Robot Kinematics

Haijun Su
The Ohio State University

7th January 2026

Contents

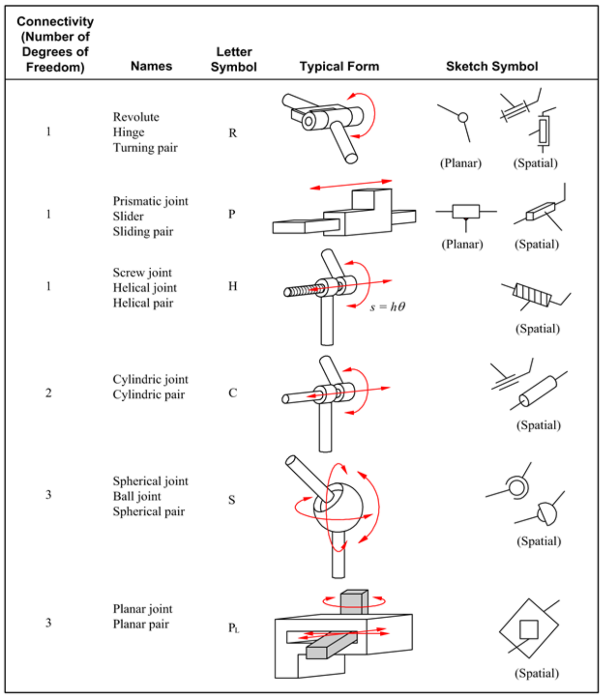
Chapter 1: Preface

This is an unofficial summary of the robot kinematics lectures on *Robot Kinematics* at The Ohio State University.

We aim to cover all the content from the exercises and supplement it with explanations from the book *Introduction to Robotics - Mechanics and Control (3rd ed.)* by John J. Craig. Much of the lecture notes are by the professors Philipp Wulff and Jan Hansen-Palmus from the Technical University of Munich GitHub repository . And the original content of their lecture notes is based on the Stanford lecture *Introduction to Robotics* by professor Oussama Khatib, which you can watch on YouTube. Many of the figures in this summary are direct screenshots of his slides. Note, that the notations are not consistent throughout this summary.

Chapter 2: Kinematics Fundamentals

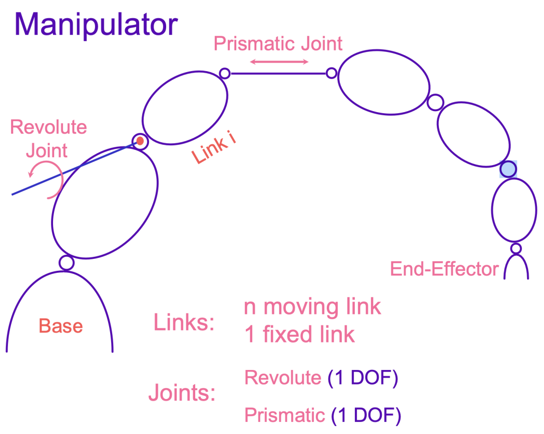
Joints usually have motion/position sensors, allowing to measure their relative position to neighboring links. The figure below shows some commonly used kinematic joints [1].



Connectivity (Number of Degrees of Freedom)	Names	Letter Symbol	Typical Form	Sketch Symbol
1	Revolute Hinge Turning pair	R		(Planar) (Spatial)
1	Prismatic joint Slider Sliding pair	P		(Planar) (Spatial)
1	Screw joint Helical joint Helical pair	H	$s = h\theta$	(Spatial)
2	Cylindric joint Cylindric pair	C		(Spatial)
3	Spherical joint Ball joint Spherical pair	S		(Spatial)
3	Planar joint Planar pair	P_L		(Spatial)

Figure 1: Commonly used kinematic joints [1].

The **end-effector** is located at the end of the chain of links that make up the manipulator. Different types of joints connect the links of a manipulator [1].



Chapter 3: Forward kinematics

3.1 Workspace

Kinematics is the science of motion that treats motion without regard to the forces which cause it. One studies position, velocity, acceleration, and all higher order derivatives of position variables.

The existence or nonexistence of a kinematic solution defines the **workspace** of a given manipulator [2]. If a solution doesn't exist, this means that the desired position/orientation lies outside of the manipulator's workspace. In other words, the workspace consists of all points that are reachable by the end-effector.

3.1.1 Configuration

The **configuration** of a moving object is a specification of the position of **every** point on the object [3].

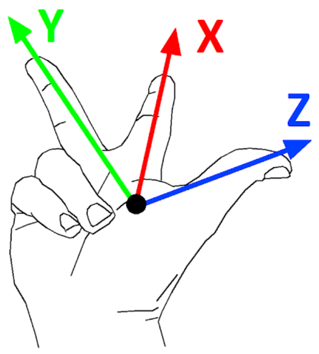
The **dimension of a config space** is the minimum number of parameters needed to specify the configuration of the object completely (also called the number of degrees of freedom of a moving object).

3.1.2 Degrees of freedom

The number of **degrees of freedom** that a manipulator possesses is the number of independent position variables that would have to be specified in order to locate all parts of the mechanism. E.g., industrial robotic manipulators often have as many d.o.f. as their number of joints, since each joint has one d.o.f. (and has 5 constraints).

3.1.3 Right-hand-rule

The signs of angles are determined by the direction of the fingers, when the thumb is pointing in the axis direction. Fingers also show the order of axis.



3.2 Spatial descriptions

We attach a coordinate system to a body and give a description of this coordinate system relative to the reference system. In Figure 2.2, system B has been attached to the body, and its description relative to A is given through its positions and orientation relative to A.

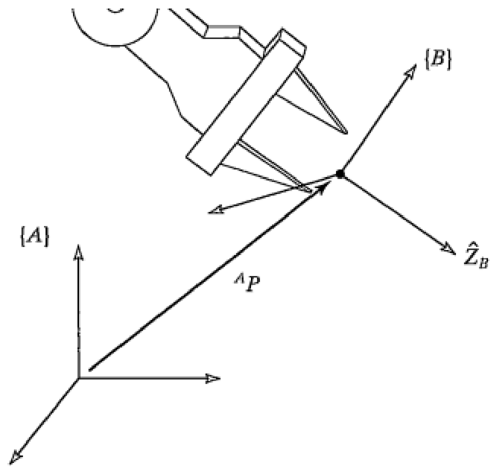


FIGURE 2.2: Locating an object in position and orientation.

3.2.1 Position

Description of a position: Since many coordinate systems will be used, one has to define to which system a vector refers, e.g.,

$$^A\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix},$$

where $^A\vec{p}$ is a vector referring to coordinate system A.

3.2.2 Orientation

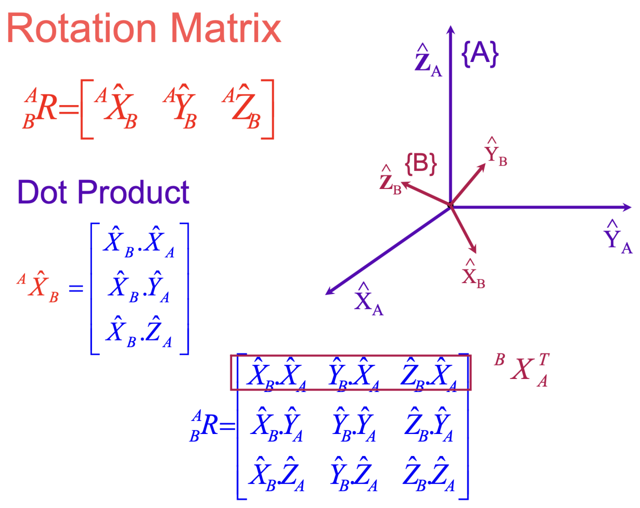
Description of an orientation: We stack three unit vectors (they specify the principal directions of the coord system) as columns, yielding the **rotation matrix:**

$$^A_BR = [^A\hat{X}_B \; ^A\hat{Y}_B \; ^A\hat{Z}_B] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \; ^B_AR^{-1} = \; ^B_AR^T$$

The rotation matrix has three constraints:

- $|^A\hat{X}_B| = |^A\hat{Y}_B| = |^A\hat{Z}_B| = 1$
- $^A\hat{X}_B \cdot \; ^A\hat{Y}_B = \; ^A\hat{X}_B \cdot \; ^A\hat{Z}_B = \; ^A\hat{X}_B \cdot \; ^A\hat{Z}_B = 0$
- $\det R = 1$

It describes the orientation of frame B relative to frame A, i.e. it is used as a *mapping* to **change the description of a vector from frame to frame**. Since we have unit magnitude and the vectors are orthogonal, the transposed matrix describes the orientation of system A written in B. These are orthonormal and length-preserving linear transformations. Also, rotation matrices preserve angles between vectors, i.e. $cos(\angle(\vec{p},\vec{q})) = cos(\angle(R\vec{p}, R\vec{q}))$. The projection of a vector $\hat{X}_B$ in coord system B into coord system A is derived from the dot product with the principal directions of the coord frame of A, hence the graphic.


Rotation Matrix
${}^A_BR = \left[{}^A\hat{X}_B \;\; {}^A\hat{Y}_B \;\; {}^A\hat{Z}_B \right]$
Dot Product
${}^A\hat{X}_B = \begin{bmatrix} \hat{X}_B.\hat{X}_A \\ \hat{X}_B.\hat{Y}_A \\ \hat{X}_B.\hat{Z}_A \end{bmatrix}$
${}^A_BR = \begin{bmatrix} \hat{X}_B.\hat{X}_A & \hat{Y}_B.\hat{X}_A & \hat{Z}_B.\hat{X}_A \\ \hat{X}_B.\hat{Y}_A & \hat{Y}_B.\hat{Y}_A & \hat{Z}_B.\hat{Y}_A \\ \hat{X}_B.\hat{Z}_A & \hat{Y}_B.\hat{Z}_A & \hat{Z}_B.\hat{Z}_A \end{bmatrix}$
${}^BX_A^T$
$\hat{Z}_A$
{A}
{B}
$\hat{Z}_B$
$\hat{Y}_B$
$\hat{Y}_A$
$\hat{X}_B$
$\hat{X}_A$

Besides mappings, another use case of rotation matrices is in the form of (rotational) *operators* to move points within the same frame.

Python script example based on SciPy and NumPy:

```python
import numpy as np
from scipy.spatial.transform import Rotation as R

# Create a random rotation
r = R.random()
R_matrix = r.as_matrix()
print("Rotation matrix:")
print(R_matrix)
print("Determinant:", np.linalg.det(R_matrix))
print("R^T R:", np.dot(R_matrix.T, R_matrix))
```

3.2.3 Special rotation matrices

The special rotation matrices for rotation about the axes are:

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, R_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, R_Z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```python
# Rotation about x by 45 degrees
r_X = R.from_rotvec([np.pi/4, 0, 0])
print("R_X(45 degrees):")
print(r_X.as_matrix())

# Similarly for y and z
r_Y = R.from_rotvec([0, np.pi/4, 0])
r_Z = R.from_rotvec([0, 0, np.pi/4])
```

3.2.4 Composition of rotations

A rigid body is subject to 2 or more consecutive rotations, defined by $[R_1], [R_2], \ldots$. The final rotation $[R]$ is defined by the product of these matrices. Depending on the reference coordinate system, there are two cases:

- Intrinsic: the 2nd rotation is about the axes of the body-fixed coordinate system, $[R] = [R_1][R_2]$
- Extrinsic: the 2nd rotation is about the global coordinate system, $[R] = [R_2][R_1]$

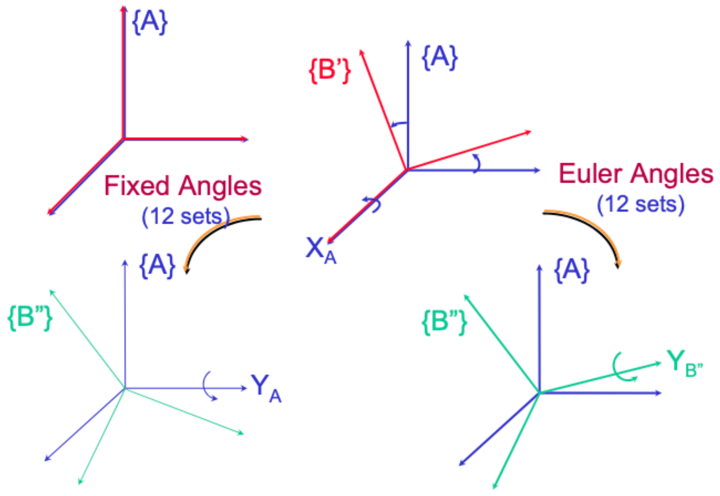
```
# Define two rotations
r1 = R.from_rotvec([np.pi/6, 0, 0]) # 30 degrees about x
r2 = R.from_rotvec([0, np.pi/4, 0]) # 45 degrees about y

# Intrinsic composition: [R] = [R1][R2]
r_intrinsic = r1 * r2
print("Intrinsic rotation matrix:")
print(r_intrinsic.as_matrix())

