution:

$$\text{Skew}(\omega) = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix}$$

Exercise 2: Calculation of $\dot{R}$

Given the following rotation matrix R:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

and the angular velocity $\omega = [0, 0, 1]^T$. Calculate the time derivative of R, i.e., $\dot{R}$.

Instructions:

1. Use the skew-symmetric matrix formula for ω.

$$\text{Skew}(\omega) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2. Multiply $\text{Skew}(\omega) \cdot R$ to find $\dot{R}$.

Solution:

$$\dot{R} = \begin{bmatrix} 0 & -\cos(\theta) & \sin(\theta) \\ \cos(\theta) & 0 & 0 \\ -\sin(\theta) & 0 & 0 \end{bmatrix}$$

Exercise 3: Finding Angular Velocity from $\dot{R}$

Given the time derivative of a rotation matrix:

$$\dot{R} = \begin{bmatrix} 0 & -0.5 & 0 \\ 0.5 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

Determine the angular velocity ω.

Instructions:

1. Use the relation: $\text{Skew}(\omega) = \dot{R} \cdot R^T$.
2. Assume that R is an identity matrix (for simplicity, $R = I$), so $\text{Skew}(\omega) = \dot{R}$.
3. Extract ω from the skew-symmetric matrix.

Solution:

$$\text{Skew}(\omega) = \begin{bmatrix} 0 & -0.5 & 0 \\ 0.5 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow \omega = [1, -1, 0.5]^T$$

Combining the linear and angular velocity Jacobians

Body velocity

Jacobian

$$\xi = J\dot{q}$$

$$\begin{pmatrix} v_n^0 \\ \omega_n^0 \end{pmatrix} = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{pmatrix} \longrightarrow \begin{matrix} v_n^0 = J_v\dot{q} \\ \omega_n^0 = J_\omega\dot{q} \end{matrix}$$

Where $J_v, J_\omega \in 3 \times n$

Linear velocity Jacobian

$$J = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} = \begin{matrix} \begin{pmatrix} J_{v_1} & \cdots & J_{v_n} \end{pmatrix} \in 3 \times n \\ \begin{pmatrix} J_{\omega_1} & \cdots & J_{\omega_n} \end{pmatrix} \in 3 \times n \end{matrix}$$

Angular velocity Jacobian

The column i of J_v is:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

The column i of J_ω is:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$

Combining the linear and angular velocity Jacobians

The column i of J_v is:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

The column i of J_ω is:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$

Therefore, to compute the Jacobian, we need:

1. The unit vectors z_i
2. The coordinates of the origins $o_1, o_2, \cdots, o_n$.

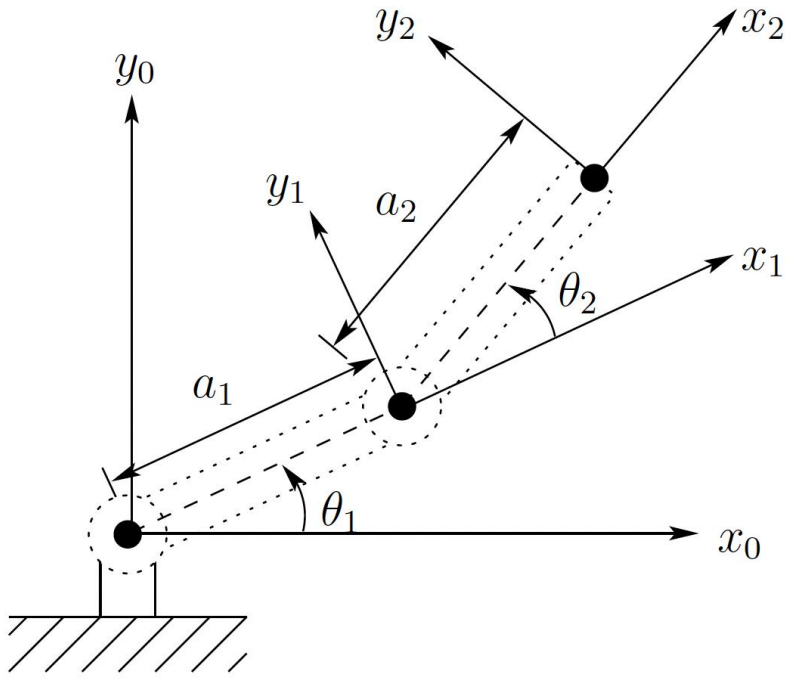
Equivalently, we need:

1. The first three elements in the third column of T_i^0.
2. The first three elements of the fourth column of T_i^0.

To compute the Jacobian, we just need the homogenous transformations T_i^0 that conform the Forward kinematics!!

Example

Let us compute the Jacobian for the two-link manipulator. For that aim, we need to recall the forward kinematics



The column i of J_v is:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

The column i of J_ω is:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$

We have two links, so $i = \{1,2\}$. An the Jacobian wil

Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1^*
2	a_2	0	0	θ_2^*

$$v_n^0 = J_v \dot{q}$$
$$\omega_n^0 = J_\omega \dot{q}$$

J_v is:

$$J_v = [J_{v_1} \quad J_{v_2}] = [z_0 \times (o_2 - o_0) \quad z_1 \times (o_2 - o_1)]$$

While J_ω is:

$$J_\omega = [J_{\omega_1} \quad J_{\omega_2}] = [z_0 \quad z_1]$$

$$J(q) = \begin{bmatrix} z_0 \times (o_2 - o_0) & z_1 \times (o_2 - o_1) \\ z_0 & z_1 \end{bmatrix}$$

We need to find those elements from the Forward kinematics.

Recall the structure of the homogenous transformation matrix:

$$T_i^0 = \begin{bmatrix} R_i^0 & o_i^0 \\ 0 & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} x_i^0 & y_i^0 & z_i^0 & o_i^0 \\ & 0 & & 1 \end{array}\right]$$

4th column

3rd column

$$T_i^0 = \begin{bmatrix} R_i^0 & o_i^0 \\ 0 & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} x_i^0 & y_i^0 & z_i^0 & o_i^0 \\ & 0 & & 1 \end{array}\right]$$

$$J(q) = \begin{bmatrix} z_0 \times (o_2 - o_0) & z_1 \times (o_2 - o_1) \\ z_0 & z_1 \end{bmatrix}$$

Forward kinematics terms [homogenous transformations] for the two-link planar manipulator:

$$z_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \qquad o_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

z_1 o_1

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1 c_1 \\ s_1 & c_1 & 0 & a_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^0 = A_1$$

o_2

$$T_2^0 = A_1 A_2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1 c_1 + a_2 c_{12} \\ s_{12} & c_{12} & 0 & a_1 s_1 + a_2 s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Performing the required calculations then yields

$$J = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$s_{12} = \sin(\theta_1 + \theta_2)$

$$J(q) = \begin{bmatrix} z_0 \times (o_2 - o_0) & z_1 \times (o_2 - o_1) \\ z_0 & z_1 \end{bmatrix}$$

Exercise 1

Make a code in python that compute the previous result given the vectors of J with values of $a_1 = a_2 = 1$, and $\theta_1 = \theta_2 = 45$ deg.

robotArmTAMIU > jacobian.py > ...

```python
import numpy as np

# Given parameters
a1 = 1.0  # Length of the first link
a2 = 1.0  # Length of the second link
theta1 = np.deg2rad(45)  # Angle of the first joint in radians
theta2 = np.deg2rad(45)  # Angle of the second joint in radians

# Define the positions of the points
o0 = np.array([0, 0, 0])
o1 = np.array([a1 * np.cos(theta1), a1 * np.sin(theta1), 0])
o2 = np.array([a1 * np.cos(theta1) + a2 * np.cos(theta1 + theta2),
               a1 * np.sin(theta1) + a2 * np.sin(theta1 + theta2), 0])

z0 = np.array([0, 0, 1])  # Rotation axis of the first joint
z1 = np.array([0, 0, 1])  # Rotation axis of the second joint
```

```python

# Compute the Jacobian columns for linear velocity
J_v1 = np.cross(z0, o2 - o0)  # Linear velocity with respect to theta1
J_v2 = np.cross(z1, o2 - o1)  # Linear velocity with respect to theta2

# Build the angular velocity part of the Jacobian
J_w1 = z0  # Angular velocity with respect to theta1
J_w2 = z1  # Angular velocity with respect to theta2

# Construct the complete Jacobian (6x2 matrix)
J = np.hstack([  # Stack rows horizontally
    np.array([J_v1.T, J_v2.T]),  # Linear velocity part (3x2)
    np.array([J_w1.T, J_w2.T])   # Angular velocity part (3x2)
])

# Display the corrected Jacobian
print("Corrected Jacobian:")
print(J.T)  # Transpose to get the 6x2 matrix
```

