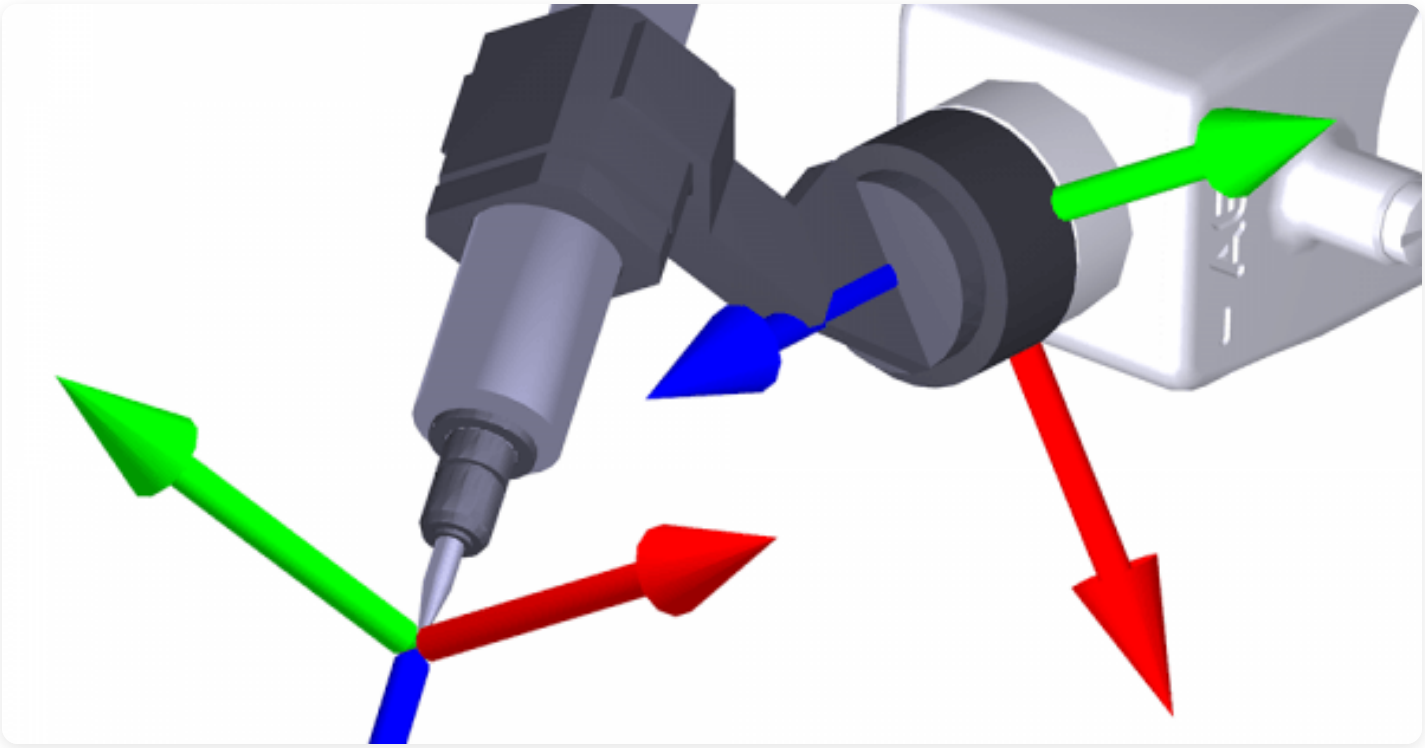
Lesson 3: 3D Rotation Matrices and Spatial Transformations



Modified: Jul. 30, 2026

Published: Nov. 21, 2025



Become Contributor

Expert Support

Describing a 2D rotation takes one number: the angle. Describing a 3D rotation is far harder. Euler angles are intuitive but suffer from gimbal lock, a singularity where two axes align and the system loses a degree of freedom, which is exactly the bug that

caused Apollo-era navigation headaches and still crashes drone control loops today. Rotation matrices avoid gimbal lock entirely but require careful composition order, because 3D rotations do not commute (rotating X then Y gives a different result than Y then X). In this lesson you will work through every major 3D rotation representation: coordinate axis rotations, Euler angle sequences, the Rodrigues formula for arbitrary axis rotation, and 4x4 homogeneous transformations, applied to robotic welding, drone gimbals, and satellite attitude control. #3DRotations #EulerAngles #GimbalLock

Learning Objectives

By the end of this lesson, you will be able to:

1. **Construct** rotation matrices for rotations about coordinate axes and arbitrary vectors
2. **Apply** Euler angle sequences for systematic 3D orientation representation
3. **Compose** 4×4 homogeneous transformation matrices for unified spatial motion
4. **Analyze** transformation sequences understanding order-dependent composition

Real-World Engineering Challenge: 3D Spatial Orientation Control

Spatial **orient**ation **cont**rol is **fundam**ental **acr**oss **robo**tics, **aeros**pace, **comp**uter **grap**hics, and **autom**ation. **Fr**om **indus**trial **rob**ots **manipu**lating **par**ts with **prec**ise **to**ol **ang**les, to **space**craft **mainta**ining **atti**tude in **orb**it, to **anima**tion **syst**ems **rota**ting 3D **mod**els - **al**l **requ**ire **mathem**atical **frame**works for **repres**enting and **compo**sing 3D **rotat**ions and **transf**ormations.

Representative Systems

3D Spatial Control Applications:

- **Industrial Robots** (6-DOF manipulators) - tool orientation for welding, machining, assembly
- **Aerospace Systems** (satellites, aircraft) - attitude control and orientation tracking
- **Computer Graphics** (animation, CAD) - 3D object rotation and camera positioning
- **Motion Capture Systems** (biomechanics, VR) - tracking 3D body segment orientations
- **Coordinate Measuring Machines** (CMMs) - probe orientation for complex surface inspection
- **Gimbals and Stabilizers** (cameras, sensors) - maintaining orientation despite platform motion

The 3D Rotation Challenge

These systems require precise control of:

Critical 3D Orientation Requirements

- **Orientation representation** using rotation matrices, Euler angles, or quaternions
- **Composition of rotations** understanding order-dependent matrix multiplication
- **Singularity management** avoiding gimbal lock and numerical instabilities
- **Coordinate frame transformations** relating multiple reference frames systematically
- **Smooth orientation interpolation** for continuous motion and animation

Engineering Question: How do we mathematically represent and compose complex 3D orientations in a systematic way that handles singularities and works across diverse applications?

Why 3D Spatial Mathematics Matters

Consequences of Poor Mathematical Foundation:

- **Orientation errors** from incorrect rotation sequences or coordinate confusion
- **Gimbal lock** causing loss of degrees of freedom at singular configurations
- **Numerical instability** from poorly conditioned transformations
- **Programming complexity** without systematic transformation framework
- **Limited scalability** to more complex multi-body or hierarchical systems

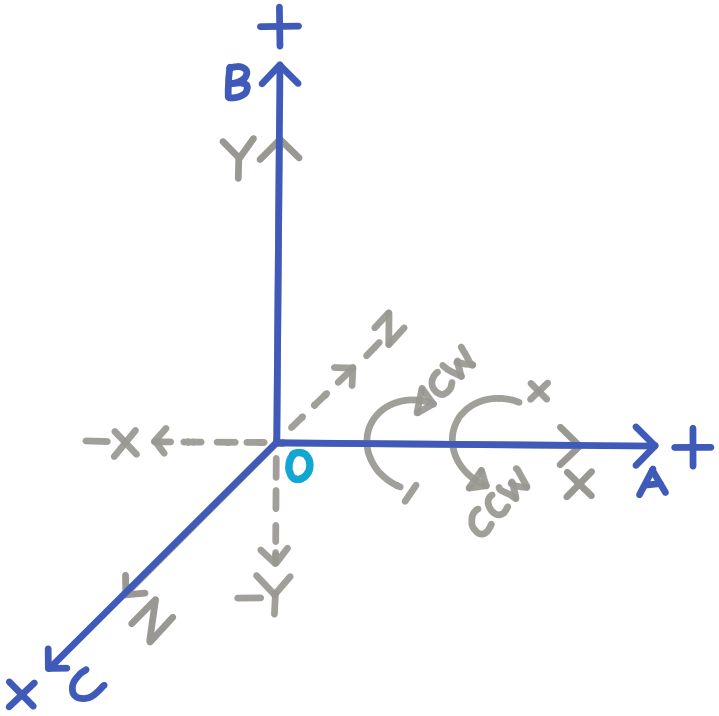
Benefits of Systematic 3D Analysis:

- **Precise orientation control** using robust mathematical representations
- **Predictable behavior** through systematic matrix composition rules
- **Singularity awareness** enabling detection and avoidance strategies
- **Unified framework** applicable across robotics, aerospace, and graphics
- **Foundation for advanced topics** (quaternions, screws, differential kinematics)