# Extrinsic composition: [R] = [R2][R1]
r_extrinsic = r2 * r1
print("Extrinsic rotation matrix:")
print(r_extrinsic.as_matrix())
```

3.2.5 Euler angles

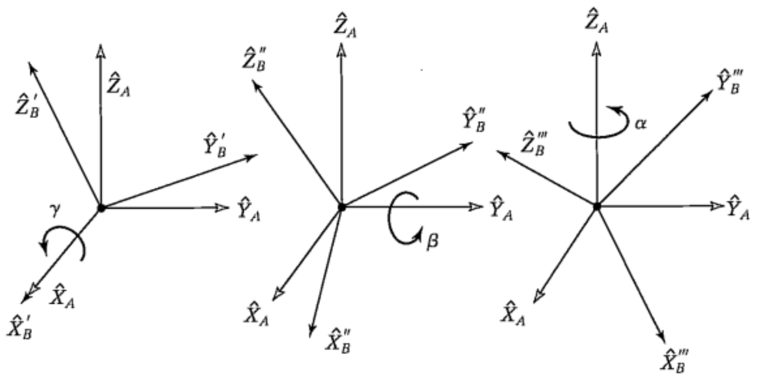
There is an issue when expressing orientations with a rotation matrix: When attempting to follow a trajectory in space by interpolating a current orientation from a start until a final orientation, the intermediary orientations are going to violate the constraints of the rotation matrix. Another possible description of a frame B uses a *three-angle-representation*. It works as follows: Frame B coincides with a known frame A. Rotate B first

- about $\hat{X}_A$ by an angle γ, then about
- $\hat{Y}_A$ by an angle β, and finally about
- $\hat{Z}_A$ by α.

If, each of the three rotations takes place about an axis in the fixed reference frame A, we call these angles *X-Y-Z fixed angles*. An alternative representation uses angles that are relative to the previously changed coordinate frame, so-called **Euler angles**.



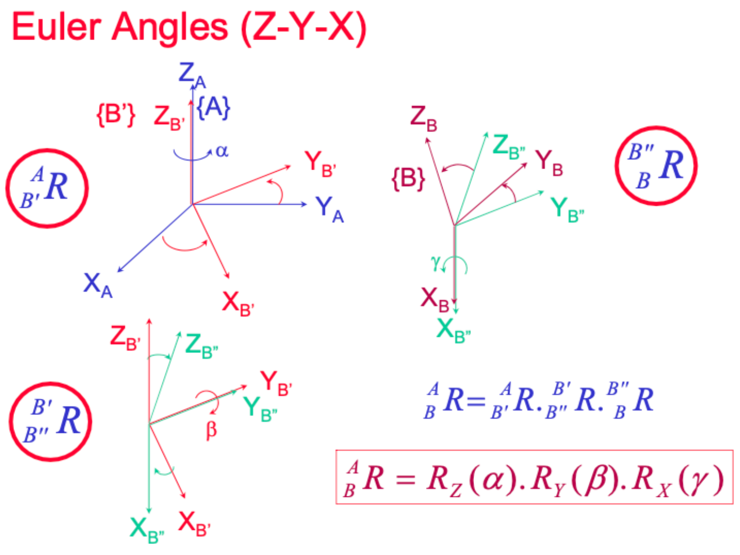
The next figure shows three subsequent rotations of a frame around the fixed axes $\hat{X}_A$, $\hat{Y}_A$ and $\hat{Z}_A$.



The combined rotation matrix is ${}^A_BR = {}^A_{B'}R \cdot {}^{B'}_{B''}R \cdot {}^{B''}_{B}R$, which is cumbersome to compute. Instead, we define:

$$\begin{aligned}
{}^A_BR_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\
&= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\
&= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}
\end{aligned}$$

When rotating a frame with Euler angles in the order of Z-Y-X, they produce the same combined rotation as X-Y-Z fixed angles, i.e. $R_{Z'Y'X'}(\alpha,\beta,\gamma) = R_{XYZ}(\gamma,\beta,\alpha)$ for the same values of (α,β,γ). This next graphic shows the rotation using Euler angles:



Inverse problem: Given ${}^A_BR$ find the X-Y-Z fixed angles (α, β, γ). Let

$$ {}^A_BR_{XYZ}(\gamma, \beta, \alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} $$

If $\cos\beta \neq 0$,

$$ \begin{aligned} &\beta_1 = \arcsin(-r_{31}), \beta_2 = \pi - \arcsin(-r_{31}) \\ &\alpha_i = \text{atan2}\left(r_{21}/\cos\beta_i, r_{11}/\cos\beta_i\right) \\ &\gamma_i = \text{atan2}\left(r_{32}/\cos\beta_i, r_{33}/\cos\beta_i\right) \end{aligned} $$

See the definition of atan2(a, b) in the Appendix.

If $\cos\beta = 0$, this is a singularity (gimbal lock), we have infinite solutions of α and γ. There are two possible solution sets:

$$ \begin{aligned} &\beta_1 = \pi/2, \quad \gamma_1 - \alpha_1 = \text{atan2}(r_{12}, r_{13}) \\ &\beta_2 = -\pi/2, \quad \gamma_2 + \alpha_2 = \text{atan2}(-r_{12}, -r_{13}) \end{aligned} $$

There are 12 different Euler angle conventions, which arise from the different possible sequences of axis rotations (choosing which axes to rotate about and in what order). The examples given below are for intrinsic Euler angles, where each subsequent rotation is performed relative to the body's current orientation. There are 12 sets of intrinsic Euler angle conventions:

- **Proper Euler angles:** These are Euler angle sequences where the first and third rotations are performed about the same axis. The middle rotation is about a different axis.
  - z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y
- **Tait–Bryan angles:** These are Euler angle sequences where all three rotations are performed about different axes, providing a complete set of three distinct axes.
  - x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z

The corresponding extrinsic Euler angle sets are the reverse sequences. For instance, the extrinsic version of the intrinsic X-Y-Z is Z-Y-X. See 6.1 in the Appendices for all 12 sets of intrinsic Euler angles and their corresponding rotation matrices [4].

Example: x-y-z (intrinsic): Let α, β, γ be the Euler angles of three consecutive rotations in the sequence of x_0, y_1 and z_2, where the subscripts indicate the coorindate system the rotation is about.

$$ [R(\alpha, \beta, \gamma)]_{xyz} = R_X(\alpha)R_Y(\beta)R_Z(\gamma) $$

$$ = \begin{pmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \sin\alpha\sin\beta\cos\gamma + \sin\gamma\cos\alpha & -\sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & -\sin\alpha\cos\beta \\ \sin\alpha\sin\gamma - \sin\beta\cos\alpha\cos\gamma & \sin\alpha\cos\gamma + \sin\beta\sin\gamma\cos\alpha & \cos\alpha\cos\beta \end{pmatrix} $$

Inverse Kinematics: For the intrinsic x-y-z sequence, the angles can be recovered from $[R]$ as follows:

$$\beta_1 = \arcsin(r_{13}), \beta_2 = \pi - \arcsin(r_{13})$$
$$\alpha_i = \operatorname{atan2}(\sin\alpha, \cos\alpha) = \operatorname{atan2}\left(-\frac{r_{23}}{\cos\beta_i}, \frac{r_{33}}{\cos\beta_i}\right)$$
$$\gamma_i = \operatorname{atan2}(\sin\gamma, \cos\gamma) = \operatorname{atan2}\left(-\frac{r_{12}}{\cos\beta_i}, \frac{r_{11}}{\cos\beta_i}\right)$$

This gives two sets of solutions: $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$. When $\cos\beta = 0$ (gimbal lock), γ and α are not independent. Use the similar formulas as above to compute β and $\gamma - \alpha$ or $\gamma + \alpha$ as shown in the previous example.

```python
import numpy as np
from scipy.spatial.transform import Rotation as R

# Given Euler angles (alpha, beta, gamma) in radians
original_angles = np.array([0.1, 0.2, 0.3]) # example values

# Generate rotation matrix from Euler angles
r_original = R.from_euler('xyz', original_angles)
R_matrix = r_original.as_matrix()
print("Rotation matrix from original angles:")
print(R_matrix)

# Now, use inverse kinematics to extract Euler angles
extracted_angles = r_original.as_euler('xyz', degrees=False)
print("Extracted Euler angles (alpha, beta, gamma):")
print(extracted_angles)

# Check if they match (within numerical precision)
difference = np.abs(original_angles - extracted_angles)
print("Difference between original and extracted:")
print(difference)
print("Are they close?", np.allclose(original_angles, extracted_angles))
```

3.2.6 Rotation axis and angle

Every rotation has an axis, which is the set of points that remain invariant (fixed) under the rotation transformation. The geometric properties of rotation axes are fundamental in understanding spatial rotations [2].

Definition: The points that remain fixed during a rotation $[R]$ form its **rotation axis**. To find these points, we consider the transformation equation

$$X = [R]X$$

This shows that a fixed point X is the solution to

$$([I] - [R])X = 0$$

For there to be solutions other than $X = 0$, the determinant of the coefficient matrix must be zero: $\det([I] - [R]) = 0$. This condition is satisfied for all spatial rotation matrices, meaning that rotation matrices always have $\lambda = 1$ as an eigenvalue. If K is a nonzero solution, then every point $P = tK$ on the line through the origin and K is also a solution. This line of points is the **rotation axis**.

Cayley's formula: To obtain an explicit equation for the rotation axis, we use Cayley's formula. Consider the points x and X that represent the initial and final positions obtained from the rotation $X = [R]x$. Using the fact that $|x| = |X|$ (rotation preserves distances), we can show that

$$X - x = b \times (X + x)$$

where $\times$ denotes the vector cross product, and b is **Rodrigues's vector**. This vector can be computed from the rotation matrix using

$$[B] = ([R] - [I])([R] + [I])^{-1}$$

where $[B]$ is the skew-symmetric matrix associated with $b = (b_x, b_y, b_z)^T$:

$$[B] = \begin{bmatrix} 0 & -b_z & b_y \\ b_z & 0 & -b_x \\ -b_y & b_x & 0 \end{bmatrix}$$

The skew-symmetric matrix $[B]$ has the property that for any vector y,

$$[B]y = b \times y$$

Cayley's formula for the rotation matrix in terms of $[B]$ is:

$$[R] = ([I] - [B])^{-1}([I] + [B])$$

To find the rotation axis explicitly, we solve $([I] - [R])X = 0$. Substituting Cayley's formula:

$$\Big([I] - ([I] - [B])^{-1}([I] + [B])\Big) X = 0$$

which simplifies to

$$[B]X = 0$$

Since $[B]X = b \times X$, it is clear that $X = b$ is a solution. Thus, **Rodrigues's vector b defines the rotation axis direction**.

We denote by $K = (k_x, k_y, k_z)^T$ the unit vector in the direction of Rodrigues's vector b, which identifies the rotation axis. The magnitude of Rodrigues's vector is related to the rotation angle θ by

$$b = \tan\left(\frac{\theta}{2}\right) K$$

Therefore, the rotation axis can be extracted from a rotation matrix by:

1. Compute the skew-symmetric matrix $[B] = ([R] - [I])([R] + [I])^{-1}$
2. Extract the vector $b = (b_x, b_y, b_z)^T$ from $[B]$
3. Normalize b to obtain the unit rotation axis: $K = \frac{b}{|b|}$
4. Compute the rotation angle: $\theta = 2\arctan(|b|)$

The rotation matrix can also be expressed explicitly in terms of its rotation axis K and angle θ using:

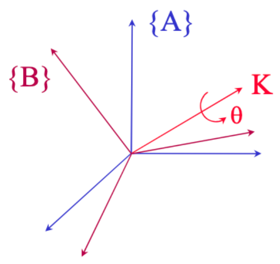
$$[R(\theta, K)] = [I] + \sin\theta[K] + (1 - \cos\theta)[K]^2$$

where $[K]$ is the skew-symmetric matrix associated with the unit vector K.

```
import numpy as np
from scipy.spatial.transform import Rotation as R

# Example: Extract rotation axis and angle from a rotation matrix
# Create a rotation: 45 degrees around axis [1, 1, 1]
axis = np.array([1, 1, 1])
axis = axis / np.linalg.norm(axis) # normalize
angle = np.pi / 4 # 45 degrees

r = R.from_rotvec(angle * axis)
R_matrix = r.as_matrix()

print("Rotation matrix:")
print(R_matrix)

# Method 1: Using scipy to extract axis-angle
rotvec = r.as_rotvec()
extracted_angle = np.linalg.norm(rotvec)
extracted_axis = rotvec / extracted_angle if extracted_angle != 0 else rotvec

print("\nExtracted rotation axis K:", extracted_axis)
print("Extracted rotation angle (radians):", extracted_angle)
print("Extracted rotation angle (degrees):", np.degrees(extracted_angle))

# Method 2: Using Cayley's formula (manual computation)
I = np.eye(3)
# Compute [B] = ([R] - [I])([R] + [I])^{-1}
try:
    B = (R_matrix - I) @ np.linalg.inv(R_matrix + I)
    # Extract b from skew-symmetric matrix
    b = np.array([B[2, 1], B[0, 2], B[1, 0]])
    b_magnitude = np.linalg.norm(b)
    K_cayley = b / b_magnitude
    theta_cayley = 2 * np.arctan(b_magnitude)