Example: SCARA Manipulator

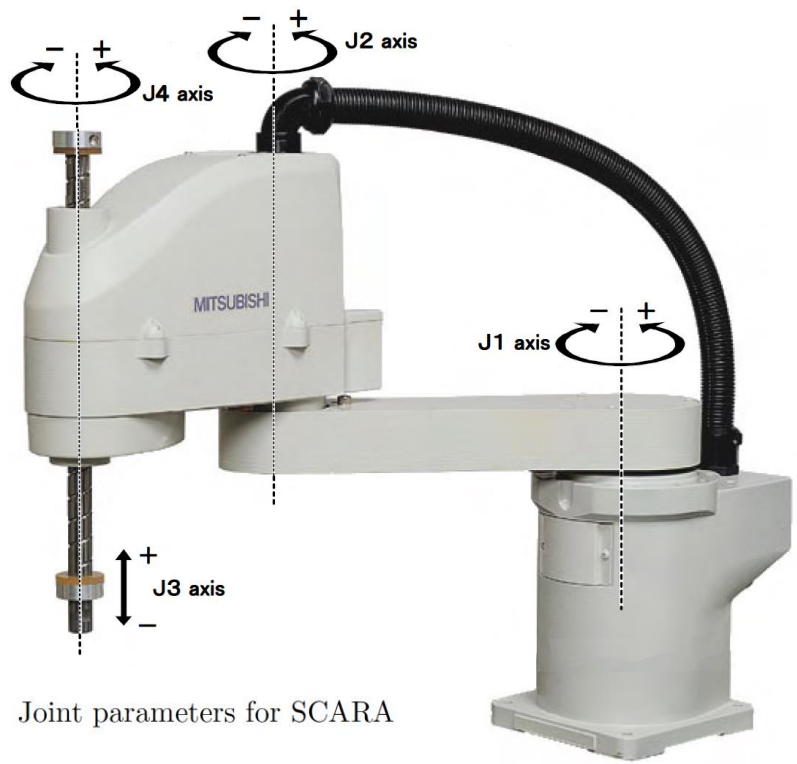


Table 3.5 Joint parameters for SCARA

Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	$\theta^\star$
2	a_2	180	0	$\theta^\star$
3	0	0	$d^\star$	0
4	0	0	d_4	$\theta^\star$

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & s_2 & 0 & a_2c_2 \\ s_2 & -c_2 & 0 & a_2s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = A_1 \cdots A_4$$

$$= \begin{bmatrix} c_{12}c_4 + s_{12}s_4 & -c_{12}s_4 + s_{12}c_4 & 0 & a_1c_1 + a_2c_{12} \\ s_{12}c_4 - c_{12}s_4 & -s_{12}s_4 - c_{12}c_4 & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

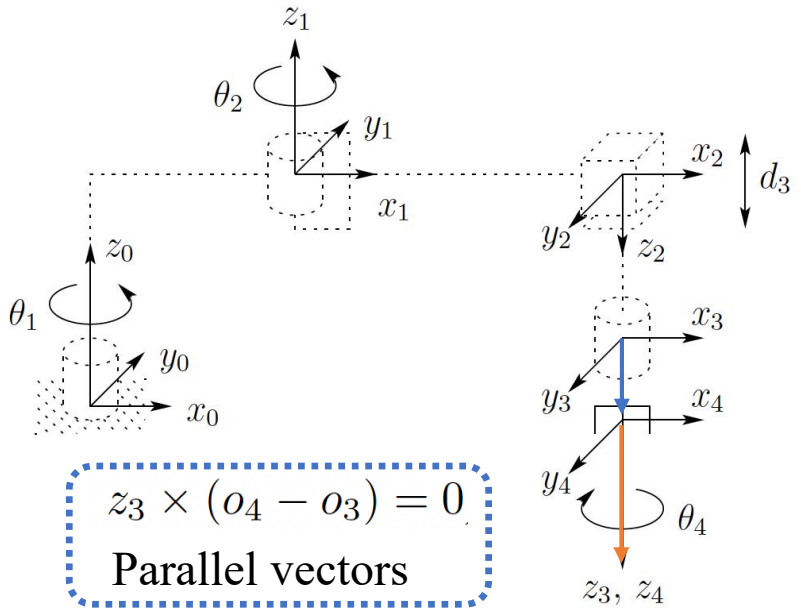
Example: SCARA Manipulator

The column i of J_v is:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

The column i of J_ω is:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$



$z_3 \times (o_4 - o_3) = 0$

Parallel vectors

R R P R

$$J = \begin{bmatrix} z_0 \times (o_4 - o_0) & z_1 \times (o_4 - o_1) & z_2 & z_3 \times (o_4 - o_3) \\ z_0 & z_1 & 0 & z_3 \end{bmatrix}$$

Fig. 3.11 DH coordinate frame assignment for the SCARA manipulator

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1 c_1 \\ s_1 & c_1 & 0 & a_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & s_2 & 0 & a_2 c_2 \\ s_2 & -c_2 & 0 & a_2 s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = A_1 \cdots A_4$$

$$= \begin{bmatrix} c_{12}c_4 + s_{12}s_4 & -c_{12}s_4 + s_{12}c_4 & 0 & a_1 c_1 + a_2 c_{12} \\ s_{12}c_4 - c_{12}s_4 & -s_{12}s_4 - c_{12}c_4 & 0 & a_1 s_1 + a_2 s_{12} \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Table 3.5 Joint parameters for SCARA

Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	$\theta^\star$
2	a_2	180	0	$\theta^\star$
3	0	0	$d^\star$	0
4	0	0	d_4	$\theta^\star$

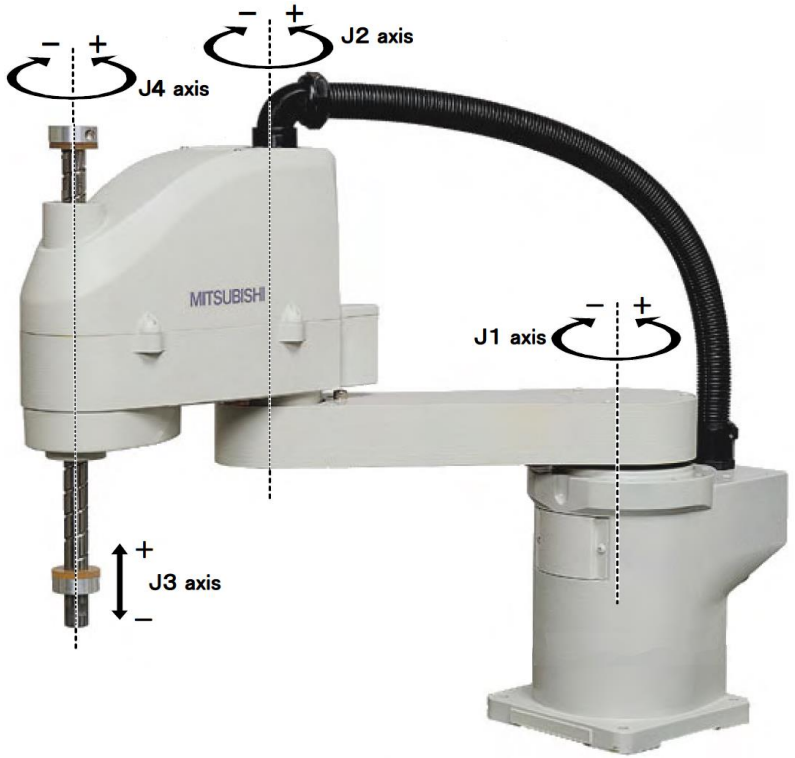
Exercise: compute the required elements to get J.