Fundamental Theory: 3D Rotation Mathematics

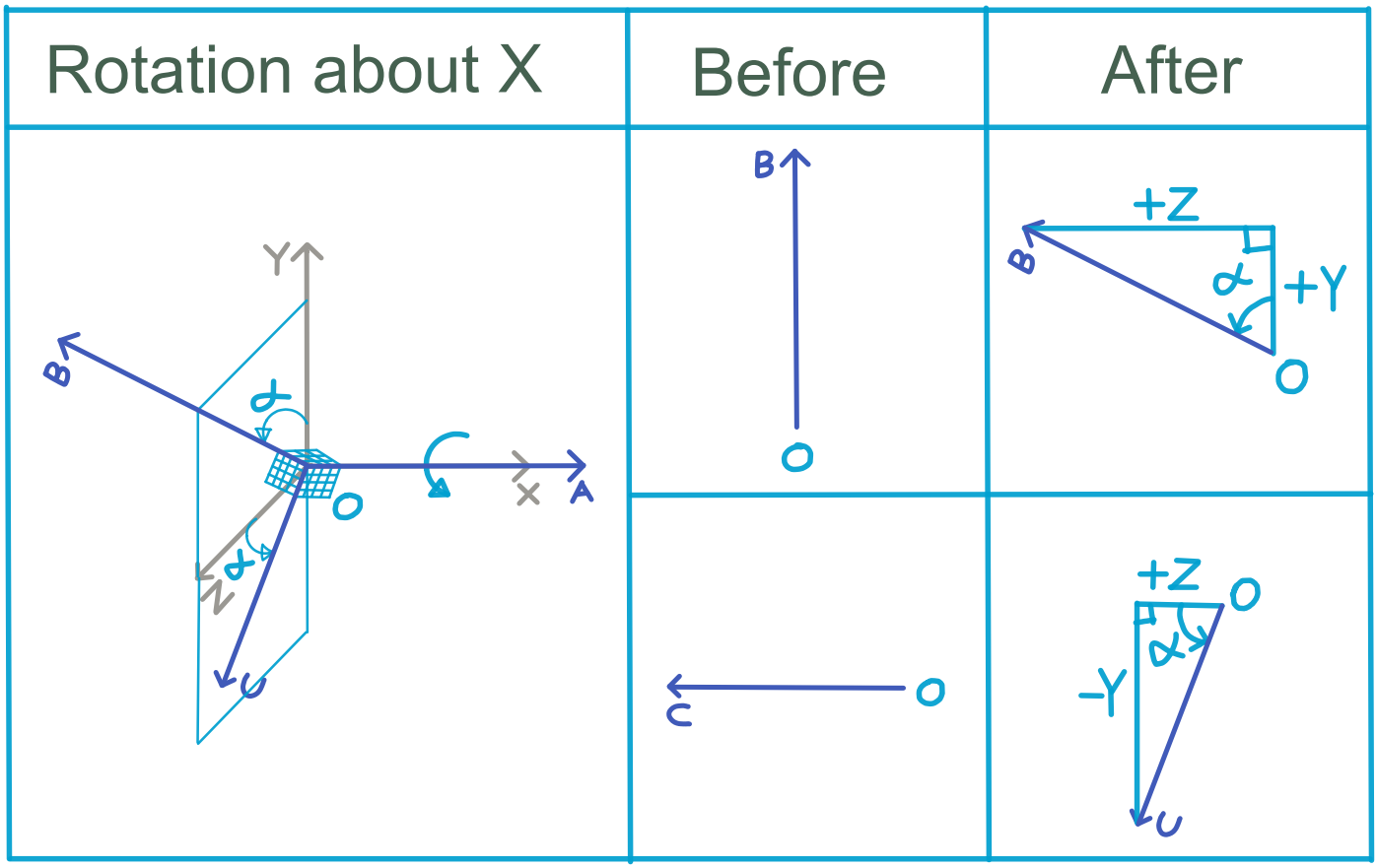
Basic Rotation Matrices About Coordinate Axes

3D **rotat**ions are **mo**re **comp**lex **th**an 2D **beca**use **rota**tion **ord**er **matt**ers and **mult**iple **repres**entations **exi**st. **Bas**ic **rotat**ions **abo**ut **coord**inate **ax**es **prov**ide the **buil**ding **blo**cks for **al**l **spat**ial **orient**ations. **Follo**wing the **establ**ished **ax**is **conve**ntion **whe**re **counter**clockwise **rota**tion is **posi**tive, we can **der**ive **rota**tion **matr**ices for **ea**ch **coord**inate **ax**is. **Ea**ch **rota**tion **trans**forms a **mob**ile **fra**me (A, B, C) **rela**tive to a **fix**ed **fra**me (X, Y, Z).