    print("\nUsing Cayley's formula:")
    print("Rodrigues vector b:", b)
    print("Rotation axis K:", K_cayley)
    print("Rotation angle (radians):", theta_cayley)
    print("Rotation angle (degrees):", np.degrees(theta_cayley))
except:
    print("\nCayley's formula computation failed (singular case)")
```

3.2.7 Unit quaternions

Another representation of orientation is by means of four values called the **Quaternions**. It can be shown that every rotation can be expressed as one rotation of θ around a single axis $K = [k_x\ k_y\ k_z]^T$. This is also known as the *equivalent angle-axis representation*.



The quaternions are defined by

$$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4) = (k_x \sin\frac{\theta}{2}, k_y \sin\frac{\theta}{2}, k_z \sin\frac{\theta}{2}, \cos\frac{\theta}{2})$$

and

$$\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \epsilon_4^2 = 1$$

This 4×1 vector is known as a unit quaternion, the orientation it describes could be visualized as a point on a unit hypersphere in four-dimensional space. It turns out that the orientation representation through Euler parameters does not have a singularity.

3.3 Spatial transformations

Problem: We know the definition of a vector with respect to some frame B and we would like to express it with respect to another frame A, where the origins of the two frames are coincident. We can compute this, if we know a description of the orientation of B relative to A. Then,

$$^AP = {}^A_BR\, {}^BP$$

computes what a **vector that is expressed in coordinate system B "looks like" if observed from system A**.

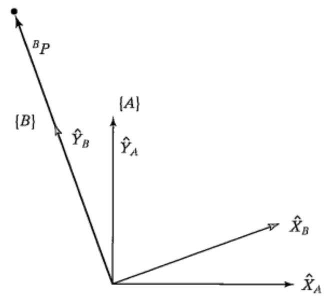


FIGURE 2.6: $\{B\}$ rotated 30 degrees about $\hat{Z}$.

$${}^A_BR = \begin{bmatrix} 0.866 & -0.500 & 0.000 \\ 0.500 & 0.866 & 0.000 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}.$$

Given

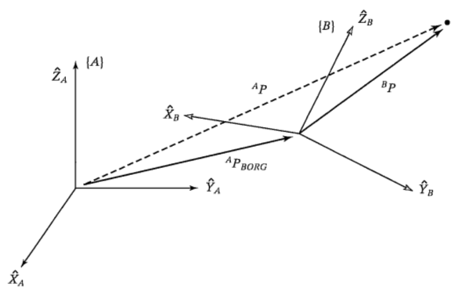
$${}^BP = \begin{bmatrix} 0.0 \\ 2.0 \\ 0.0 \end{bmatrix},$$

we calculate ^AP as

$${}^AP = {}^A_BR\, {}^BP = \begin{bmatrix} -1.000 \\ 1.732 \\ 0.000 \end{bmatrix}.$$

Now, the **general case**, where the systems don't have the same origin, but B is shifted by P_{BORG} from the origin (or $^AP_{BORG}$ when expressed in A):

$$^AP = {}^A_BR\, {}^BP + {}^AP_{BORG}$$



Computing a position of a point connected by multiple subsequent links of a manipulator with individual coordinate frames, involves the multiplication of sums given by the general transform equation. This is because *rotation* and *translation* are both needed to propagate from one frame to another. To simplify this calculation, a **homogeneous transform** combines rotation and translation into a matrix ${}^A_BT$, such that: ${}^AP = {}^A_BT\ {}^BP$. Therefore, the homogeneous transform is a *general operator*.

$$ {}^AP = {}^A_BR\,{}^BP + {}^AP_{BORG} $$

$$\left[\begin{array}{c} {}^AP \\ \hline 1 \end{array}\right] = \left[\begin{array}{ccc|c} & {}^A_BR & & {}^AP_{Borg} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{c} {}^BP \\ \hline 1 \end{array}\right]$$

$$\underline{\underline{\underset{(4x1)}{{}^AP} = \underset{(4x4)}{{}^A_BT}\ \underset{(4x1)}{{}^BP}}}$$

Due to the generality of the homogeneous transform, it can be interpreted in different ways: 1.) as a description of a frame $\{B\} = \{{}^A_BR\ \ {}^AP_{BORG}\}$; 2.) as a mapping of a point in frame B to frame A; 3.) as an operator to rotate and translate a point in the same frame. This is an example for the second case:

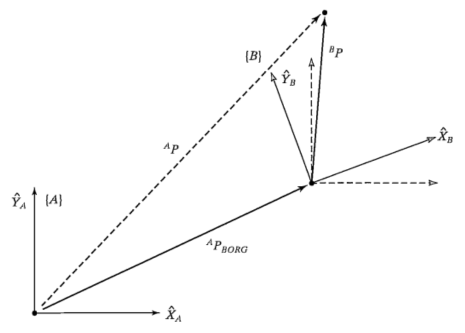


FIGURE 2.8: Frame $\{B\}$ rotated and translated.

Figure 2.8 shows a frame $\{B\}$, which is rotated relative to frame $\{A\}$ about $\hat{Z}$ by 30 degrees, translated 10 units in $\hat{X}_A$, and translated 5 units in $\hat{Y}_A$. Find ${}^AP$, where ${}^BP = [3.0\ 7.0\ 0.0]^T$.

The definition of frame $\{B\}$ is

$$ {}^A_BT = \begin{bmatrix} 0.866 & -0.500 & 0.000 & 10.0 \\ 0.500 & 0.866 & 0.000 & 5.0 \\ 0.000 & 0.000 & 1.000 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.21)$$

Given

$$ {}^BP = \begin{bmatrix} 3.0 \\ 7.0 \\ 0.0 \end{bmatrix}, \quad (2.22)$$

we use the definition of $\{B\}$ just given as a transformation:

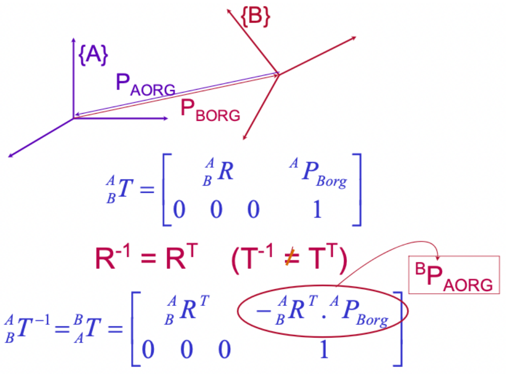
$$ {}^AP = {}^A_BT\ {}^BP = \begin{bmatrix} 9.098 \\ 12.562 \\ 0.000 \end{bmatrix}. \quad (2.23)$$

```python
import numpy as np
from scipy.spatial.transform import Rotation as R

# Homogeneous transformation matrix from frame B to frame A
A_T_B = np.array([
    [0.866, -0.500, 0.000, 10.0],
    [0.500, 0.866, 0.000, 5.0],
    [0.000, 0.000, 1.000, 0.0],
    [0.000, 0.000, 0.000, 1.0]
])

# Point P in frame B (homogeneous coordinates)
B_P = np.array([3.0, 7.0, 0.0, 1.0])

# Transform point into frame A
A_P = A_T_B @ B_P

# Extract just the 3D coordinates
print("\nCoordinates of P in frame A:")
print(A_P[:3])
```

3.3.1 Inverse of the homogeneous transform

The inverse of a homogeneous transform ${}^A_BT = [{}^A_BR \ \ {}^AP_{BORG}]$ is given by:

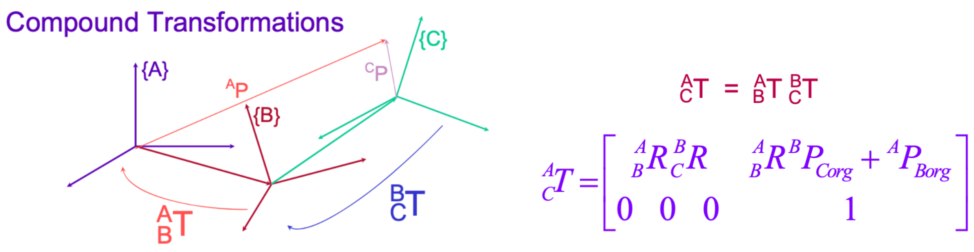
$$ {}^B_AT = \begin{bmatrix} {}^A_BR^T & -{}^A_BR^T \ {}^AP_{BORG} \\ 0 & 1 \end{bmatrix} $$



{B}
{A}
P_AORG
P_BORG
${}^A_BT = \begin{bmatrix} & {}^A_BR & & {}^AP_{Borg} \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$R^{-1} = R^T \quad (T^{-1} \neq T^T)$
${}^BP_{AORG}$
${}^A_BT^{-1} = {}^B_AT = \begin{bmatrix} & {}^A_BR^T & & -{}^A_BR^T . {}^AP_{Borg} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3.3.2 Compound transformations

When multiple transformations are applied in sequence, the overall transformation can be obtained by multiplying the individual homogeneous transformation matrices. This is known as **compound transformation**.



Compound Transformations
{A}
{B}
{C}
${}^AP$
${}^CP$
${}^A_BT$
${}^B_CT$
${}^A_CT = {}^A_BT \ {}^B_CT$
${}^A_CT = \begin{bmatrix} {}^A_BR \, {}^B_CR & {}^A_BR \, {}^BP_{Corg} + {}^AP_{Borg} \\ 0 \ \ 0 \ \ 0 & 1 \end{bmatrix}$

Transform equation: ${}^A_BT \ {}^B_CT \ {}^C_DT \ {}^D_AT = I$.

```
import numpy as np
from scipy.spatial.transform import Rotation as R

# Define two homogeneous transformations

# Transformation T1: rotation by 30 degrees around z-axis, translation [10, 5, 0]
r1 = R.from_euler('z', np.pi/6)
T1 = np.eye(4)
T1[:3, :3] = r1.as_matrix()
T1[:3, 3] = [10, 5, 0]

# Transformation T2: rotation by 45 degrees around x-axis, translation [2, 3, 1]
r2 = R.from_euler('x', np.pi/4)
T2 = np.eye(4)
T2[:3, :3] = r2.as_matrix()
T2[:3, 3] = [2, 3, 1]
```

```
# Compound transformation: T_compound = T1 @ T2
T_compound = T1 @ T2
print("Compound transformation matrix:")
print(T_compound)

# Inverse of the compound transformation
T_inv = np.linalg.inv(T_compound)
print("Inverse transformation matrix:")
print(T_inv)

# Example: Apply compound transformation to a point
P = np.array([1.0, 2.0, 3.0, 1.0]) # Point in homogeneous coordinates
P_transformed = T_compound @ P
print("Point after compound transformation:", P_transformed[:3])

# Apply inverse to get back
P_back = T_inv @ P_transformed
print("Point after applying inverse:", P_back[:3])
```

3.3.3 Screw axis and displacement

A fundamental result in spatial kinematics, known as **Chasles' theorem**, states that every spatial displacement can be represented as a rotation about an axis combined with a translation along that same axis. This axis is called the **screw axis**, and the combined motion is called a **screw displacement** [2].

Definition: While spatial displacements have no fixed points in general, there exists a line, called the **screw axis**, that remains invariant (fixed in direction and position) during the displacement. A spatial displacement consists of:

- A rotation by angle θ about the screw axis
- A translation by distance t along the screw axis

Consider a spatial displacement defined by the homogeneous transform $[T] = [R, d]$, where $[R]$ is the 3×3 rotation matrix and d is the translation vector. If a point C has the same coordinates before and after the displacement, it must satisfy:

$$C = [R]C + d$$

which simplifies to:

$$([I] - [R])C = d$$

Since all rotation matrices have $\lambda = 1$ as an eigenvalue (meaning $\det([I] - [R]) = 0$), this equation generally has no solution, confirming that spatial displacements have no fixed points. However, we can find a fixed line by decomposing the translation.

Computing the screw axis: The screw axis can be computed by decomposing the translation vector d into components parallel and perpendicular to the rotation axis K of $[R]$:

$$d = d_{\perp} + tK$$

where:

- $K = (k_x, k_y, k_z)^T$ is the unit vector defining the rotation axis (computed using methods from the previous section)
- $t = d \cdot K$ is the translation distance along the screw axis

- $d_\perp = d - tK$ is the component perpendicular to K

The displacement can now be decomposed as:

$$[T] = [I, tK][R, d_\perp] = [R, d_\perp + tK]$$

A point C on the screw axis can be found using Rodrigues's vector $b = \tan(\theta/2)K$ from the rotation matrix $[R]$:

$$C = \frac{b \times (d_\perp - b \times d_\perp)}{2(b \cdot b)}$$

The screw axis is then the line:

$$S : P(s) = C + sK$$

where s is a parameter along the line.

Screw parameters: The complete screw displacement is characterized by four parameters:

1. **Screw axis direction**: $K = (k_x, k_y, k_z)^T$ (unit vector)
2. **Point on screw axis**: C (determines axis location)
3. **Rotation angle**: θ (rotation about the axis)
4. **Translation**: t (slide along the axis)

The screw displacement can be expressed as:

$$[T(\theta, t, K, C)] = [R(\theta, K), ([I] - [R])C + tK]$$

This shows that any point X transforms according to:

$$X - C = [R(\theta, K)](x - C) + tK$$

which explicitly shows the rotation about point C followed by translation along K.

```python
import numpy as np
from scipy.spatial.transform import Rotation as R

# Example: Compute screw axis from a spatial displacement
# Define a rotation: 60 degrees around axis [1, 1, 1]
axis = np.array([1, 1, 1])
K = axis / np.linalg.norm(axis) # normalize to get unit vector
theta = np.deg2rad(60)

r = R.from_rotvec(theta * K)
R_matrix = r.as_matrix()

# Define a translation vector
d = np.array([3.0, 4.0, 5.0])

# Decompose translation into parallel and perpendicular components
t = np.dot(d, K) # translation along axis
d_perp = d - t * K # perpendicular component

print("Rotation axis K:", K)
print("Rotation angle (degrees):", np.degrees(theta))
print("Translation along axis t:", t)
print("Perpendicular translation d_perp:", d_perp)

# Find a point C on the screw axis using Rodrigues's vector
I = np.eye(3)
try:
    B = (R_matrix - I) @ np.linalg.inv(R_matrix + I)
    b = np.array([B[2, 1], B[0, 2], B[1, 0]])

    # Compute point on screw axis
    if np.linalg.norm(b) > 1e-10:
        C = np.cross(b, (d_perp - np.cross(b, d_perp))) / (2 * np.dot(b, b))
    else:
        # Pure translation case
        C = np.zeros(3)

    print("\nPoint on screw axis C:", C)

    # Verify: reconstruct the homogeneous transform
    T_reconstructed = np.eye(4)
    T_reconstructed[:3, :3] = R_matrix
    T_reconstructed[:3, 3] = (I - R_matrix) @ C + t * K

    print("\nReconstructed translation vector:", T_reconstructed[:3, 3])
    print("Original translation vector:", d)
    print("Match:", np.allclose(T_reconstructed[:3, 3], d))

except:
    print("Computation failed (singular case or pure translation)")