For that goal we need to compute:

- $A_1 A_2$ to get z_2
- $A_1 A_2 A_3$ to get z_3
- $A_1 A_2 A_3 A_4$ to get o_4
- All the operations in J.

Example: SCARA Manipulator

$$J = \begin{bmatrix} z_0 \times (o_4 - o_0) & z_1 \times (o_4 - o_1) & z_2 & 0 \\ z_0 & z_1 & 0 & z_3 \end{bmatrix}$$



$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & s_2 & 0 & a_2c_2 \\ s_2 & -c_2 & 0 & a_2s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Table 3.5 Joint parameters for SCARA

Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	$\theta^\star$
2	a_2	180	0	$\theta^\star$
3	0	0	$d^\star$	0
4	0	0	d_4	$\theta^\star$

$$T_4^0 = A_1 \cdots A_4 = \begin{bmatrix} c_{12}c_4 + s_{12}s_4 & -c_{12}s_4 + s_{12}c_4 & 0 & a_1c_1 + a_2c_{12} \\ s_{12}c_4 - c_{12}s_4 & -s_{12}s_4 - c_{12}c_4 & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Performing the indicated calculations, one obtains

$$o_1 = \begin{bmatrix} a_1c_1 \\ a_1s_1 \\ 0 \end{bmatrix} \qquad o_2 = \begin{bmatrix} a_1c_1 + a_2c_{12} \\ a_1s_1 + a_2s_{12} \\ 0 \end{bmatrix} \tag{4.96}$$

$$o_4 = \begin{bmatrix} a_1c_1 + a_2c_{12} \\ a_1s_2 + a_2s_{12} \\ d_3 - d_4 \end{bmatrix} \tag{4.97}$$

Similarly $z_0 = z_1 = k$, and $z_2 = z_3 = -k$. Therefore the Jacobian of the SCARA Manipulator is

$$J = \begin{bmatrix} -a_1s_1 - a_2s_{12} & -a_2s_{12} & 0 & 0 \\ a_1c_1 + a_2c_{12} & a_2c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} \tag{4.98}$$

Exercise 2

$$J = \begin{bmatrix} z_0 \times (o_4 - o_0) & z_1 \times (o_4 - o_1) & z_2 & 0 \\ z_0 & z_1 & 0 & z_3 \end{bmatrix} \quad J = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} & 0 & 0 \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

Make a code in python that compute the previous result given the vectors of J with values of $a_1 = 3, a_2 = 2$, and $\theta_1 = 30,\ \theta_2 = 45$ deg.

robotArmTAMIU > jacobianSCARA.py > ...

```python
import numpy as np

# Given parameters
a1 = 3.0  # Length of the first link
a2 = 2.0  # Length of the second link
theta1 = np.deg2rad(30)  # Angle of the first joint in radians
theta2 = np.deg2rad(45)  # Angle of the second joint in radians

# Define the sine and cosine terms for readability
c1 = np.cos(theta1)
s1 = np.sin(theta1)
c12 = np.cos(theta1 + theta2)
s12 = np.sin(theta1 + theta2)

# Define the positions of the points as numpy arrays
o0 = np.array([0, 0, 0])  # Origin
o1 = np.array([a1 * c1, a1 * s1, 0])  # First joint
o4 = np.array([a1 * c1 + a2 * c12, a1 * s1 + a2 * s12, -1])  # End-effector position (z component d3 - d4 = -1)

# Define the rotation axes as specified
z0 = np.array([0, 0, 1])  # Rotation axis for the first joint
z1 = np.array([0, 0, 1])  # Rotation axis for the second joint
z2 = np.array([0, 0, -1])  # Rotation axis for the third component
z3 = np.array([0, 0, -1])  # Rotation axis for the fourth component
```

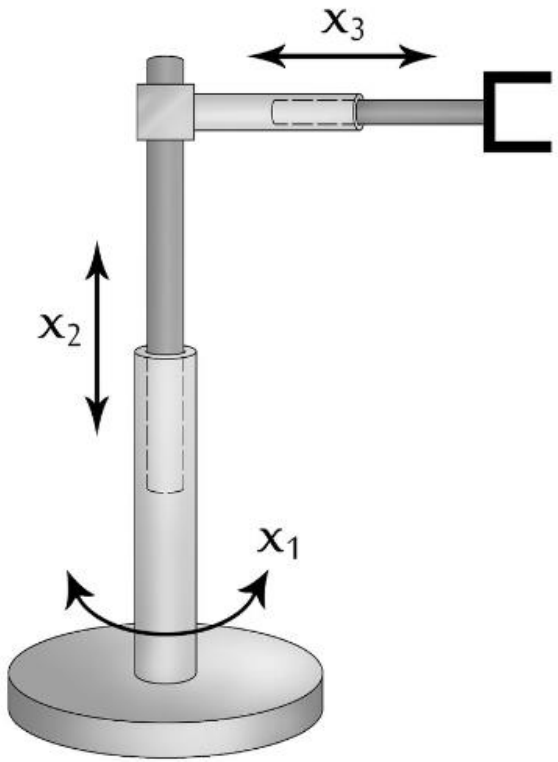
Exercise 2

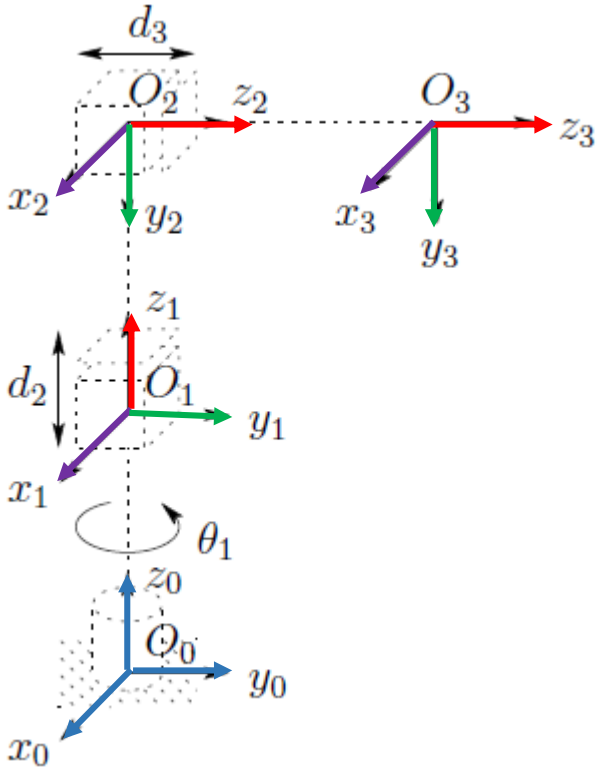
$$J = \begin{bmatrix} z_0 \times (o_4 - o_0) & z_1 \times (o_4 - o_1) & z_2 & 0 \\ z_0 & z_1 & 0 & z_3 \end{bmatrix} \quad J = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} & 0 & 0 \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

Make a code in python that compute the previous result given the vectors of J with values of $a_1 = 3, a_2 = 2$, and $\theta_1 = 30,\ \theta_2 = 45$ deg.

```python

# Compute the Jacobian columns for linear velocity
J_v1 = np.cross(z0, o4 - o0)  # Linear velocity contribution from the first joint
J_v2 = np.cross(z1, o4 - o1)  # Linear velocity contribution from the second joint
J_v3 = z2  # Contribution from the third axis
J_v4 = np.array([0, 0, 0])  # No linear contribution for the fourth axis

# Build the angular velocity part of the Jacobian
J_w1 = z0  # Angular velocity contribution from the first joint
J_w2 = z1  # Angular velocity contribution from the second joint
J_w3 = np.array([0, 0, 0])  # No angular contribution from the third axis
J_w4 = z3  # Angular velocity contribution from the fourth axis

# Construct the complete Jacobian (6x4 matrix)
J = np.hstack([
    np.array([J_v1, J_v2, J_v3, J_v4]),  # Linear velocity part (3x4)
    np.array([J_w1, J_w2, J_w3, J_w4])   # Angular velocity part (3x4)
])

# Display the calculated Jacobian
print("Calculated Jacobian:")
print(J.T)