+
B
Y
Z
CW
X
-X
O
CCW
X
A
+
-Y
Z
X
C

X-Axis Rotation | Y-Axis Rotation | Z-Axis Rotation

Rotation about X-axis by angle α:



Geometric Analysis:

After rotating axis B:

$$\cos(\alpha) = \frac{+y}{B} \quad \Rightarrow \quad b = B\cos(\alpha)$$
$$\sin(\alpha) = \frac{+z}{B} \quad \Rightarrow \quad c = B\sin(\alpha)$$

After rotating axis C:

$$\cos(\alpha) = \frac{+z}{C} \quad \Rightarrow \quad c = C\cos(\alpha)$$
$$\sin(\alpha) = \frac{-y}{C} \quad \Rightarrow \quad b = -C\sin(\alpha)$$

Transformation Summary:

Before	After
A(1,0,0)	A(1,0,0)
B(0,1,0)	B(0, cos α, sin α)
C(0,0,1)	C(0, -sin α, cos α)

🔄 X-Axis Rotation Matrix

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

Physical Meaning: Rotation about X-axis corresponds to “roll” motion - like an aircraft banking left or right. Rotates vectors around the X-axis, leaving X-coordinates unchanged while rotating Y and Z components in the YZ-plane.

Rotation Matrix Properties

📐 Essential Rotation Matrix Properties

Orthogonality: $R^T R = I$ (columns are orthonormal vectors) **Determinant:** $\det(R) = +1$ (proper rotations, no reflections) **Inverse:** $R^{-1} = R^T$ (transpose equals inverse) **Composition:** $R_3 R_2 R_1$ applies R_1 first, then R_2, then R_3

Physical Meaning: Rotation matrices preserve lengths and angles, representing pure rotations without scaling or reflection in 3D space.

4×4 Homogeneous Transformation Matrices

Extending the 2D **homogeneous** **coordinate** **concept** to 3D, we **use** 4×4 **matrices** to **unify** **rotation** and **translation** **into** a **single** **mathematical** **operation**. **This** **powerful** **framework** is the **foundation** for **all** **modern** **robot** **kinematics** and **computer** **graphics**.

🎯 Spatial Transformation Matrix

General 4×4 transformation: $T = \begin{bmatrix} R_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}$

Where:

- $R_{3\times3}$ = 3×3 rotation matrix
- $\mathbf{t}_{3\times1}$ = 3×1 translation vector
- $\mathbf{0}_{1\times3}$ = [0 0 0] zero vector
- Last element = 1 (homogeneous coordinate)

Physical Meaning: 4×4 matrices unify rotation and translation into single mathematical operation for 3D spatial transformations.

Composite 3D Transformations for Robotics

Real **robot** **control** **requires** **precise** **composition** of **multiple** **rotations** and **translations** in 3D **space**. **Understanding** the **systematic** **rules** for **matrix** **multiplication** **order** is **essential** for **accurate** **end-effector** **positioning** and **complex** **trajectory** **programming**.

🔧 3D Transformation Composition Rules

Matrix multiplication is non-commutative - order matters!

For robot positioning with multiple transformations:

1. **Initial state:** Fixed and mobile frames are coincident → **Identity matrix**
2. **Fixed frame operations:** Rotate/translate about fixed axes (X,Y,Z) → **Pre-multiply** current matrix
3. **Mobile frame operations:** Rotate/translate about mobile axes (A,B,C) → **Post-multiply** current matrix