# Visualize the screw parameters
print("\n=== Screw Displacement Summary ===")
print(f"Screw axis direction K: [{K[0]:.4f}, {K[1]:.4f}, {K[2]:.4f}]")
print(f"Rotation angle theta: {np.degrees(theta):.2f} degrees")
print(f"Translation along axis t: {t:.4f}")
if 'C' in locals():
    print(f"Point on axis C: [{C[0]:.4f}, {C[1]:.4f}, {C[2]:.4f}]")
```

Special screw displacement matrices: For screw displacements about the principal coordinate axes (passing through the origin), the 4×4 homogeneous transformation matrices take simplified forms. A screw displacement consists of a rotation by angle θ about an axis and a translation by distance t along that same axis.

Screw displacement about the x-axis:

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

Screw displacement about the y-axis:

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

Screw displacement about the z-axis:

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

These matrices combine the rotation matrices $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ with translations along their respective axes. Note that the translation component appears in the position corresponding to the axis of rotation (x-translation in first row, y-translation in second row, z-translation in third row).

3.4 Forward kinematics

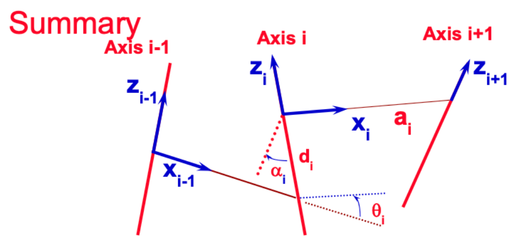
Forward kinematics solves the static geometrical problem of computing the position and the orientation of the end-effector of the manipulator. Specifically, given a **set of joint angles**, the forward kinematic problem is to compute the position and orientation of the tool frame relative to the base frame.

3.4.1 Denavit-Hartenberg Convention

The homogeneous transform between consecutive frames of links is represented via only 4 D-H-parameters (x-rot, x-trans, z-trans, z-rot). In general, 6 parameters are needed to represent an arbitrary rigid body transformation! Thus, a restriction of it is, that it **cannot represent a rotation around the y-axis and that the y- and z-position are coupled** [4].

Procedure for deriving the D-H-parameters:

1. Identify the joint axes and imagine (or draw) infinite lines along them. For steps 2 through 5 below, consider two of these neighboring lines (at axes i and $i+1$).
2. Identify the common perpendicular between them, or point of intersection. At the point of intersection, or at the point where the common perpendicular meets the ith axis, assign the link-frame origin.
3. Assign the $\hat{Z}_i$ axis pointing along the ith joint axis.
4. Assign the $\hat{X}_i$ axis pointing along the common perpendicular, or, if the axes intersect, assign $\hat{X}_i$ to be normal to the plane containing the two axes.
5. Assign the $\hat{Y}_i$ axis to complete a right-hand coordinate system.
6. Assign {0} to match {1} when the first joint variable is zero. For $\{N\}$, choose an origin location and $\hat{X}_N$ direction freely, but generally so as to cause as many linkage parameters as possible to become zero.



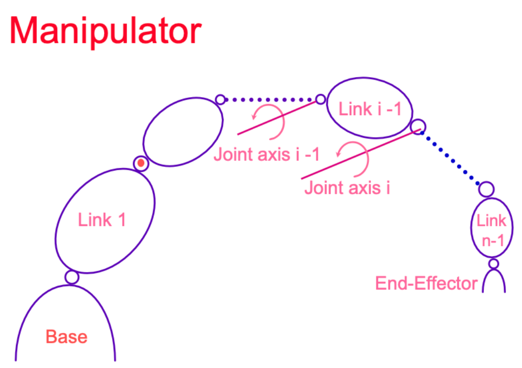
a_i = *the distance from* $\hat{Z}_i$ *to* $\hat{Z}_{i+1}$ *measured along* $\hat{X}_i$;

α_i = *the angle from* $\hat{Z}_i$ *to* $\hat{Z}_{i+1}$ *measured about* $\hat{X}_i$;

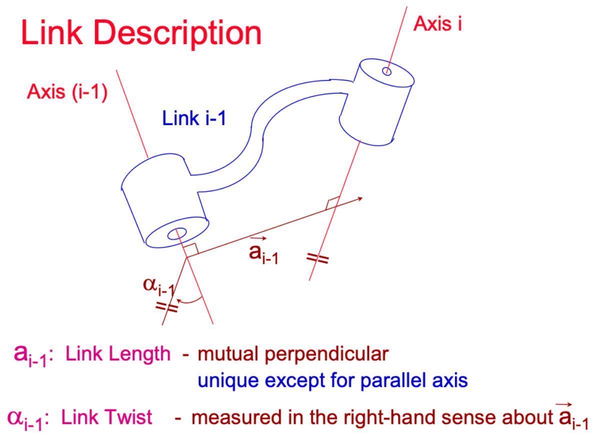
d_i = *the distance from* $\hat{X}_{i-1}$ *to* $\hat{X}_i$ *measured along* $\hat{Z}_i$; and

θ_i = *the angle from* $\hat{X}_{i-1}$ *to* $\hat{X}_i$ *measured about* $\hat{Z}_i$.

Among the four D-H-parameters are three fixed link parameters and one joint variable, which is θ_i for a revolute joint or d_i for a prismatic joint. The D-H-parameters are plugged into homogenous transformation matrices, which represent the single transformations between the frames of subsequent links. Then, we can propagate through all homogeneous transformations to calculate the position and orientation of the end-effector. Hereafter, the above described procedure is detailed further:

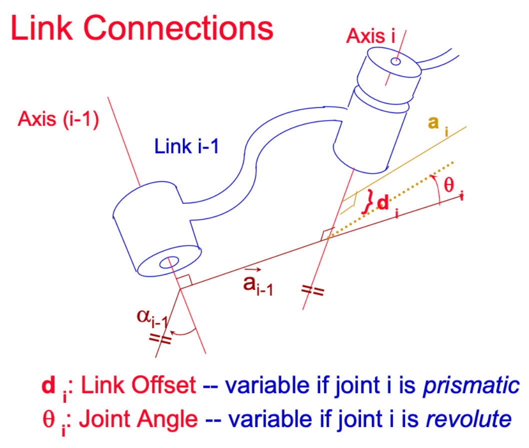


(a) Identify the joint axes; consider axes i and $i-1$. By convention, a joint axis points **in the direction of the rotation/ movement** for revolute/ prismatic joints.

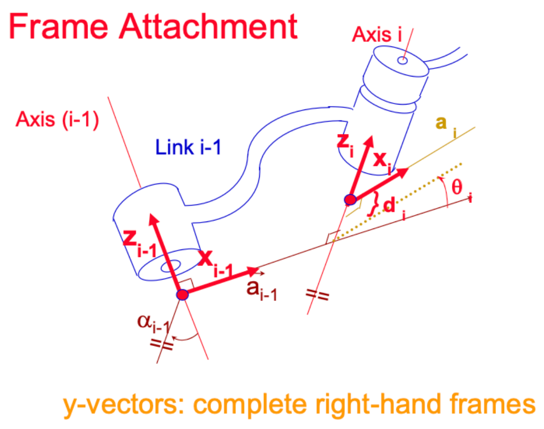

Link Description
Axis i
Axis (i-1)
Link i-1
$\vec{a}_{i-1}$
α_{i-1}
a_{i-1}: Link Length - mutual perpendicular
unique except for parallel axis
α_{i-1}: Link Twist - measured in the right-hand sense about $\vec{a}_{i-1}$

(b) Identify the common perpendicular. If the axes intersect, the common perpendicular is a normal through the plane they span and the direction of α_i is determined by the direction of this normal. a_i **and** α_i **describe the** i**th link**. ("We rotate axis $i-1$ around the normal about α so that it coincides with axis i. This aligns the Z-axes.")

Figure 2: D-H parameters assignment

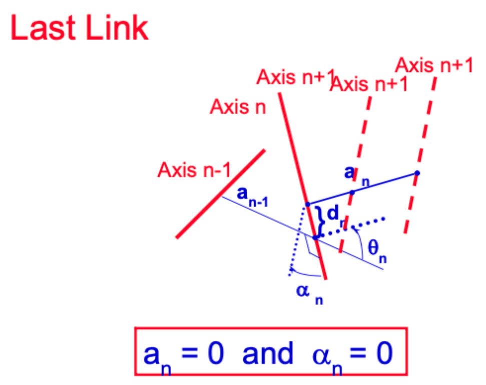

Link Connections
Axis i
Axis (i-1)
Link i-1
a_i
d_i
θ_i
$\vec{a}_{i-1}$
α_{i-1}
d_i: Link Offset -- variable if joint i is *prismatic*
θ_i: Joint Angle -- variable if joint i is *revolute*

(c) d_i **and** θ_i **describe the** $i-1$**th link connection**. If the axes of link $i-1$ and link i are parallel, the origin of a coordinate frame $\{i-1\}$ should be attached, such that $d_i = 0$. ("θ_i aligns the X-axes by rotation around Z_i.")

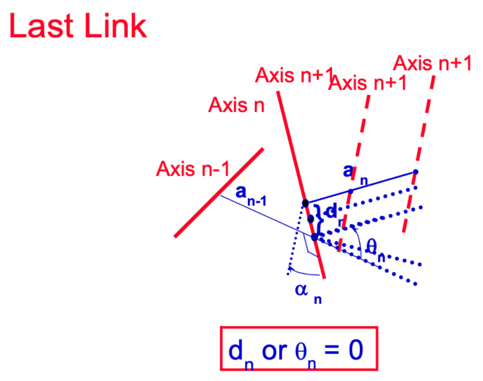


(d) Attach ith frame, such that $\hat{Z}_i$ is in the direction of the ith joint axis and $\hat{X}_i$ points along the common perpendicular (if $\hat{Z}_{i-1}$ and $\hat{Z}_i$ intersect, choose $\hat{X}_{i-1}$, such that $\alpha_{i-1} > 0$). $\hat{Y}_i$ completes the right-hand frame.

Figure 2: (Continued)

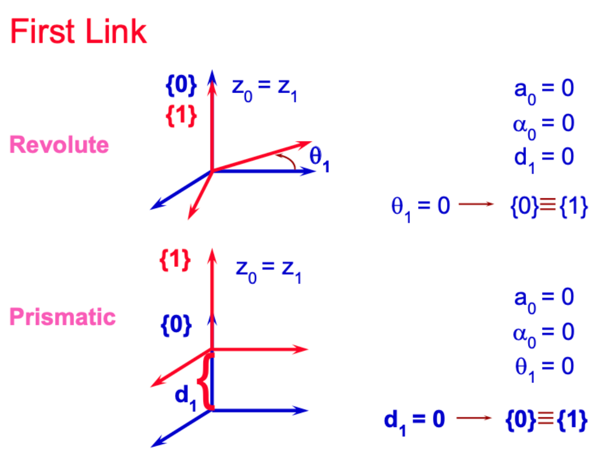


(e) a_i and α_i depend on joint axes i and $i+1$. Thus, select axes 0 and $n+1$, such that $a_0 = a_n = 0$ and $\alpha_0 = \alpha_n = 0$ (by making axis 0 coincident with axis 1 and axis $n+1$ coincident with axis n). This simplifies the forward kinematics.

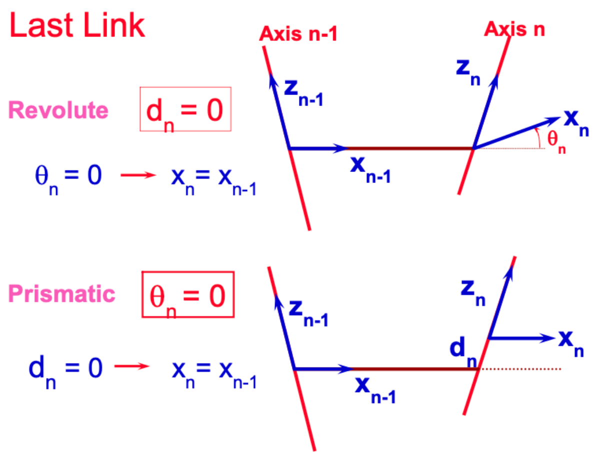


(f) θ_i and d_i depend on joint axes i and $i-1$. Again, select axes 0 and $n+1$, such that depending on the joint type a_0 or $\theta_0 = 0$ and a_n or $\theta_n = 0$ (by coinciding axes and moving the intersection point that becomes the origin of the frame so that $d = 0$ or orienting the axis so that $\theta = 0$).

Figure 2: (Continued)



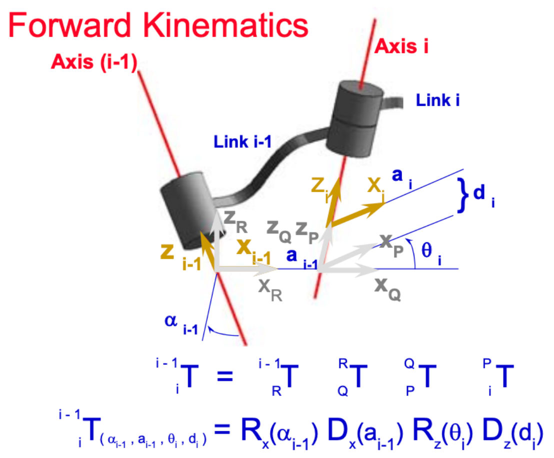
(g) $\{0\}$ is assigned, such that is equals $\{1\}$ when the first joint variable is 0.



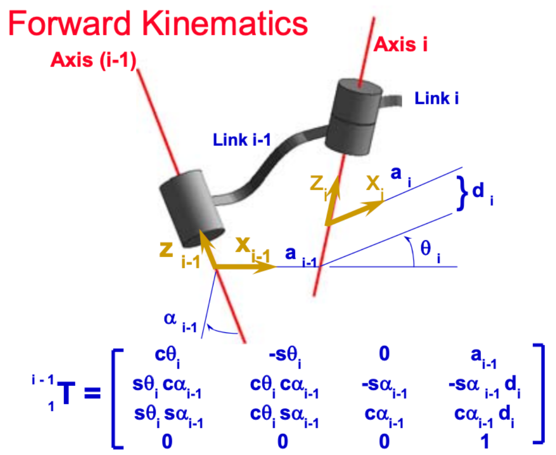
(h) Assign $\{N\}$, such that the most parameters are 0.

Figure 2: (Continued)

Then, the forward kinematics of a single link using the D-H-parameters derive as follows:


Forward Kinematics
Axis (i-1)
Axis i
Link i
Link i-1
Z_i
X_i
a_i
d_i
Z_R
Z_Q
Z_P
X_P
θ_i
Z_{i-1}
X_{i-1}
a_{i-1}
X_R
X_Q
α_{i-1}
$^{i-1}_{i}T = {}^{i-1}_{R}T \; {}^{R}_{Q}T \; {}^{Q}_{P}T \; {}^{P}_{i}T$
$^{i-1}_{i}T_{(\alpha_{i-1}, a_{i-1}, \theta_i, d_i)} = R_x(\alpha_{i-1}) \, D_x(a_{i-1}) \, R_z(\theta_i) \, D_z(d_i)$

(a) To obtain $^{i-1}_{i}T$: Translate along Z_i about d_i ($\rightarrow \{P\}$), then rotate around Z_i about θ_i ($\rightarrow \{Q\}$), then translate along X_{i-1} about a_{i-1} ($\rightarrow \{R\}$) and finally rotate around X_i about α_i ($\rightarrow \{i-1\}$). This results in the homogeneous transform for a single link from $\{i\}$ to $\{i-1\}$ and also works in the other direction to compute the inverse homogeneous transform.


Forward Kinematics
Axis (i-1)
Axis i
Link i
Link i-1
Z_i
X_i
a_i
d_i
θ_i
Z_{i-1}
X_{i-1}
a_{i-1}
α_{i-1}
$^{i-1}_{1}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(b) The **homogeneous transformation from link i to link $i-1$** (the formula in the slide should say i instead of 1).

Forward Kinematics: $^{0}_{N}T = {}^{0}_{1}T \;\; {}^{1}_{2}T \; \dots \; {}^{N-1}_{N}T$

(c) The homogeneous transformation which transforms points from the frame of link N to the frame of link 0.

Note, that the homogeneous transform $^{i-1}_{i}T$ cannot express arbitrary rigid body transformations, since no rotation about $\hat{Y}$ is possible. A good, practical explanation of how to place the coordinate frames at each link, derive the D-H-parameters and compute the homogeneous transformations and forward kinematics is given in this video (based on an example).

3.4.2 D-H Parameter Table for Robot Manipulators

The structural parameters of a serial chain robot manipulator can be described by a table of Denavit-Hartenberg (D-H) parameters. Each row in the table corresponds to a joint/link and specifies four parameters: a_{i-1} (link length), α_{i-1} (link twist), d_i (link offset), and θ_i (joint angle). These parameters are essential for systematically modeling the kinematics of robot arms.

Example: UR5e Robot Arm The official D-H parameters for the UR5e robot arm are as follows:[1].The table lists the link lengths (a_{i-1}), link twists (α_{i-1}), link offsets (d_i), and joint angles (θ_i) for each joint of the UR5e. The figure shows the kinematic diagram and the assignment of DH frames for the UR series robots.

The standard Denavit-Hartenberg (DH) parameters use a_{i-1}, α_{i-1}, d_i, and θ_i, while the modified DH (DHM) convention uses a_i, α_i, d_i, and θ_i, with positive link lengths preferred and potential offsets added to θ for negative lengths. RoboDK, a powerful software for robot simulation, offline programming, and

[1]Source: Universal Robots: Denavit-Hartenberg Parameters

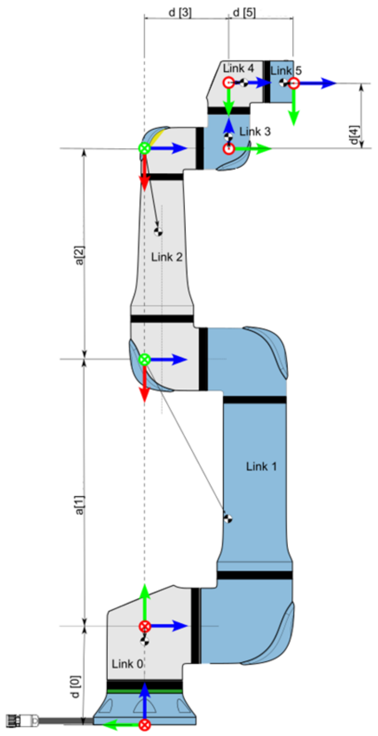


Table 1: Denavit-Hartenberg Parameters for UR5e

Joint i	a_{i-1} [m]	α_{i-1} [deg]	d_i [m]	θ_i [deg]
1	0	0	0.1625	θ_1
2	-0.425	0	0	θ_2
3	-0.3922	0	0	θ_3
4	0	90	0.1333	θ_4
5	0	-90	0.0997	θ_5
6	0	0	0.0996	θ_6

Table 2: Modified Denavit-Hartenberg Parameters (DHM) for UR5e (RoboDK)

Joint i	a_i [mm]	α_i [rad]	d_i [mm]	θ_i [rad]
1	0	0	162.5	0
2	0	$\frac{\pi}{2}$	0	π
3	425	0	0	0
4	392.25	0	133.3	0
5	0	$-\frac{\pi}{2}$	99.7	0
6	0	$\frac{\pi}{2}$	99.6	π

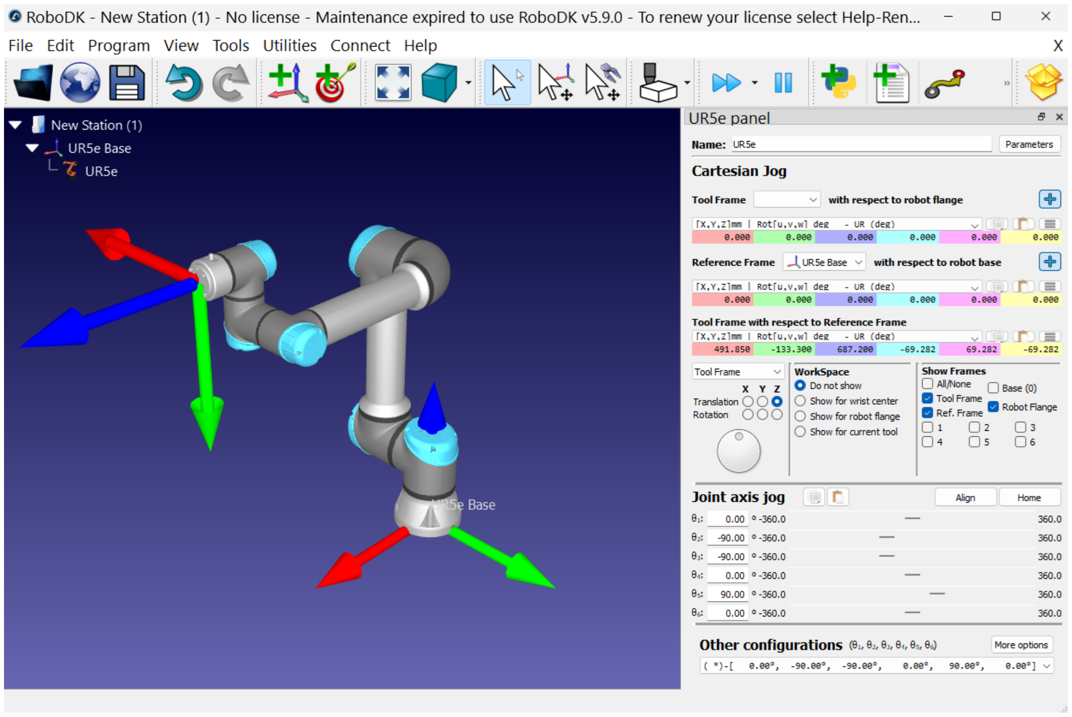
Figure 3: UR5e kinematic diagram (left) and DH parameter tables (right).

analysis, employs the DHM convention for its robot models. You can view and interact with the UR5e robot model in RoboDK's online library at RoboDK Robot Library.