```

Example 3: Three-Link Cylindrical Robot





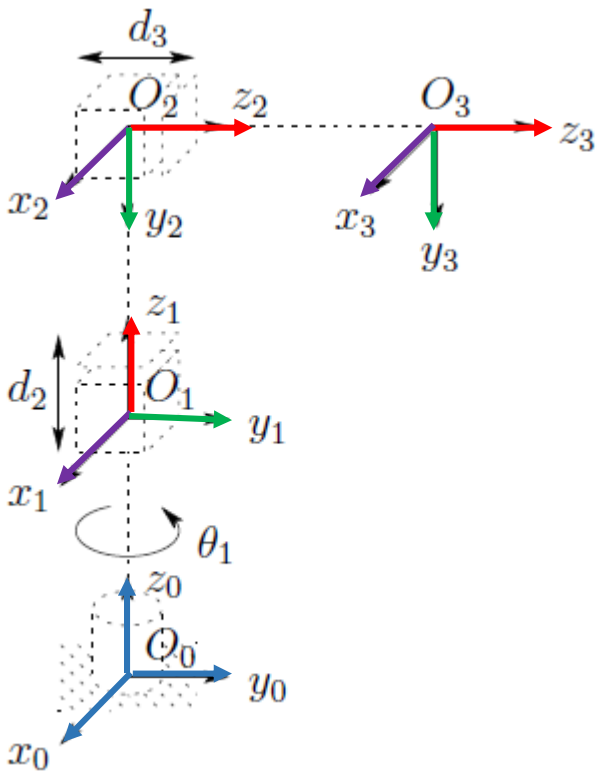
$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Link	a_i	α_i	d_i	θ_i
1	0	0	d_1	θ_1^*
2	0	−90	d_2^*	0
3	0	0	d_3^*	0

* variable

Example 3: Three-Link Cylindrical Robot



The column i of J_v is:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

The column i of J_ω is:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$

$$J = \begin{bmatrix} z_0 \times (o_3 - o_0) & z_1 & z_2 \\ z_0 & 0 & 0 \end{bmatrix}$$

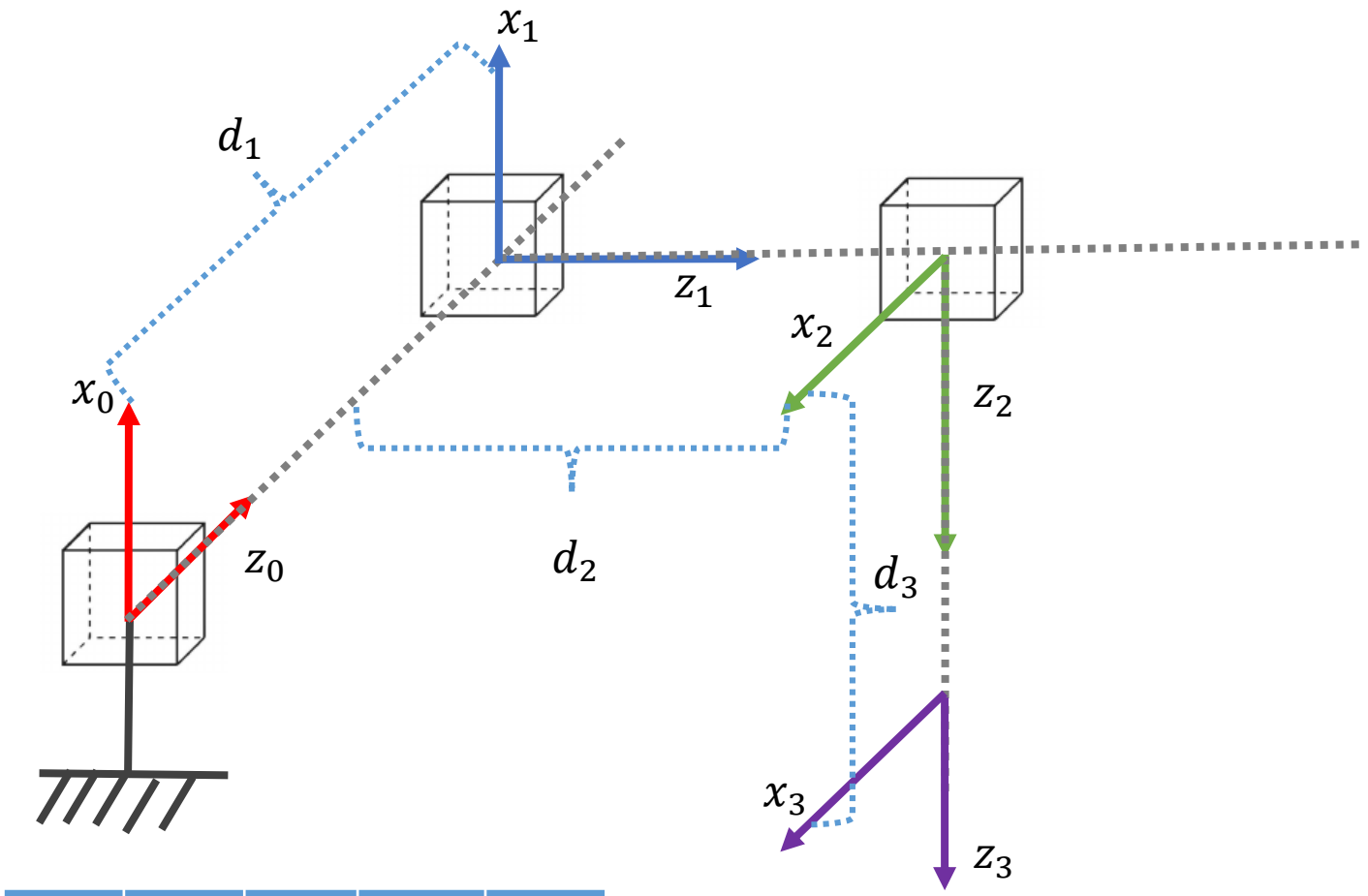
Robot configuration: **R P P**
Degrees of freedom: **n**

Exercise 3

$$J = \begin{bmatrix} z_0 \times (o_3 - o_0) & z_1 & z_2 \\ z_0 & 0 & 0 \end{bmatrix}$$

- Make a code in python that compute the previous result given the vectors of J with values of $d_3 = 3$, and $\theta_1 = 90$.
- Is the linear velocity Jacobian singular? What are the implications of a singular Jacobian?

Example 4: Three-link cartesian robot



Link	a_i	α_i	d_i	θ_i
1	0	-90	d_1^*	0
2	0	-90	d_2^*	90
3	0	0	d_3^*	0

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_1^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & d_2^* \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3^* \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^3 = A_1 A_2 A_3 = \begin{bmatrix} 0 & 0 & -1 & -d_3^* \\ 0 & -1 & 0 & d_2^* \\ -1 & 0 & 0 & d_1^* \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$J = \begin{bmatrix} z_0 & z_1 & z_2 \\ 0 & 0 & 0 \end{bmatrix}$$

Exercise 4

Make a code in python that compute the previous result given the vectors of J.

Jacobian of an arbitrary point

Suppose now that we want to compute the linear velocity v and the angular velocity ω of the center of link 2 as shown.