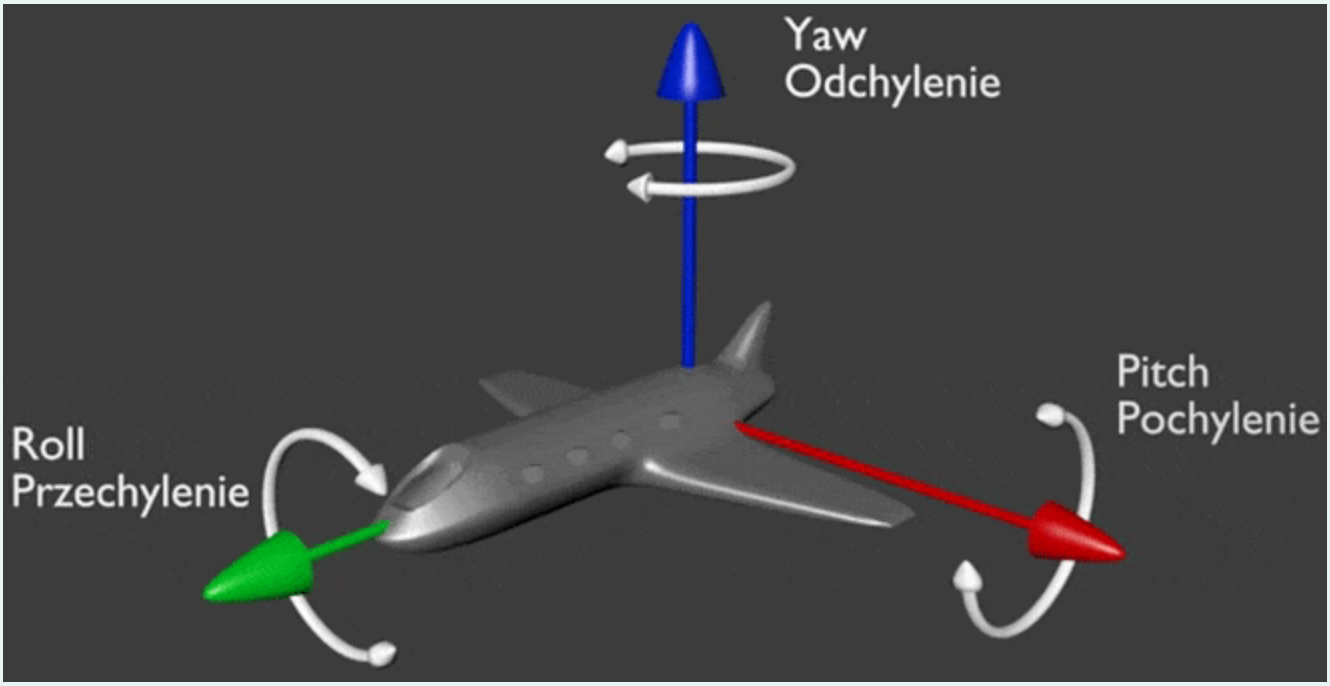
General composition: $H = H_n \cdot H_{n-1} \cdot \ldots \cdot H_2 \cdot H_1$

Where transformations are applied in sequence: H_1 first, H_n last.

Euler Angle Representations

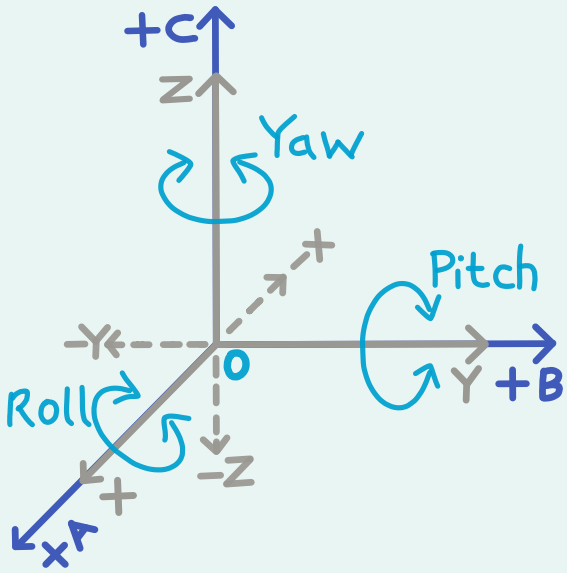
Euler **ang**les **prov**ide an **intui**tive **wa**y to **desc**ribe 3D **orient**ations **usi**ng **thr**ee **seque**ntial **rotat**ions **abo**ut **coord**inate **ax**es. **Howe**ver, **diffe**rent **seque**nces **exi**st and **singul**arities must be **caref**ully **mana**ged. **Additi**onally, the **termin**ology (**ro**ll, **pit**ch, **ya**w) **depe**nds **heav**ily on **whi**ch **coord**inate **sys**tem **conve**ntion is **bei**ng **us**ed.

Important: Coordinate System Conventions Matter!