```
import numpy as np
from scipy.spatial.transform import Rotation as R

# DHM parameters for UR5e (in mm and rad)
a = [0, 0, 425, 392.25, 0, 0]
alpha = [0, np.pi/2, 0, 0, -np.pi/2, np.pi/2]
d = [162.5, 0, 0, 133.3, 99.7, 99.6]
theta_offsets = [0, np.pi, 0, 0, 0, np.pi]

# Given joint angles in degrees
joint_angles_deg = [0, -90, -90, 0, 90, 0]
joint_angles_rad = np.deg2rad(joint_angles_deg)

# Add offsets to joint angles
theta = joint_angles_rad + theta_offsets

# Function to compute DH transformation matrix
def dh_transform(a, alpha, d, theta):
    T = np.eye(4)
    # Trans_x(a)
    T[0, 3] = a
    # Rot_x(alpha)
    T[:3, :3] = R.from_rotvec([alpha, 0, 0]).as_matrix()
    # Trans_z(d)
    T_z = np.eye(4)
    T_z[2, 3] = d
    T = T @ T_z
    # Rot_z(theta)
    T_theta = np.eye(4)
    T_theta[:3, :3] = R.from_rotvec([0, 0, theta]).as_matrix()
    T = T @ T_theta
    return T

# Compute total transformation
T_total = np.eye(4)
for i in range(6):
    T_i = dh_transform(a[i], alpha[i], d[i], theta[i])
    T_total = T_total @ T_i

print("End-effector transformation matrix:")
print(T_total)
```

The expected output is:

```
[[ 0. 0. 1. 491.85]
 [-1. 0. 0. -133.30]
 [ 0. -1. 0. 687.20]
 [ 0. 0. 0. 1.]]
```


RoboDK - New Station (1) - No license - Maintenance expired to use RoboDK v5.9.0 - To renew your license select Help-Ren...
File Edit Program View Tools Utilities Connect Help
New Station (1)
UR5e Base
UR5e
UR5e Base

UR5e panel

Name: UR5e

Cartesian Jog

Tool Frame with respect to robot flange

[X,Y,Z]mm | Rot[u,v,w] deg - UR (deg)

0.000	0.000	0.000	0.000	0.000	0.000

Reference Frame UR5e Base with respect to robot base

[X,Y,Z]mm | Rot[u,v,w] deg - UR (deg)

0.000	0.000	0.000	0.000	0.000	0.000

Tool Frame with respect to Reference Frame

[X,Y,Z]mm | Rot[u,v,w] deg - UR (deg)

491.850	-133.300	687.200	-69.282	69.282	-69.282

Tool Frame

	X	Y	Z
Translation	○	○	●
Rotation	○	○	○

WorkSpace
- [✓ Do not show]
- Show for wrist center
- Show for robot flange
- Show for current tool

Show Frames
- ☐ All/None
- ☐ Base (0)
- ✓ Tool Frame
- ✓ Robot Flange
- ✓ Ref. Frame
- ☐ 1 ☐ 2 ☐ 3
- ☐ 4 ☐ 5 ☐ 6

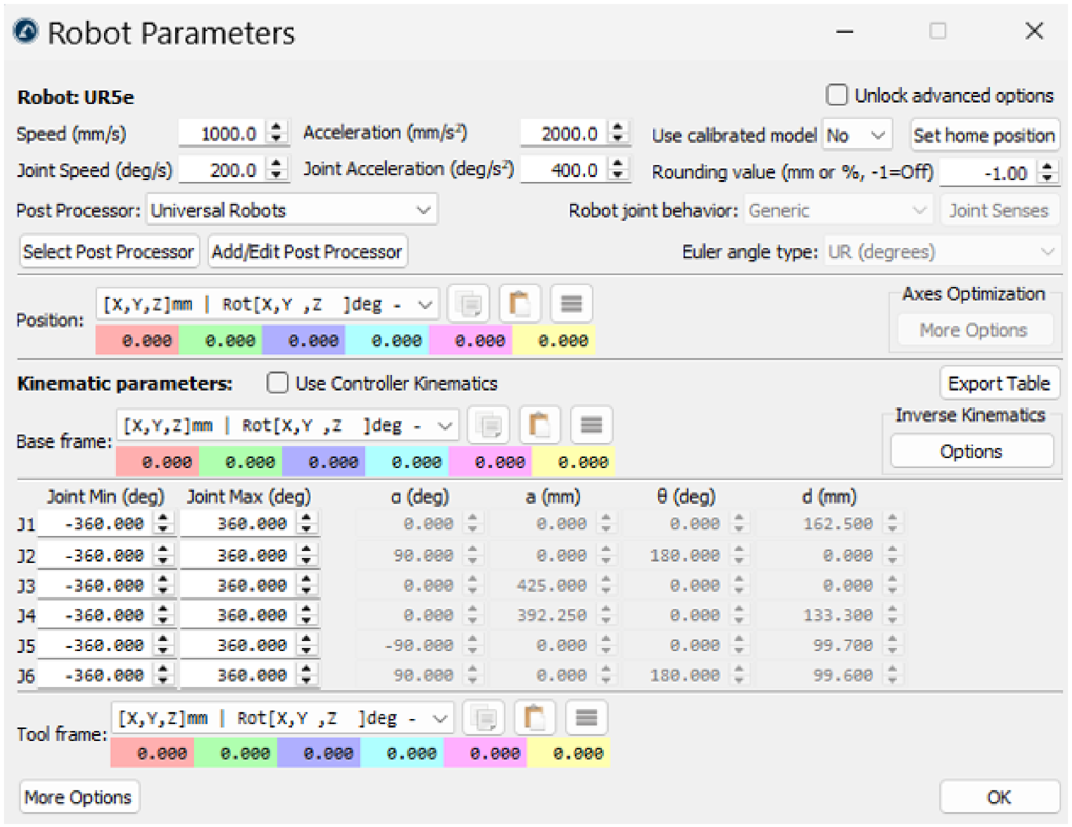
Joint axis jog Align Home

θ_1:	0.00	° -360.0	360.0
θ_2:	-90.00	° -360.0	360.0
θ_3:	-90.00	° -360.0	360.0
θ_4:	0.00	° -360.0	360.0
θ_5:	90.00	° -360.0	360.0
θ_6:	0.00	° -360.0	360.0

Other configurations ($\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$) More options

(*)-[0.00°, -90.00°, -90.00°, 0.00°, 90.00°, 0.00°]

(a) RoboDK software GUI.



Robot Parameters

Robot: UR5e ☐ Unlock advanced options

Speed (mm/s)	1000.0	Acceleration (mm/s²)	2000.0	Use calibrated model	No
Joint Speed (deg/s)	200.0	Joint Acceleration (deg/s²)	400.0	Rounding value (mm or %, -1=Off)	-1.00

Post Processor: Universal Robots

Robot joint behavior: Generic

Euler angle type: UR (degrees)

Position: [X,Y,Z]mm | Rot[X,Y ,Z]deg -

0.000	0.000	0.000	0.000	0.000	0.000

Kinematic parameters: ☐ Use Controller Kinematics

Base frame: [X,Y,Z]mm | Rot[X,Y ,Z]deg -

0.000	0.000	0.000	0.000	0.000	0.000

	Joint Min (deg)	Joint Max (deg)	α (deg)	a (mm)	θ (deg)	d (mm)
J1	-360.000	360.000	0.000	0.000	0.000	162.500
J2	-360.000	360.000	90.000	0.000	180.000	0.000
J3	-360.000	360.000	0.000	425.000	0.000	0.000
J4	-360.000	360.000	0.000	392.250	0.000	133.300
J5	-360.000	360.000	-90.000	0.000	0.000	99.700
J6	-360.000	360.000	90.000	0.000	180.000	99.600

Tool frame: [X,Y,Z]mm | Rot[X,Y ,Z]deg -

0.000	0.000	0.000	0.000	0.000	0.000

(b) UR5e robot with DHM parameters in RoboDK.

Figure 4: RoboDK software interface (top) and UR5e robot model with DHM parameters (bottom).

Chapter 4: Inverse Kinematics

Recap forward kinematics: Given a joint configuration, find the pose of some part of the robot.
Inverse kinematics: Given a pose, figure out the joint configurations.

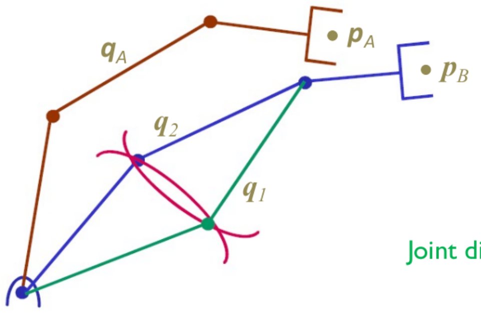
Input data for the problem is of the form:

$$T = \begin{bmatrix} R & t \\ 0\,0\,0 & 1 \end{bmatrix}$$

This is a nonlinear problem, thus it's not sure if there is a (unique/multiple/infinite/outside-of-workspace/none) solution. Inverse kinematics are not explicitly discussed the Stanford lecture and there is not particular exercise on it (albeit it is included in the lecture).

4.1 Multiplicity of Solutions

If there are multiple solutions (configurations), choose the closest one:



Initial configuration for $\boldsymbol{p}_A$: $\boldsymbol{q}_A$

New configurations for $\boldsymbol{p}_B$: $\boldsymbol{q}_1$, $\boldsymbol{q}_2$

Is it better to choose $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$? $\boldsymbol{q}_2$

Joint distance between configurations:

$$\left.\begin{aligned} d_1 &= \| \mathbf{q}_1 - \mathbf{q}_A \| \\ d_2 &= \| \mathbf{q}_2 - \mathbf{q}_A \| \end{aligned}\right\} \text{Shortest distance is preferred}$$

In general, if there are N possible configurations for $\mathbf{p}_B$, choose:

$$\mathbf{q}_b = \arg\min_{\mathbf{q}} \| \mathbf{q} - \mathbf{q}_A \| \qquad \text{for} \qquad \mathbf{q} \in \{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_N\}$$

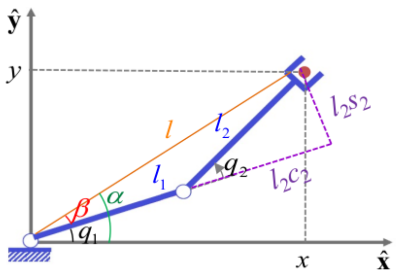
4.2 Analytic solutions for the inverse kinematics

4.2.1 Geometric solution

- only when robot has 3 or less dofs
- not a generic solution

Example:

Find the inverse kinematics for the position of the R-R robot using a geometric approach



Solution

For q_2:

- Using the law of cosines:

$$l^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(180 - q_2)$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 \cos(q_2)$$

$$c_2 = \frac{x^2 + y^2 - (l_1^2 + l_2^2)}{2l_1 l_2}$$

- Using a trigonometric identity:

$$s_2^2 + c_2^2 = 1$$

$$s_2 = \pm\sqrt{1 - c_2^2}$$

$$q_2 = \operatorname{atan2}(s_2, c_2)$$

For q_1 (using the geometry of the figure)

$$q_1 = \alpha - \beta$$

$$\alpha = \operatorname{atan2}(y, x)$$

$$\beta = \operatorname{atan2}(l_2 s_2, l_1 + l_2 c_2)$$

Inverse kinematics:

$$q_1 = \operatorname{atan2}(y, x) - \operatorname{atan2}(l_2 s_2, l_1 + l_2 c_2)$$

$$q_2 = \operatorname{atan2}(s_2, c_2)$$

4.2.2 Algebraic solution

Solution using algebraic (and polynomial) equations.
Given: Formula to find the kinematic equations of an arm easily, given link params. E.g.:

$$ {}^B_W T = {}^0_3 T = \begin{bmatrix} c_{123} & -s_{123} & 0.0 & l_1 c_1 + l_2 c_{12} \\ s_{123} & c_{123} & 0.0 & l_1 s_1 + l_2 s_{12} \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This describes the wrist (manipulator) frame relative to the base frame.
We also know what is necessary to describe such a position/orientation:

$$ {}^B_W T = \begin{bmatrix} c_\phi & -s_\phi & 0.0 & x \\ s_\phi & c_\phi & 0.0 & y \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_i$$

So just an x-y positon and an angle ϕ. By equating both, we get a set of nonlinear equations, which have to be solved for l_1, l_2, θ, using magical algebra:

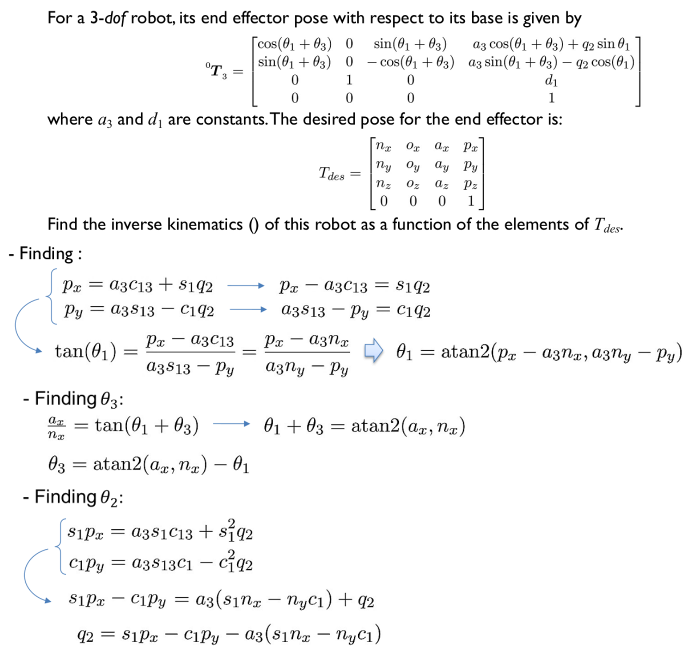
$$c_\phi = c_{123},$$

$$s_\phi = s_{123},$$

$$x = l_1 c_1 + l_2 c_{12}$$

$$y = l_1 s_1 + l_2 s_{12}$$

4.2.3 Example



For a 3-*dof* robot, its end effector pose with respect to its base is given by

$$ {}^{0}\boldsymbol{T}_3 = \begin{bmatrix} \cos(\theta_1+\theta_3) & 0 & \sin(\theta_1+\theta_3) & a_3\cos(\theta_1+\theta_3)+q_2\sin\theta_1 \\ \sin(\theta_1+\theta_3) & 0 & -\cos(\theta_1+\theta_3) & a_3\sin(\theta_1+\theta_3)-q_2\cos(\theta_1) \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where a_3 and d_1 are constants. The desired pose for the end effector is:

$$ T_{des} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Find the inverse kinematics () of this robot as a function of the elements of T_{des}.

- Finding :

$$ \begin{cases} p_x = a_3c_{13} + s_1q_2 \xrightarrow{\quad} p_x - a_3c_{13} = s_1q_2 \\ p_y = a_3s_{13} - c_1q_2 \xrightarrow{\quad} a_3s_{13} - p_y = c_1q_2 \end{cases} $$

$$ \tan(\theta_1) = \frac{p_x - a_3c_{13}}{a_3s_{13} - p_y} = \frac{p_x - a_3n_x}{a_3n_y - p_y} \Rightarrow \theta_1 = \operatorname{atan2}(p_x - a_3n_x, a_3n_y - p_y) $$

- Finding θ_3:

$$ \frac{a_x}{n_x} = \tan(\theta_1+\theta_3) \xrightarrow{\quad} \theta_1+\theta_3 = \operatorname{atan2}(a_x, n_x) $$

$$ \theta_3 = \operatorname{atan2}(a_x, n_x) - \theta_1 $$

- Finding θ_2:

$$ \begin{cases} s_1p_x = a_3s_1c_{13} + s_1^2q_2 \\ c_1p_y = a_3s_{13}c_1 - c_1^2q_2 \end{cases} $$

$$ s_1p_x - c_1p_y = a_3(s_1n_x - n_yc_1) + q_2 $$

$$ q_2 = s_1p_x - c_1p_y - a_3(s_1n_x - n_yc_1) $$

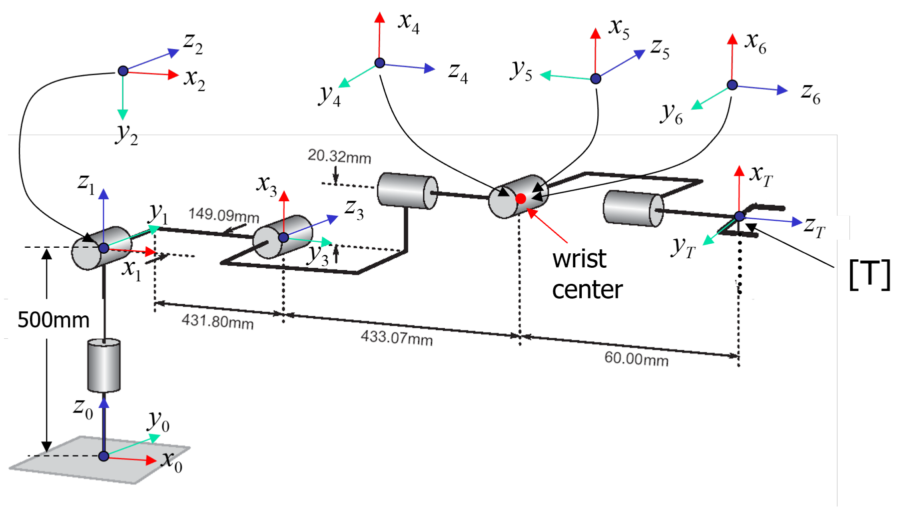
4.3 PUMA 560 Inverse Kinematics

4.3.1 DH Parameters

The PUMA 560 is a 6-DOF manipulator. Its Denavit-Hartenberg parameters are given in the following table:

Link	α_i (deg)	a_i (mm)	d_i (mm)	θ_i (deg)
1	0	0	0	θ_1
2	-90	0	0	θ_2
3	0	431.8	149.09	θ_3
4	-90	20.3	433.07	θ_4
5	90	0	0	θ_5
6	-90	0	0	θ_6

Table 3: DH Parameters for PUMA 560



4.3.2 Forward Kinematics

The forward kinematics for the PUMA 560 involves the following key steps:

1. Define the Denavit-Hartenberg (DH) parameters for each joint-link pair.
2. Compute the homogeneous transformation matrix for each link using the DH parameters. The transformation matrix for each link is:

$$[{}^{i-1}_{i}T] = [X(a_i,\alpha_i)][Z(d_i,\theta_i)] = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i\cos\alpha_i & \cos\theta_i\cos\alpha_i & -\sin\alpha_i & -d_i\sin\alpha_i \\ \sin\theta_i\sin\alpha_i & \cos\theta_i\sin\alpha_i & \cos\alpha_i & d_i\cos\alpha_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Multiply the transformation matrices sequentially to obtain the end-effector pose:

$$[{}^0_6T] = [{}^0_1T]\cdot[{}^1_2T]\cdot[{}^2_3T]\cdot[{}^3_4T]\cdot[{}^4_5T]\cdot[{}^5_6T]$$

.

4. Multiply The Base Frame and The Tool Frame:

$$T = G \times [{}^0_6T] \times H$$

where G is base transform matrix and H is the transform from the wrist center to the end-effector (tool) frame.

4.3.3 Inverse Kinematics Solution

The inverse kinematics for the PUMA 560 is solved analytically using the following steps:

1. **Remove Base/Tool Offsets**: Compute the effective transformation by removing the base and tool transforms from the target pose:

$$[{}^0_6T] = G^{-1}\cdot T\cdot H^{-1}$$

2. **Extract Wrist Center**: The wrist center position is extracted from the 4th columns of the transformation matrix:
$$\mathbf{p}_{wc} = [^0_6T][1:3,4] = (p_x, p_y, p_z)$$

3. **Solve for (θ_1)**: Use a trigonometric equation to solve for possible values of θ_1 (θ_1^+, θ_1^-):
$$-\sin\theta_1 \cdot p_x + \cos\theta_1 \cdot p_y = d_3$$
which is in the form of $A\cos(x) + B\sin(x) = C$ and can be solved analytically using the formula given in the appendix.

4. **Solve for (θ_3)**: Use a distance equation and trigonometric solver to find possible θ_3 values (θ_3^+, θ_3^-):
$$2a_2(a_3\cos\theta_3 - d_4\sin\theta_3) = (p_x^2 + p_y^2 + p_z^2) - (a_2^2 + a_3^2 + d_3^2 + d_4^2)$$
rewritten as: $A\cos\theta_3 + B\sin\theta_3 = C$, with
$$A = 2a_2a_3, \quad B = -2a_2d_4, \quad C = (p_x^2 + p_y^2 + p_z^2) - (a_2^2 + a_3^2 + d_3^2 + d_4^2)$$
The solutions can be found using the trigonometric equation solver provided in the appendix.

5. **Solve for (θ_2)**: For each (θ_1, θ_3) pair, solve the following linear equations for c_2 and s_2 to obtain $\theta_2^{1,2,3,4}$:
$$c_2(c_1p_x + s_1p_y) - s_2p_z = a_2 + a_3\cos\theta_3 - d_4\sin\theta_3$$
$$-s_2(c_1p_x + s_1p_y) - c_2p_z = a_3\sin\theta_3 + d_4\cos\theta_3$$
And then solve for θ_2 by
$$\theta_2 = \text{atan2}(s_2, c_2)$$