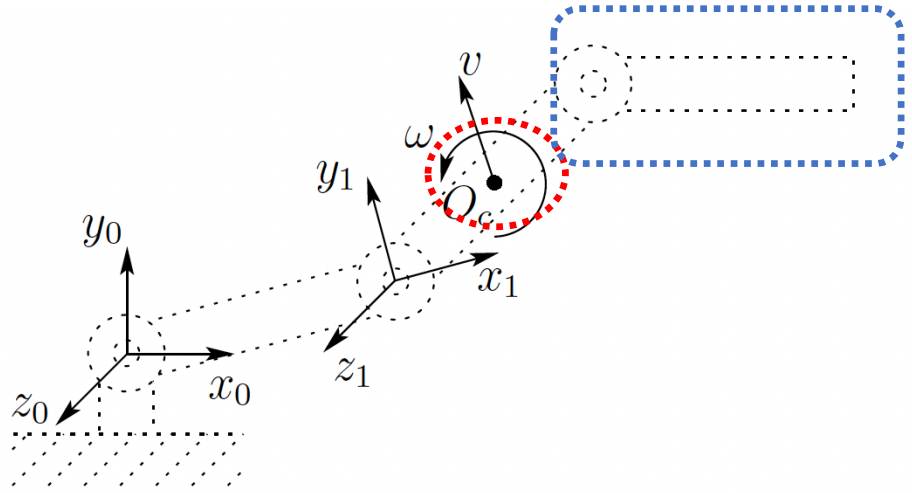
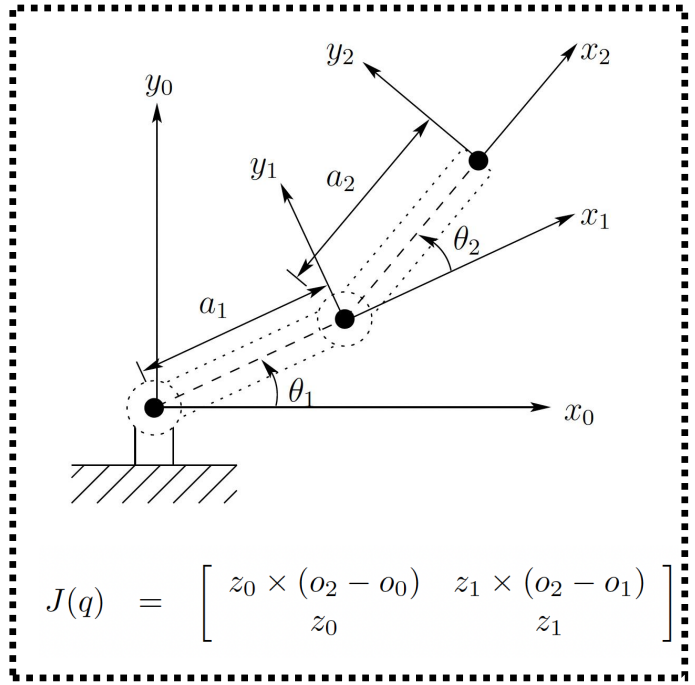


Fig. 4.2 Finding the velocity of link 2 of a 3-link planar robot.

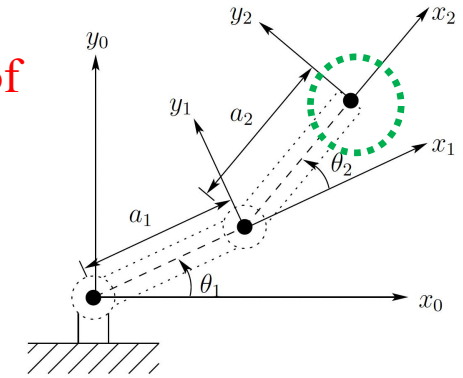
This vector is zero since the velocity of the second link is unaffected by motion of the 3rd link

$$J(q) = \begin{bmatrix} z_0 \times (o_c - o_0) & z_1 \times (o_c - o_1) & 0 \\ z_0 & z_1 & 0 \end{bmatrix}$$



We just changed the coordinate of the last frame. And the vector o_c must be computed as it is not given directly by the T matrices.

Recall the forward kinematics of the planar robot.



$$J(q) = \begin{bmatrix} z_0 \times (o_2 - o_0) & z_1 \times (o_2 - o_1) \\ z_0 & z_1 \end{bmatrix}$$

Forward kinematics terms [homogenous transformations] for the two-link planar manipulator:

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1 c_1 \\ s_1 & c_1 & 0 & a_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^0 = A_1$$

$$T_2^0 = A_1 A_2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1 c_1 + a_2 c_{12} \\ s_{12} & c_{12} & 0 & a_1 s_1 + a_2 s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

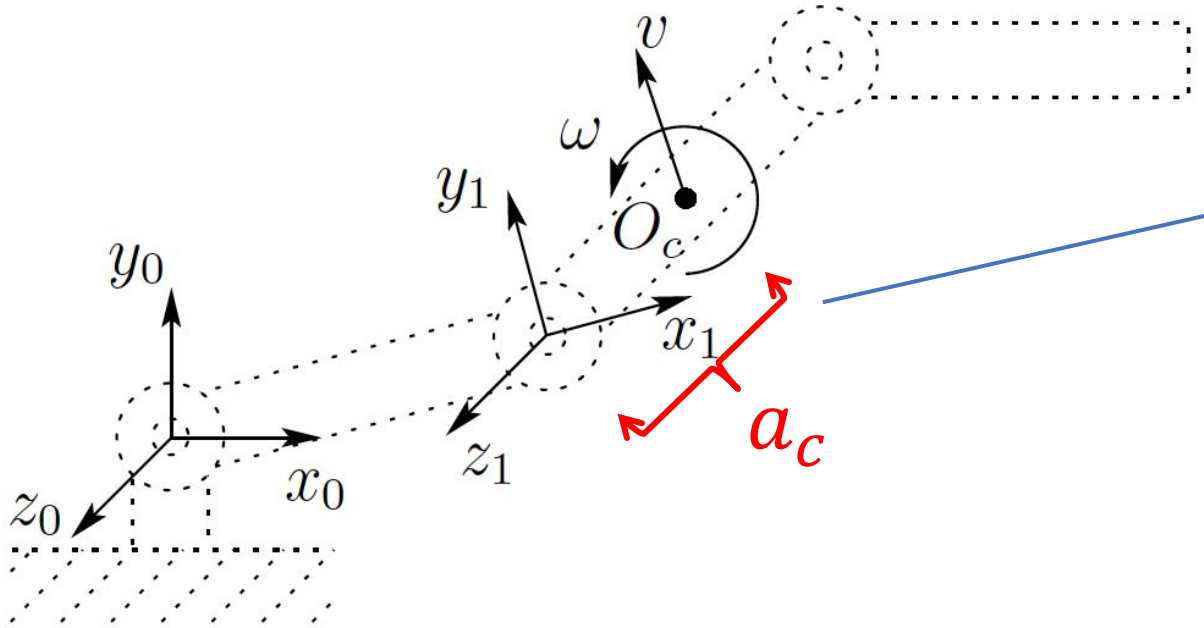
o_2

Performing the required calculations then yields

$$J = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$s_{12} = \sin(\theta_1 + \theta_2)$

We use the previous result and substitute the corresponding values [a_2 by a_c] as follows:



$$
\begin{aligned}
x_c &= a_1c_1 + a_cc_{12} \\
y_c &= a_1s_1 + a_cc_{12} \\
z_c &= 0
\end{aligned}
$$

$$
\begin{aligned}
z_0 &= z_1 = (0,0,1)^T \\
o_0 &= (0,0,0)^T \\
o_c &= (a_1c_1 + a_cc_{12}, a_1s_1 + a_cs_{21}, 0)^T \\
o_1 &= (a_1c_1, a_1s_1, 0)^T
\end{aligned}
$$

$$
\begin{aligned}
z_0 \times (o_c - o_0) &= (-a_1s_1 - a_cs_{12}, a_cc_1 + a_cc_{12}, 0)^T \\
z_1 \times (o_c - o_1) &= (-a_cs_{12}, a_cc_{12}, 0)^T
\end{aligned}
$$

The results is

$$
J = \begin{bmatrix}
-a_1s_1 - a_1s_{12} & -a_cs_{12} & 0 \\
a_1c_1 + a_cc_{12} & a_cc_{12} & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
1 & 1 & 0
\end{bmatrix}
$$

Thus, we can use the forward kinematics matrices by properly changing the coordinated of the point of interest.