The physical meanings of **roll**, **pitch**, and **yaw** don't change, but **which mathematical axis (X, Y, Z) corresponds to which rotation depends entirely on your coordinate system**. Always verify which coordinate system convention is being used when working with Euler angles! Different fields use different conventions:

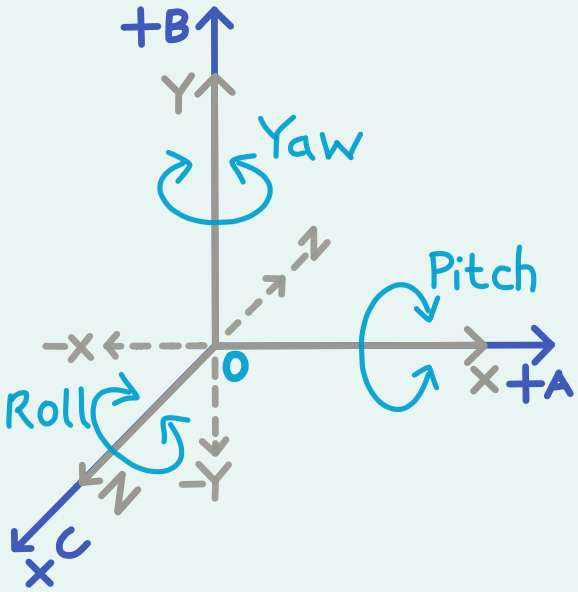
Aerospace/Aviation Convention:



- **Axes:** X = forward (nose), Y = right (wing), Z = down (belly)

- **Rotations:** Roll = rotation about X (longitudinal), Pitch = rotation about Y (lateral/horizontal), Yaw = rotation about Z (vertical)

Mathematics/Robotics Convention *(used in this course)*:



- **Axes:** X = right, Y = up, Z = forward
- **Rotations:** Roll = rotation about Z (longitudinal), Pitch = rotation about X (lateral/horizontal), Yaw = rotation about Y (vertical)

Key Point: Roll always means “banking” (rotating around the forward-pointing axis), Pitch means “nodding” (tilting up/down), and Yaw means “turning” (spinning left/right). But which axis is X, Y, or Z depends on your coordinate system!

ZYX Convention | ZYZ Convention | Singularity Issues

🎯 ZYX Euler Angles

Sequential rotations:

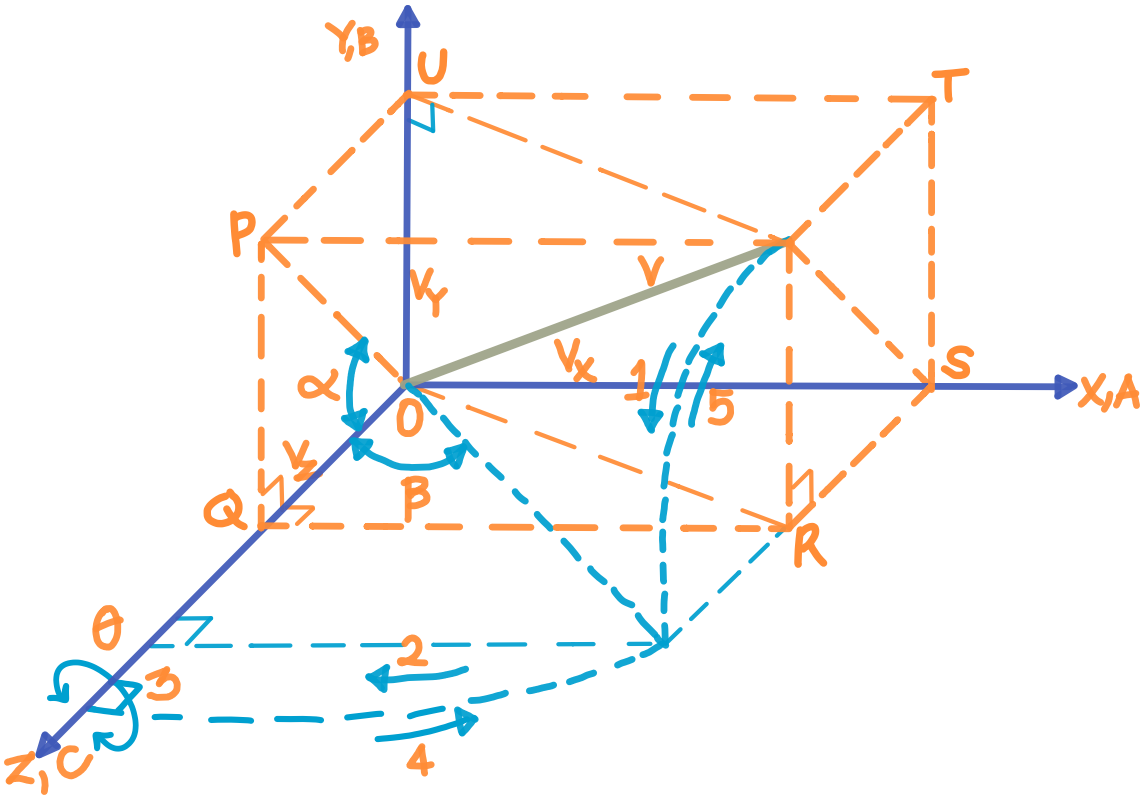
1. Rotate γ about Z-axis (Roll in Math/Robotics convention)
2. Rotate β about new Y-axis (Yaw in Math/Robotics convention)
3. Rotate α about final X-axis (Pitch in Math/Robotics convention)