where θ_2^1 for (θ_1^+, θ_3^+), θ_2^2 for (θ_1^+, θ_3^-), θ_2^3 for (θ_1^-, θ_3^+), θ_2^4 for (θ_1^-, θ_3^-).

6. **Compute Wrist Rotation**: For each of the 4 sets (θ_1, θ_2^k, θ_3) where $k = 1, 2, 3, 4$, calculate the wrist rotation matrix and solve for the last three joint angles (θ_4, θ_5, θ_6) using spherical wrist formulas (2 solutions for each set):
$$[^0_3T] = [^0_1T] \cdot [^1_2T] \cdot [^2_3T] \quad R_0^3 = [^0_3T][1:3,1:3]$$
$$R_0^6 = [^0_6T][1:3,1:3]$$
$$R_3^6 = Inv(R_0^3) \cdot R_0^6$$
For the spherical wrist:
   - $\theta_5^\pm = \pm\arccos(R_3^6[3,3])$
   - $\theta_4^\pm = \text{atan2}(\frac{R_3^6[2,3]}{-\sin\theta_5}, \frac{R_3^6[1,3]}{-\sin\theta_5})$
   - $\theta_6^\pm = \text{atan2}(\frac{R_3^6[3,2]}{-\sin\theta_5}, \frac{R_3^6[3,1]}{\sin\theta_5})$

In summary, the PUMA 560 inverse kinematics yields up to 8 solutions: θ_1^+, θ_1^-, θ_3^+, θ_3^-, leading to θ_2^1, θ_2^2, θ_2^3, θ_2^4, and for each of these 4 sets, two solutions for the wrist angles ($\theta_4^\pm$, $\theta_5^\pm$, $\theta_6^\pm$).

The Python implementation is available in the [GitHub repository:] `https://github.com/haijunsu-osu/ik_puma560_notes`. You can also explore and run the code interactively in Google Colab: [Open in Colab]`https://colab.research.google.com/github/haijunsu-osu/ik_puma560_notes/blob/main/PUMA560Kinema` `FollowNotes_Answer.ipynb`.

Chapter 5: Motion Planning

Consider a start position P_S and end position P_E of a rigid body. To move from P_S to P_E, a trajectory must be generated, hence the term “motion planning” or “trajectory generation”. Such a trajectory is defined by the path it takes to get between positions (the Interpolated Motion) and the mathematical function describing the trajectory between positions (the Trajectory Planning).

5.1 Interpolated Motion

Any motion the robotic arm makes use of interpolation to navigate between positions P_S and P_E. Such interpolation can be done in several ways. Two common methods of interpolation are:

- **Joint motion moveJ():** Interpolation using 6DOF representation
- **Linear motion moveL():** Interpolation using equivalent angle axis representation

5.1.1 Joint motion

Using the joint values θ_S and θ_E a joint motion, using each joint independently of one another, can be made. The idea is to have each joint of a robotic arm follow their seperate trajectories between two points.

$$\text{moveJ}(t) = \begin{bmatrix} \theta_1(t) & \dots & \theta_n(t) \end{bmatrix}^T$$

Using such a joint motion to go from P_S to P_E requires that these positions can be represented using the joints of the robotic arm. This can be done by using Inverse Kinematics on said positions converting them into joint values.

$$\theta_S = \begin{bmatrix} \theta_{1,S} & \dots & \theta_{n,S} \end{bmatrix}^T = \text{IK}\left(P_S\right) \quad \theta_E = \begin{bmatrix} \theta_{1,E} & \dots & \theta_{n,E} \end{bmatrix}^T = \text{IK}\left(P_E\right)$$

5.1.2 Linear motion

Provided the position P_S and P_E in transform matrix format ${}^B_S T$ and ${}^B_E T$, it becomes possible to make a single transform matrix able to represent a linear motion between the two points ${}^S_E T = {}^B_S T^{-1} \cdot {}^B_E T$. And in cases where only joint values are provided, Forward Kinematics can be used to get said positions.

$$P_S = \text{FK}\left(\begin{bmatrix} \theta_{1,S} & \dots & \theta_{n,S} \end{bmatrix}^T\right) \quad P_E = \text{FK}\left(\begin{bmatrix} \theta_{1,E} & \dots & \theta_{n,E} \end{bmatrix}^T\right)$$

Using ${}^S_E T$, a linear motion between P_S and P_E becomes plausible using an angle-axis representation defined as:

$$\text{moveL}(t) = \begin{bmatrix} x(t) & y(t) & z(t) & \theta(t) \end{bmatrix}^T$$

Where

$$x(t) = {}^S_E T_x, \quad y(t) = {}^S_E T_y, \quad z(t) = {}^S_E T_z, \quad \theta(t) = \frac{cos^{-1}({}^S_E T_{11} + {}^S_E T_{22} + {}^S_E T_{33} - 1)}{2}$$

Do note that a rotation matrix representation of θ can be made using the below equation:

$$
{}^{S}_{E}R = \begin{bmatrix} k_x \cdot k_x \cdot (1-\cos\theta) + \cos\theta & k_x \cdot k_y \cdot (1-\cos\theta) - k_z \cdot \sin\theta & k_x \cdot k_z \cdot (1-\cos\theta) + k_y \cdot \sin\theta \\ k_y \cdot k_x \cdot (1-\cos\theta) + k_z \cdot \sin\theta & k_y \cdot k_y \cdot (1-\cos\theta) + \cos\theta & k_y \cdot k_z \cdot (1-\cos\theta) - k_x \cdot \sin\theta \\ k_z \cdot k_x \cdot (1-\cos\theta) - k_y \cdot \sin\theta & k_z \cdot k_y \cdot (1-\cos\theta) + k_x \cdot \sin\theta & k_z \cdot k_z \cdot (1-\cos\theta) + \cos\theta \end{bmatrix}
$$

Where

$$
K = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} = \frac{1}{2 \cdot \sin(\theta)} \cdot \begin{bmatrix} {}^{S}_{E}T_{32} - {}^{S}_{E}T_{23} \\ {}^{S}_{E}T_{13} - {}^{S}_{E}T_{31} \\ {}^{S}_{E}T_{21} - {}^{S}_{E}T_{12} \end{bmatrix}
$$

5.2 Trajectory Planning

The trajectory planner of any robotic arm can be set to run in a meriat of ways. In this repository, two such trajectory planners have been used.

- **Cubic Polynomial**
- **Parabolic Blend** aka. Linear Segments with Parabolic Blend (LSPB)

Note that for any joint or DOF over time $\theta(t)$, movement between 2 positions may be described using these 4 constraints:

- $\theta(0) = \theta_0$
- $\theta(f) = \theta_f$
- $\dot{\theta}(0) = \dot{\theta}_0$
- $\dot{\theta}(f) = \dot{\theta}_f$

5.2.1 Cubic Polynomial

The 4 contraints described above may be satisfied using a cubic polynomial such as the one seen below:

$$
\theta(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3, t \in [0, t_f]
$$

Using this cubic polynomial, the velocity of the trajectory can be found as its derivarive $\dot{\theta}(t)$ and the acceleration as the second derivative $\ddot{\theta}(t)$.

$$
\dot{\theta}(t) = a_1 + 2a_2 \cdot t + 3a_3 \cdot t^2, t \in [0, t_f]
$$
$$
\ddot{\theta}(t) = 2a_2 + 6a_3 \cdot t, t \in [0, t_f]
$$

Using $\theta(t)$ and $\dot{\theta}(t)$, the 4 constraints can be expressed as the following equations:

- $\theta(0) = a_0$
- $\theta(f) = a_0 + a_1 \cdot t_f + a_2 \cdot t_f^2 + a_3 \cdot t_f^3$
- $\dot{\theta}(0) = a_1$

- $\dot{\theta}(f) = a_1 + 2a_2 \cdot t + 3a_3 \cdot t^2$

Solving for the parameters a_0, a_1, a_2, and a_3 using the above 4 expressions, the following equations are obtained:

- $a_0 = \theta(0)$
- $a_1 = \dot{\theta}(0)$
- $a_2 = \frac{3}{t_f^2} \cdot (\theta_f - \theta_0) - \frac{2}{t_f} \cdot \dot{\theta} - \frac{1}{t_f} \cdot \dot{\theta}_f$
- $a_3 = -\frac{2}{t_f^3} \cdot (\theta_f - \theta_0) + \frac{1}{t_f^2} \cdot (\dot{\theta}_0 + \dot{\theta}_f)$

Fig. 5 show the plots of the cubic polynomial, its first derivative (velocity), and its second derivative (acceleration). Assuming the robotic arm has n joints, a matrix-vector equation can be established to describe the cubic polynomial of each of these joints:

$$\text{moveJ}(t) = \begin{bmatrix} a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \dots & \dots & \dots & \dots \\ a_{n,0} & a_{n,1} & a_{n,2} & a_{n,3} \end{bmatrix} \cdot \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix}^T, \text{for n joints}$$

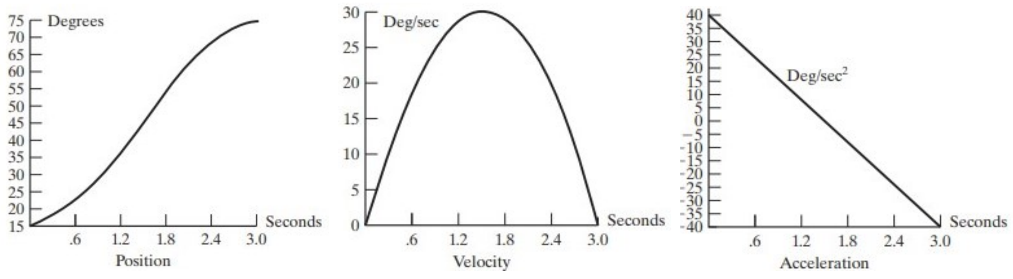


Figure 5: Cubic polynomial, its derivative, and second derivative

5.2.2 Linear Segments with Parabolic Blend (LSPB)

The parabolic blend is derived using the 3 Kinematics Equations in terms of rotation $\theta(t)$

$$\theta(t) = \tfrac{1}{2}\ddot{\theta} \cdot t^2 + \dot{\theta}_0 \cdot t + \theta_0$$
$$\dot{\theta}(t) = \ddot{\theta} \cdot t + \dot{\theta}_0$$
$$\dot{\theta}(t)^2 = \dot{\theta}_0^2 + 2\ddot{\theta} \cdot (\theta_f - \theta_0)$$

Using the 4 constraints mentioned above, these 3 Kinematics Equations wind up like this:

$$\theta(t) = \tfrac{1}{2}\ddot{\theta} \cdot t^2 + \theta_0$$
$$\dot{\theta}(t) = \ddot{\theta} \cdot t$$
$$\dot{\theta}(t)^2 = 2\ddot{\theta} \cdot (\theta_f - \theta_0)$$

Using these 3 equations, the 3 segments of the Parabolic Blend can be defined as:

$$\theta(t) = \begin{cases} \theta_0 + \frac{1}{2}\ddot{\theta} \cdot t^2, & \text{if } t_0 \le t \le t_b \\ \theta_b + a \cdot t_b \cdot (t - t_b), & \text{if } t_b < t < t_f - t_b \\ \theta_f - \frac{1}{2}\ddot{\theta} \cdot t^2 + \theta_0, & \text{if } t_f - t_b \le t \le t_f \end{cases} \Bigg\}, \text{for } \left[t_0, t_f\right]$$

Where

$$t_b = \frac{t_f}{2} - \text{abs}\left(\sqrt{\frac{\ddot{\theta}^2 \cdot t_f^2 - 4 \cdot \ddot{\theta} \cdot (Pe - Ps)}{(2 \cdot \ddot{\theta})}}\right)$$

The parabolic blend comes with its own condition, because it has to comply with the 3 Kinematic Equations. Using the third Kinematic Equation, this condition can be derived assuming no overlap occurs between the 3 segments meaning, $t_f \ge 2t_b$ must be true:

$$\dot{\theta}(t_b)^2 = 2\ddot{\theta} \cdot (\theta_f - \theta_0) \rightarrow \dot{\theta}(t_f)^2 \ge 4\ddot{\theta} \cdot (\theta_f - \theta_0) \rightarrow (\ddot{\theta} \cdot t_f)^2 \ge 4\ddot{\theta} \cdot (\theta_f - \theta_0) \rightarrow \ddot{\theta} \ge \frac{4 \cdot (\theta_f - \theta_0)}{t_f^2}$$

See Fig. 6 for a graphical representation of the Parabolic Blend trajectory, its velocity, and its acceleration.

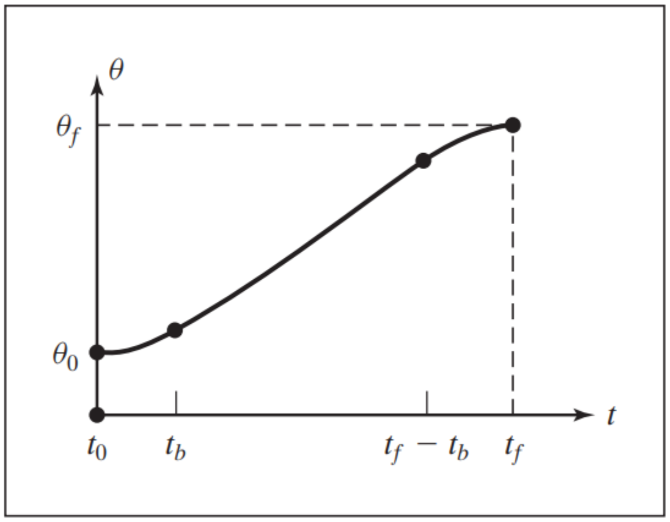


Figure 6: Parabolic blend trajectory

5.3 Calculating execution time t_f

The used method of calculating execution time t_f of any trajectory makes use of the assumption that a parabolic blend is used.

Consider that the Parabolic Blend has a maximum velocity v_{max} and maximum acceleration a_{max}. This means the Parabolic Blend has a velocity of v_{max} during its linear segment and an acceleration of a_{max} during acceleration (negative a_{max} during deceleration). This way, the time t_b becomes equal to:

$$t_b = v_{max}/a_{max}$$

Using t_b, it becomes possible to determine the execution time t_f. In cases where $2t_b = t_f$, t_f becomes:

$$t_f = \sqrt{\frac{4\cdot(\theta_f - \theta_0)}{a_{max}}}, \text{if } 2t_b \geq t_f$$

Note how even though the expression $2t_b > t_f$ is theoretically implausible, it accounts for cases when the specified v_{max} surpases the maximum reached velocity of the specified Parabolic Blend.

On the other hand, when $2t_b < t_f$, the linear segment has to be taken into account. This is done by using the below expression:

$$t_f = t_b + \frac{\theta_f - \theta_0}{v_{max}}, \text{if } 2t_b < t_f$$