The use of Jacobians in control: motivation

We seek expressions of the form:

$$v_n^0 = J_v \dot{q}$$
$$\omega_n^0 = J_\omega \dot{q}$$

$$\xi = J\dot{q}$$

Where $J_v, J_\omega \in 3 \times n$

$$\xi = \begin{pmatrix} v_n^0 \\ \omega_n^0 \end{pmatrix}, \qquad J = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix}$$

Body velocity

Jacobian

$$\xi = J\dot{q}$$

$$\dot{q} = J^{-1}\xi$$

We can control the joints of the robot given by q by modifying the velocity vector ξ!

The goal is to find ξ such that the end effector achieves a desired position.

For that, we should define an error function that considers rotations and translations, i.e., an error function in the form of homogenous transformations [HG].

Let

$$T_c = \begin{bmatrix} R_c & p_c \\ 0 & 1 \end{bmatrix}, \qquad T_d = \begin{bmatrix} R_d & p_d \\ 0 & 1 \end{bmatrix},$$

where T_c is the current HG representing the actual pose of the end effector, and T_d the desired HG that represents the desired position and orientation of the end effector.

Thus, we define the error function:

$$e_{position} = p_d - p_c, \qquad R_{error} = R_d R_c^T$$

Since we want an error vector as an element of $\mathbb{R}^6$, we use the following transformation for the error matrix:

$$e_{orientation} = vee(\log R_{error})$$

Example: Using the Vee Operator

If we have a skew-symmetric matrix $\mathbf{S}$:

$$\mathbf{S} = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix}$$

Applying the vee operator:

$$\text{vee}(\mathbf{S}) = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Computation of $e_{orientation}$

Matrix Logarithm: For a rotation matrix $\mathbf{R}_{\text{error}}$ that represents a rotation by an angle θ around a unit axis $\mathbf{u} = [u_x, u_y, u_z]^T$, the matrix logarithm $\log(\mathbf{R}_{\text{error}})$ gives a skew-symmetric matrix $\mathbf{S}$ such that:

$$\log(\mathbf{R}_{\text{error}}) = \theta \mathbf{S}$$

where $\mathbf{S}$ is the skew-symmetric matrix associated with $\mathbf{u}$:

$$\mathbf{S} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

Calculating θ and $\mathbf{u}$:

- The angle θ of rotation can be computed from $\mathbf{R}_{\text{error}}$ as:

$$\theta = \cos^{-1}\left(\frac{\text{trace}(\mathbf{R}_{\text{error}}) - 1}{2}\right)$$

- The rotation axis $\mathbf{u}$ can be extracted (assuming $\theta \neq 0$) from the elements of $\mathbf{R}_{\text{error}}$:

$$\mathbf{u} = \frac{1}{2\sin(\theta)} \begin{bmatrix} \mathbf{R}_{\text{error}}[2,1] - \mathbf{R}_{\text{error}}[1,2] \\ \mathbf{R}_{\text{error}}[0,2] - \mathbf{R}_{\text{error}}[2,0] \\ \mathbf{R}_{\text{error}}[1,0] - \mathbf{R}_{\text{error}}[0,1] \end{bmatrix}$$

Computation of $e_{orientation}$

Vee Operator (Converting to Angular Error Vector): To obtain a 3D angular error vector $\mathbf{e}_{\text{orientation}}$, we use the "vee" operator, which converts the skew-symmetric matrix $\theta\mathbf{S}$ into a vector:

$$\mathbf{e}_{\text{orientation}} = \theta\mathbf{u}$$

This vector $\mathbf{e}_{\text{orientation}}$ represents the **axis-angle** form of the orientation error, where the direction of $\mathbf{e}_{\text{orientation}}$ is along $\mathbf{u}$ and its magnitude is θ.

So, the orientation error $\mathbf{e}_{\text{orientation}}$ is computed as:

$$\mathbf{e}_{\text{orientation}} = \text{vee}(\log(\mathbf{R}_{\text{error}})) = \theta\mathbf{u}$$

This vector can then be combined with the position error to create a full 6D error vector for use in control.

Once we have defined the errors:

$$e_{position} = p_d - p_c,$$
$$e_{orientation} = vee(\log R_{error})$$

$$e = \begin{bmatrix} e_{position} \\ e_{orientation} \end{bmatrix} \in \mathbb{R}^6$$

$$\xi = J\dot{q}$$

We can now proceed to propose a controller given by:

$$v = \xi = -k\mathrm{e} \in \mathbb{R}^6$$

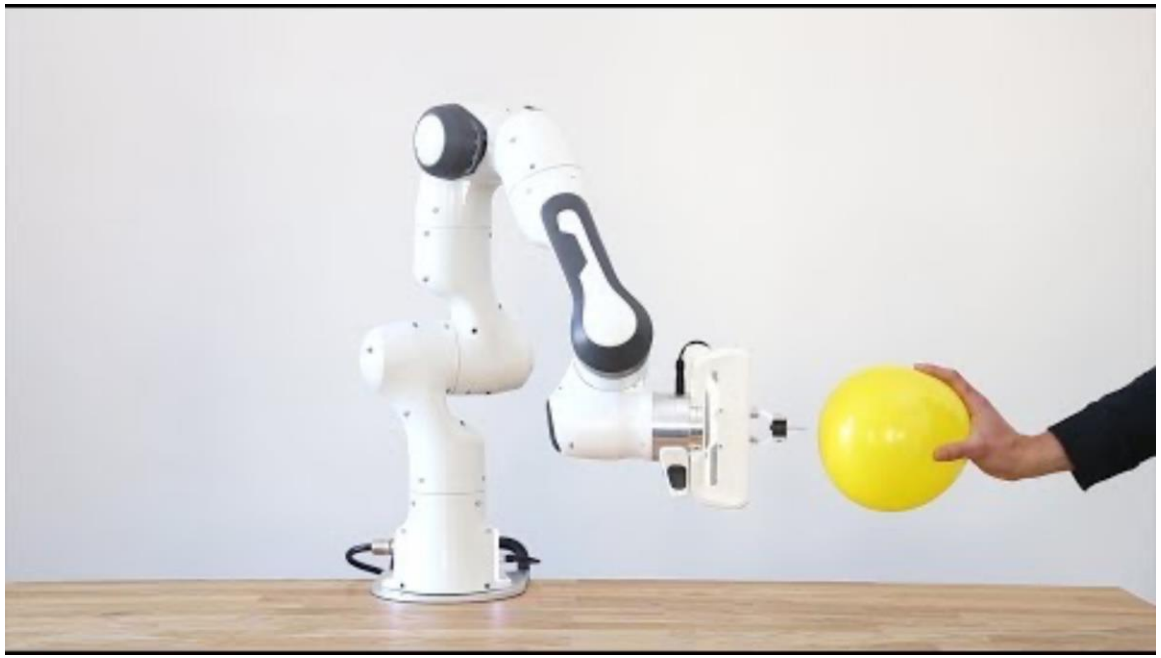
where k is a positive real number. We substitute such control in the kinematics equation above:

$$\xi = J\dot{q}$$
$$J^{-1}\,(\xi = J\dot{q})$$
$$J^{-1}\xi = \dot{q}$$
$$-J^{-1}k\mathrm{e} = \dot{q}$$