Combined rotation matrix: $R_{ZYX}(\gamma, \beta, \alpha) = R_x(\alpha)R_y(\beta)R_z(\gamma)$

Physical Meaning: Commonly used in robotics and computer graphics. The sequence applies Z rotation first, then Y rotation, then X rotation.

Note: Matrix multiplication order is right-to-left, so $R_z(\gamma)$ is applied first to the vector, then $R_y(\beta)$, then $R_x(\alpha)$.

Rotation About an Arbitrary Axis Through the Origin



Rotation about an arbitrary axis is essential when the rotation cannot be decomposed into simple X, Y, Z rotations. Two primary methods exist: the decomposition approach and Rodrigues' formula. Each has specific advantages depending on the application context.

Problem Setup: Given a fixed frame OXYZ and an arbitrary rotation axis **V = (x,y,z)** with components **V_x, V_y, V_z**, we need to construct the rotation matrix **R(V,θ)** for rotation angle **θ**.

Method 1: Decomposition Approach (General Arbitrary Axis Formula)

🔄 Arbitrary Axis Rotation Strategy

Five-step decomposition process:

1. Rotation by angle **α** about X-axis
2. Rotation by angle **-β** about Y-axis
3. Rotation by angle **θ** about Z-axis

4. Rotation by angle **β** about Y-axis
5. Rotation by angle **-α** about X-axis

Matrix composition: R(V,θ) = R(x,-α) R(y,β) R(z,θ) R(y,-β) R(x,α)

The complete rotation matrix:

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

Geometric relationships: For unit vector $\|\mathbf{V}\| = 1$:

$$\sin\alpha = \frac{V_y}{\sqrt{V_y^2 + V_z^2}}, \quad \cos\alpha = \frac{V_z}{\sqrt{V_y^2 + V_z^2}}$$

$$\sin\beta = V_x, \quad \cos\beta = \sqrt{V_y^2 + V_z^2}$$

☆ ⚡ Simplified Arbitrary Axis Formula

Final rotation matrix in compact form:

$$R(V,\theta) = \begin{bmatrix} V_x^2T + C & V_xV_yT - V_zS & V_xV_zT + V_yS \\ V_xV_yT + V_zS & V_y^2T + C & V_yV_zT - V_xS \\ V_xV_zT - V_yS & V_yV_zT + V_xS & V_z^2T + C \end{bmatrix}$$

Where: C = cos θ, S = sin θ, T = (1 - cos θ)

Unit vector components:

$$V_x = \frac{x}{\sqrt{x^2+y^2+z^2}}, \quad V_y = \frac{y}{\sqrt{x^2+y^2+z^2}}, \quad V_z = \frac{z}{\sqrt{x^2+y^2+z^2}}$$

Example: 90° rotation about V = (2, 2, 2)

> **Click to reveal decomposition method calculations**

When to Use Decomposition Approach

Best for:

- Hand calculations and analytical derivations
- Understanding the geometric decomposition
- Teaching and learning rotation concepts
- When rotation axis alignment with coordinate axes is important

Advantages:

- Intuitive geometric interpretation
- Systematic step-by-step process
- Direct connection to Euler angle concepts

Disadvantages:

- More computationally expensive (multiple matrix multiplications)
- More complex to implement in code

Method 2: Rodrigues' Rotation Formula

Rodrigues' **form**ula **prov**ides a **comp**act, **eleg**ant **expre**ssion for **arbit**rary **ax**is **rotat**ions **usi**ng **mat**rix **expone**ntials and the **sk**ew-**symme**tric **mat**rix **fo**rm. **Th**is is the **prefe**rred **met**hod for **comput**ational **implem**entations.