Using the two above equation describing execution time t_f, the theoretically lowest t_f is acquired assuming the specified v_{max} and a_{max} apply to the trajectory. That is to say these equations assume a parabolic blend is used as the trajectory planner.

Chapter 6: Appendices

6.1 Euler Angles

Euler angles are used to describe the orientation of a rigid body with respect to a fixed coordinate system. There are 12 sets of Euler angles (intrinsic) and 12 sets of fixed (extrinsic) Euler angles, each corresponding to different rotation sequences.

$$R_{X'Y'Z'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\beta c\gamma & -c\beta s\gamma & s\beta \\ s\alpha s\beta c\gamma + c\alpha s\gamma & -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta \\ -c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha s\beta s\gamma + s\alpha c\gamma & c\alpha c\beta \end{bmatrix}$$

$$R_{X'Z'Y'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\beta c\gamma & -s\beta & c\beta s\gamma \\ c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma \\ s\alpha s\beta c\gamma - c\alpha s\gamma & s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{Y'X'Z'}(\alpha,\beta,\gamma)=\begin{bmatrix} s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & s\alpha c\beta \\ c\beta s\gamma & c\beta c\gamma & -s\beta \\ c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha c\beta \end{bmatrix},$$

$$R_{Y'Z'X'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\alpha c\beta & -c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha s\beta s\gamma + s\alpha c\gamma \\ s\beta & c\beta c\gamma & -c\beta s\gamma \\ -s\alpha c\beta & s\alpha s\beta c\gamma + c\alpha s\gamma & -s\alpha s\beta s\gamma + c\alpha c\gamma \end{bmatrix}$$

$$R_{Z'X'Y'}(\alpha,\beta,\gamma)=\begin{bmatrix} -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta & s\alpha s\beta c\gamma + c\alpha s\gamma \\ c\alpha s\beta s\gamma + s\alpha c\gamma & c\alpha c\beta & -c\alpha s\beta c\gamma + s\alpha s\gamma \\ -c\beta s\gamma & s\beta & c\beta c\gamma \end{bmatrix}$$

$$R_{Z'Y'X'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix},$$

(a) Intrinsic Euler angles set 1

$$R_{X'Y'X'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\beta & s\beta s\gamma & s\beta c\gamma \\ s\alpha s\beta & -s\alpha c\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta c\gamma - c\alpha s\gamma \\ -c\alpha s\beta & c\alpha c\beta s\gamma + s\alpha c\gamma & c\alpha c\beta c\gamma - s\alpha s\gamma \end{bmatrix}$$

$$R_{X'Z'X'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\beta & -s\beta c\gamma & s\beta s\gamma \\ c\alpha s\beta & c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha c\beta s\gamma - s\alpha c\gamma \\ s\alpha s\beta & s\alpha c\beta c\gamma + c\alpha s\gamma & -s\alpha c\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{Y'X'Y'}(\alpha,\beta,\gamma)=\begin{bmatrix} -s\alpha c\beta s\gamma + c\alpha c\gamma & s\alpha s\beta & s\alpha c\beta c\gamma + c\alpha s\gamma \\ s\beta s\gamma & c\beta & -s\beta c\gamma \\ -c\alpha c\beta s\gamma - s\alpha c\gamma & c\alpha s\beta & c\alpha c\beta c\gamma - s\alpha s\gamma \end{bmatrix},$$

$$R_{Y'Z'Y'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha s\beta & c\alpha c\beta s\gamma + s\alpha c\gamma \\ s\beta c\gamma & c\beta & s\beta s\gamma \\ -s\alpha c\beta c\gamma - c\alpha s\gamma & s\alpha s\beta & -s\alpha c\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{Z'X'Z'}(\alpha,\beta,\gamma)=\begin{bmatrix} -s\alpha c\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta c\gamma - c\alpha s\gamma & s\alpha s\beta \\ c\alpha c\beta s\gamma + s\alpha c\gamma & c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha s\beta \\ s\beta s\gamma & s\beta c\gamma & c\beta \end{bmatrix},$$

$$R_{Z'Y'Z'}(\alpha,\beta,\gamma)=\begin{bmatrix} c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha c\beta s\gamma - s\alpha c\gamma & c\alpha s\beta \\ s\alpha c\beta c\gamma + c\alpha s\gamma & -s\alpha c\beta s\gamma + c\alpha c\gamma & s\alpha s\beta \\ -s\beta c\gamma & s\beta s\gamma & c\beta \end{bmatrix}.$$

(b) Intrinsic Euler angles set 2

Figure 7: 12 sets of Euler angles (intrinsic)

$$R_{XYZ}(\gamma,\beta,\alpha)=\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix},$$

$$R_{XZY}(\gamma,\beta,\alpha)=\begin{bmatrix} c\alpha c\beta & -c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha s\beta s\gamma + s\alpha c\gamma \\ s\beta & c\beta c\gamma & -c\beta s\gamma \\ -s\alpha c\beta & s\alpha s\beta c\gamma + c\alpha s\gamma & -s\alpha s\beta s\gamma + c\alpha c\gamma \end{bmatrix}$$

$$R_{YXZ}(\gamma,\beta,\alpha)=\begin{bmatrix} -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta & s\alpha s\beta c\gamma + c\alpha s\gamma \\ c\alpha s\beta s\gamma + s\alpha c\gamma & c\alpha c\beta & -c\alpha s\beta c\gamma + s\alpha s\gamma \\ -c\beta s\gamma & s\beta & c\beta c\gamma \end{bmatrix}$$

$$R_{YZX}(\gamma,\beta,\alpha)=\begin{bmatrix} c\beta c\gamma & -s\beta & c\beta s\gamma \\ c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma \\ s\alpha s\beta c\gamma - c\alpha s\gamma & s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{ZXY}(\gamma,\beta,\alpha)=\begin{bmatrix} s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & s\alpha c\beta \\ c\beta s\gamma & c\beta c\gamma & -s\beta \\ c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha c\beta \end{bmatrix},$$

$$R_{ZYX}(\gamma,\beta,\alpha)=\begin{bmatrix} c\beta c\gamma & -c\beta s\gamma & s\beta \\ s\alpha s\beta c\gamma + c\alpha s\gamma & -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta \\ -c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha s\beta s\gamma + s\alpha c\gamma & c\alpha c\beta \end{bmatrix},$$

$$R_{XYX}(\gamma,\beta,\alpha)=\begin{bmatrix} c\beta & s\beta s\gamma & s\beta c\gamma \\ s\alpha s\beta & -s\alpha c\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta c\gamma - c\alpha s\gamma \\ -c\alpha s\beta & c\alpha c\beta s\gamma + s\alpha c\gamma & c\alpha c\beta c\gamma - s\alpha s\gamma \end{bmatrix},$$

$$R_{XZX}(\gamma,\beta,\alpha)=\begin{bmatrix} c\beta & -s\beta c\gamma & s\beta s\gamma \\ c\alpha s\beta & c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha c\beta s\gamma - s\alpha c\gamma \\ s\alpha s\beta & s\alpha c\beta c\gamma + c\alpha s\gamma & -s\alpha c\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{YXY}(\gamma,\beta,\alpha)=\begin{bmatrix} -s\alpha c\beta s\gamma + c\alpha c\gamma & s\alpha s\beta & s\alpha c\beta c\gamma + c\alpha s\gamma \\ s\beta s\gamma & c\beta & -s\beta c\gamma \\ -c\alpha c\beta s\gamma - s\alpha c\gamma & c\alpha s\beta & c\alpha c\beta c\gamma - s\alpha s\gamma \end{bmatrix},$$

$$R_{YZY}(\gamma,\beta,\alpha)=\begin{bmatrix} c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha s\beta & c\alpha c\beta s\gamma + s\alpha c\gamma \\ s\beta c\gamma & c\beta & s\beta s\gamma \\ -s\alpha c\beta c\gamma - c\alpha s\gamma & s\alpha s\beta & -s\alpha c\beta s\gamma + c\alpha c\gamma \end{bmatrix},$$

$$R_{ZXZ}(\gamma,\beta,\alpha)=\begin{bmatrix} -s\alpha c\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta c\gamma - c\alpha s\gamma & s\alpha s\beta \\ c\alpha c\beta s\gamma + s\alpha c\gamma & c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha s\beta \\ s\beta s\gamma & s\beta c\gamma & c\beta \end{bmatrix},$$

$$R_{ZYZ}(\gamma,\beta,\alpha)=\begin{bmatrix} c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha c\beta s\gamma - s\alpha c\gamma & c\alpha s\beta \\ s\alpha c\beta c\gamma + c\alpha s\gamma & -s\alpha c\beta s\gamma + c\alpha c\gamma & s\alpha s\beta \\ -s\beta c\gamma & s\beta s\gamma & c\beta \end{bmatrix}.$$

Figure 8: 12 sets of fixed (extrinsic) Euler angles

6.2 Algebraic Formula

Cross product: $\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$

Trigonometric Identities:

$$\sin(\alpha)^2 + \cos(\alpha)^2 = 1$$
$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$
$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$

Determinant:

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$
$$= aei + bfg + cdh - ceg - bdi - afh$$

To compute atan2(a, b), use the following definition:

$$\operatorname{atan2}(a, b) = \begin{cases} \arctan\left(\frac{a}{b}\right) & \text{if } b > 0 \\ \frac{\pi}{2} & \text{if } b = 0, a > 0 \\ \text{undefined} & \text{if } b = 0, a = 0 \\ -\frac{\pi}{2} & \text{if } b = 0, a < 0 \\ \arctan\left(\frac{a}{b}\right) + \pi & \text{if } b < 0 \end{cases}$$

6.3 Solutions to Common Trigonometric Equations

The following formula presents solutions to some commonly used algebraic (trigonometric) equations in robotics, as compiled from Craig's book [4].

The single equation

$$\sin\theta = a \tag{C.1}$$

has two solutions, given by

$$\theta = \pm\mathrm{Atan2}(\sqrt{1-a^2}, a). \tag{C.2}$$

Likewise, given

$$\cos\theta = b, \tag{C.3}$$

there are two solutions:

$$\theta = \mathrm{Atan2}(b, \pm\sqrt{1-b^2}). \tag{C.4}$$

If both (C.1) and (C.3) are given, then there is a unique solution given by

$$\theta = \mathrm{Atan2}(a, b). \tag{C.5}$$

The transcendental equation

$$a\cos\theta + b\sin\theta = 0 \tag{C.6}$$

has the two solutions

$$\theta = \mathrm{Atan2}(a, -b) \tag{C.7}$$

and

$$\theta = \mathrm{Atan2}(-a, b). \tag{C.8}$$

The equation

$$a\cos\theta + b\sin\theta = c, \tag{C.9}$$

which we solved in Section 4.5 with the tangent-of-the-half-angle substitutions, is also solved by

$$\theta = \mathrm{Atan2}(b, a) \pm \mathrm{Atan2}(\sqrt{a^2+b^2-c^2}, c). \tag{C.10}$$

The set of equations

$$\begin{aligned} a\cos\theta - b\sin\theta &= c, \\ a\sin\theta + b\cos\theta &= d, \end{aligned} \tag{C.11}$$

which was solved in Section 4.4, also is solved by

$$\theta = \mathrm{Atan2}(ad - bc,\ ac + bd). \tag{C.12}$$

References

[1] K. J. Waldron, G. L. Kinzel, and S. K. Agrawal, *Kinematics, Dynamics, and Design of Machinery*. Hoboken, NJ, USA: Wiley, 3rd ed., 2016.

[2] J. M. McCarthy and G. S. Soh, *Geometric Design of Linkages*. New York, NY, USA: Springer, 2011.

[3] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge, UK: Cambridge University Press, 2017.

[4] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Upper Saddle River, NJ, USA: Pearson-/Prentice Hall, 3rd ed., 2005.