$\mathbb{R}^{n\times 6}$ $\mathbb{R}^{1\times 1}$ $\mathbb{R}^{6\times 1}$ $\mathbb{R}^{n\times 1}$

$$\dot{q} = \underbrace{-J^{-1}k\mathrm{e}}_{v}$$

The Panda robot



Code example

test2CorkeVideo.py > ...

```python
import swift
import roboticstoolbox as rtb
import spatialmath as sm
import numpy as np

env = swift.Swift()
env.launch(realtime=True)

panda = rtb.models.Panda()
panda.q = panda.qr

Tep = panda.fkine(panda.q) * sm.SE3.Trans(-0.2, -0.2, 0.45)

arrived = False
env.add(panda)
```

Code example

https://youtu.be/E_Q8NQSN5kc?si=1aBo0N9-Sq8LMhyC

```
16
17  dt = 0.05
18
19  while not arrived:
20
21      v, arrived = rtb.p_servo(panda.fkine(panda.q), Tep, 1)
22      panda.qd = np.linalg.pinv(panda.jacobe(panda.q)) @ v
23      env.step(dt)
24
25  # Uncomment to stop the browser tab from closing
26  # env.hold()
```

```
import swift
import roboticstoolbox as rtb
import spatialmath as sm
import numpy as np
```

These libraries allow you to create a simulation environment (Swift), control the robot (Robotics Toolbox), perform spatial math operations (SpatialMath), and work with arrays (NumPy).

```
env = swift.Swift()
env.launch(realtime=True)
```

Here, a simulation environment named env is created and launched in real-time with realtime=True.

```
panda = rtb.models.Panda()
panda.q = panda.qr
```

- panda = rtb.models.Panda() loads the Panda robot model.
- panda.q = panda.qr sets the robot's initial joint configuration to a predefined resting position (qr).

```
Tep = panda.fkine(panda.q) * sm.SE3.Trans(-0.2, -0.2, 0.45)
```

$$T_{ep} = T_c T_d$$

- Tep is the desired target position for the robot's end-effector, calculated from the initial position using forward kinematics (fkine).
- sm.SE3.Trans(-0.2, -0.2, 0.45) creates a transformation that translates the initial position by x=−0.2x = -0.2x=−0.2, y=−0.2y = -0.2y=−0.2, and z=0.45z = 0.45z=0.45.

Thus, Tep becomes the desired end position (target end position) for the robot's end-effector.

```
env.add(panda)
```

This adds the panda robot to the simulation environment env, making it visible and interactive.

```
dt = 0.05
while not arrived:
    v, arrived = rtb.p_servo(panda.fkine(panda.q), Tep, 1)
    panda.qd = np.linalg.pinv(panda.jacobe(panda.q)) @ v
    env.step(dt)
```

Homogenous transformations

`v, arrived = rtb.p_servo(current_pose, target_pose, gain)`

This function computes the velocity required to reduce the error between the robot's current end-effector position and the desired target position.

Velocity [control]

This while loop moves the robot towards the target position (Tep) using **visual servoing control** (p_servo), which calculates a velocity to reach the target position.

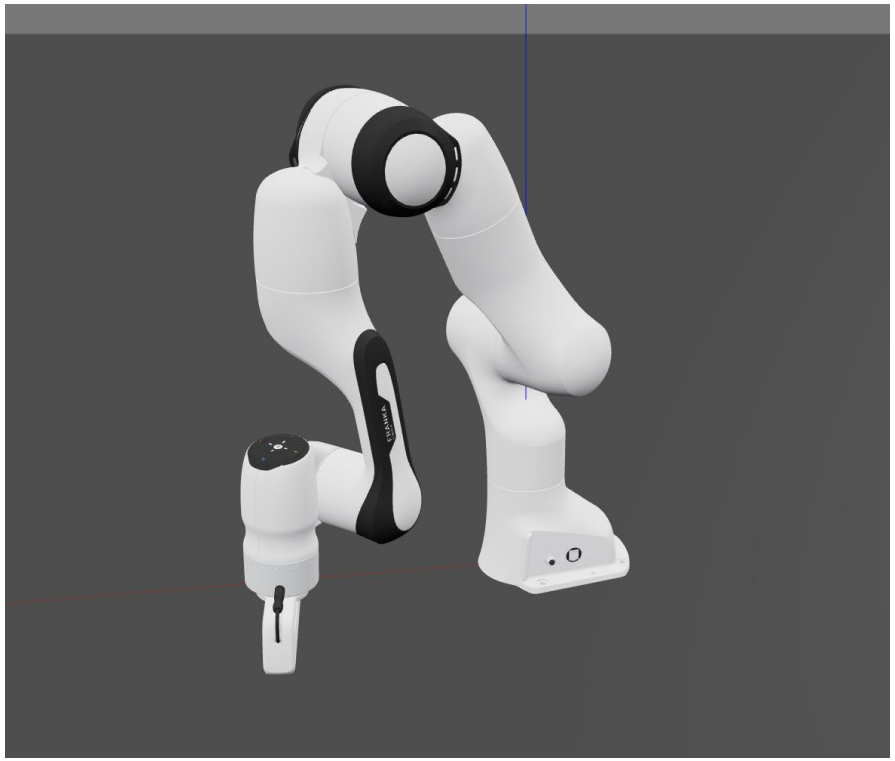
•**Step-by-Step**:

- v, arrived = rtb.p_servo(panda.fkine(panda.q), Tep, 1): calculates the required linear and angular velocity (v) to reach Tep. It returns arrived = True when the robot is close enough to the target position.
- panda.qd = np.linalg.pinv(panda.jacobe(panda.q)) @ v: uses the **pseudoinverse of the Jacobian** (jacobe) to calculate the required joint velocities qd to achieve the desired end-effector velocity v.
- env.step(dt) : advances the simulation environment by a time step of dt = 0.05 seconds.

The loop ends when arrived becomes True, meaning the robot has reached the target position.

Extra points!

Run the previous code on your laptop and share the result with me to earn extra points on the exam. The panda robot much achieve the desired position.



The Euler Lagrange equations

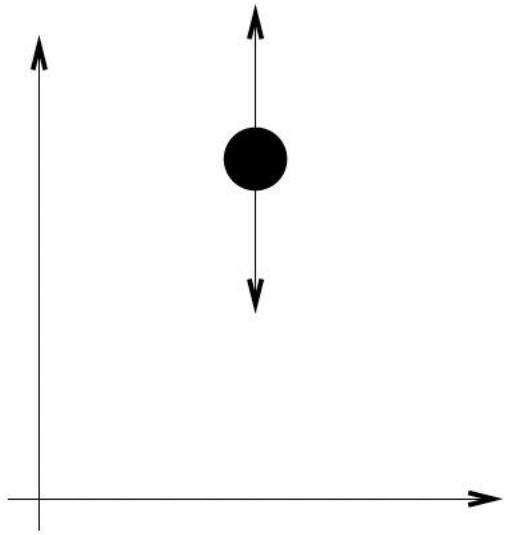


Fig. 6.1 One Degree of Freedom System

$$m\ddot{y} = f - mg$$

$$m\ddot{y} = \frac{d}{dt}(m\dot{y}) = \frac{d}{dt}\frac{\partial}{\partial \dot{y}}\left(\frac{1}{2}m\dot{y}^2\right) = \frac{d}{dt}\frac{\partial \mathcal{K}}{\partial \dot{y}}$$

$\mathcal{K} = \frac{1}{2}m\dot{y}^2$ is the **kinetic energy**

$$m\ddot{y} = \frac{d}{dt}(m\dot{y}) = \frac{d}{dt}\frac{\partial}{\partial \dot{y}}\left(\frac{1}{2}m\dot{y}^2\right) = \frac{d}{dt}\frac{\partial \mathcal{K}}{\partial \dot{y}}$$

$$mg = \frac{\partial}{\partial y}(mgy) = \frac{\partial \mathcal{P}}{\partial y}$$

$\mathcal{K} = \frac{1}{2}m\dot{y}^2$ is the **kinetic energy**.

$\mathcal{P} = mgy$ is the **potential energy due to gravity.**

→ Important definitions

We define:

$$\mathcal{L} = \mathcal{K} - \mathcal{P} = \frac{1}{2}m\dot{y}^2 - mgy$$

→ The Lagrangian

Notice that we can write $m\ddot{y} = f - mg$ as follows:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = f$$

→ The Euler Lagrange equation

$$\mathcal{L} = \mathcal{K} - \mathcal{P} = \frac{1}{2}m\dot{y}^2 - mgy$$

$$\frac{\partial \mathcal{L}}{\partial \dot{y}} = \frac{\partial \mathcal{K}}{\partial \dot{y}}$$

$$\frac{\partial \mathcal{L}}{\partial y} = -\frac{\partial \mathcal{P}}{\partial y}$$

Example 1

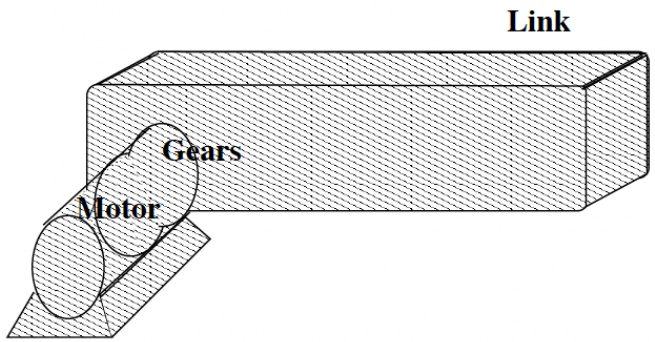
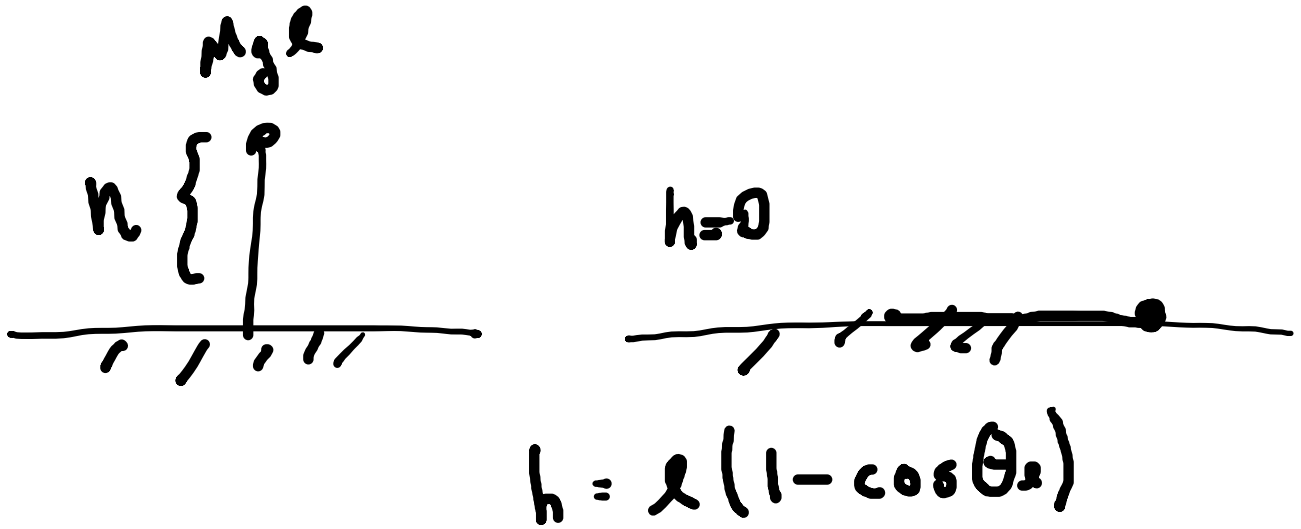


Fig. 6.2 Single-Link Robot.



Let θ_l and θ_m denote the angles of the link and motor shaft, respectively.
Then, $\theta_m = r\theta_l$, where r: 1 is the gear ratio. J_m, J_l are the rotational inertias.

$$
\begin{aligned}
K &= \frac{1}{2} J_m \dot{\theta}_m^2 + \frac{1}{2} J_\ell \dot{\theta}_\ell^2 \\
&= \frac{1}{2}(r^2 J_m + J_\ell)\dot{\theta}_\ell^2 \\
&\qquad \underbrace{\qquad\qquad}_{J = r^2 J_m + J_\ell}
\end{aligned}
\qquad\qquad
\begin{aligned}
P &= Mg\ell(1 - \cos\theta_\ell) \\
\mathcal{L} &= \frac{1}{2} J \dot{\theta}_\ell^2 - Mg\ell(1 - \cos\theta_\ell)
\end{aligned}
$$

Example

$$\mathcal{L} = \frac{1}{2} J\dot{\theta}_\ell^2 - Mg\ell(1-\cos\theta_\ell) \qquad \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = f$$

Substituting this expression into the Euler-Lagrange equations yields the equation of motion

$$J\ddot{\theta}_\ell + Mg\ell\sin\theta_\ell = \tau_\ell$$

→ Dynamics of the arm!

The generalized force τ_l represents those external forces and torques that are not derivable from a potential function

---

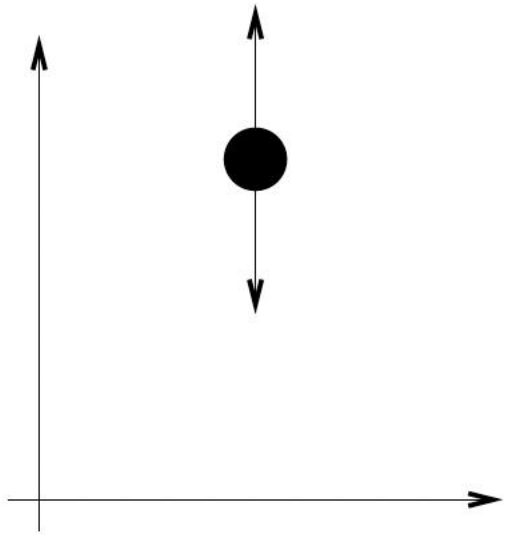
In general, for any system of the type considered, an application of the Euler-Lagrange equations leads to a system of n coupled, second order nonlinear ordinary differential equations of the form:

Euler-Lagrange Equations

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \quad i = 1, \ldots, n$$

Example 2

Let's code this:



$$m\ddot{y} \quad = \quad f - mg$$

Fig. 6.1 One Degree of Freedom System

Example

dynamics.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# System parameters
m = 1.0  # mass (kg)
g = 9.81  # gravitational acceleration (m/s^2)
f = 10  # external force (N) -m*g - 10*y[1]

# Define the differential equation
def system(t, y):
    # y[0] = position (y)
    # y[1] = velocity (v = dy/dt)
    # control: f = -m*g - 10*y[0] - 7*y[1]
    dydt = [y[1], (f - m * g) / m]  # velocity and acceleration
    return dydt
```

Example

```
17
18   # Initial conditions
19   y0 = [5, 8]  # initial position and initial velocity
20   t_span = (0, 10)  # time interval (from 0 to 10 seconds)
21   t_eval = np.linspace(t_span[0], t_span[1], 100)  # points for evaluation
22
23   # Solve the differential equation
24   sol = solve_ivp(system, t_span, y0, t_eval=t_eval)
25
26   # Extract results
27   t = sol.t        # time values
28   y = sol.y[0]     # position values
29   v = sol.y[1]     # velocity values
30
```

Example

```
31    # Plot the results
32    plt.figure(figsize=(10, 5))
33
34    # Position plot
35    plt.subplot(2, 1, 1)
36    plt.plot(t, y, label='Position')
37    plt.xlabel('Time (s)')
38    plt.ylabel('Position (m)')
39    plt.legend()
40    plt.grid()
41
42    # Velocity plot
43    plt.subplot(2, 1, 2)
44    plt.plot(t, v, label='Velocity', color='orange')
45    plt.xlabel('Time (s)')
46    plt.ylabel('Velocity (m/s)')
47    plt.legend()
48    plt.grid()
49
50    plt.tight_layout()
51    plt.show()
```

Exercise

Create a code for the system of Example 1