☆ ⚡ Rodrigues' Rotation Formula

Matrix exponential form:

$$R(V,\theta) = e^{W\theta} = I + \sin(\theta)W + (1 - \cos(\theta))W^2$$

Where:

- **I** = 3×3 identity matrix
- **W** = skew-symmetric matrix of unit vector $\hat{V}$
- **θ** = rotation angle

Skew-symmetric matrix construction:

$$W = \begin{bmatrix} 0 & -V_z & V_y \\ V_z & 0 & -V_x \\ -V_y & V_x & 0 \end{bmatrix}$$

Key property: $W^2 = \hat{V}\hat{V}^T - I$ (outer product minus identity)

Note: Rodrigues formula only works for rotations about axes passing through the origin.

Example: Same 90° rotation about V = (2, 2, 2) using Rodrigues

> **Click to reveal Rodrigues method calculations**

When to Use Rodrigues Formula

Best for:

- Computational implementations (robotics, graphics, simulation)
- Quaternion-to-matrix conversions
- Real-time applications requiring efficiency
- Interpolation between orientations (SLERP)

Advantages:

- Computationally efficient (requires only 2 matrix operations)
- Elegant mathematical formulation
- Direct connection to Lie algebra/exponential map theory
- Easy to implement in code (fewer operations)

Disadvantages:

- Less intuitive geometric interpretation
- Requires understanding of skew-symmetric matrices

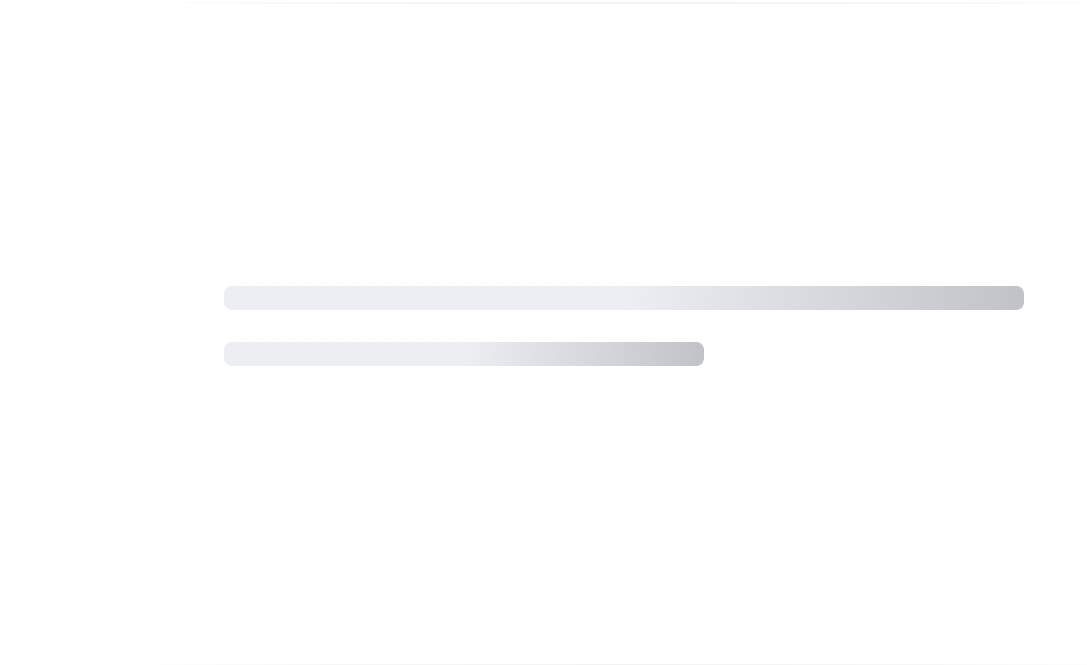
Comparison and Method Selection

Criterion	Decomposition	Rodrigues
Computational cost	High (5 matrix mults)	Low (2 matrix ops)
Code complexity	Medium	Low
Geometric intuition	Excellent	Poor
Numerical stability	Good	Excellent
Best use case	Education, analysis	Implementation

Criterion	Decomposition	Rodrigues
Connection to	Euler angles	Quaternions

General recommendation: Use Rodrigues' formula for all computational work (robot control, graphics engines, simulation). Use decomposition approach for teaching, learning, and analytical derivations.

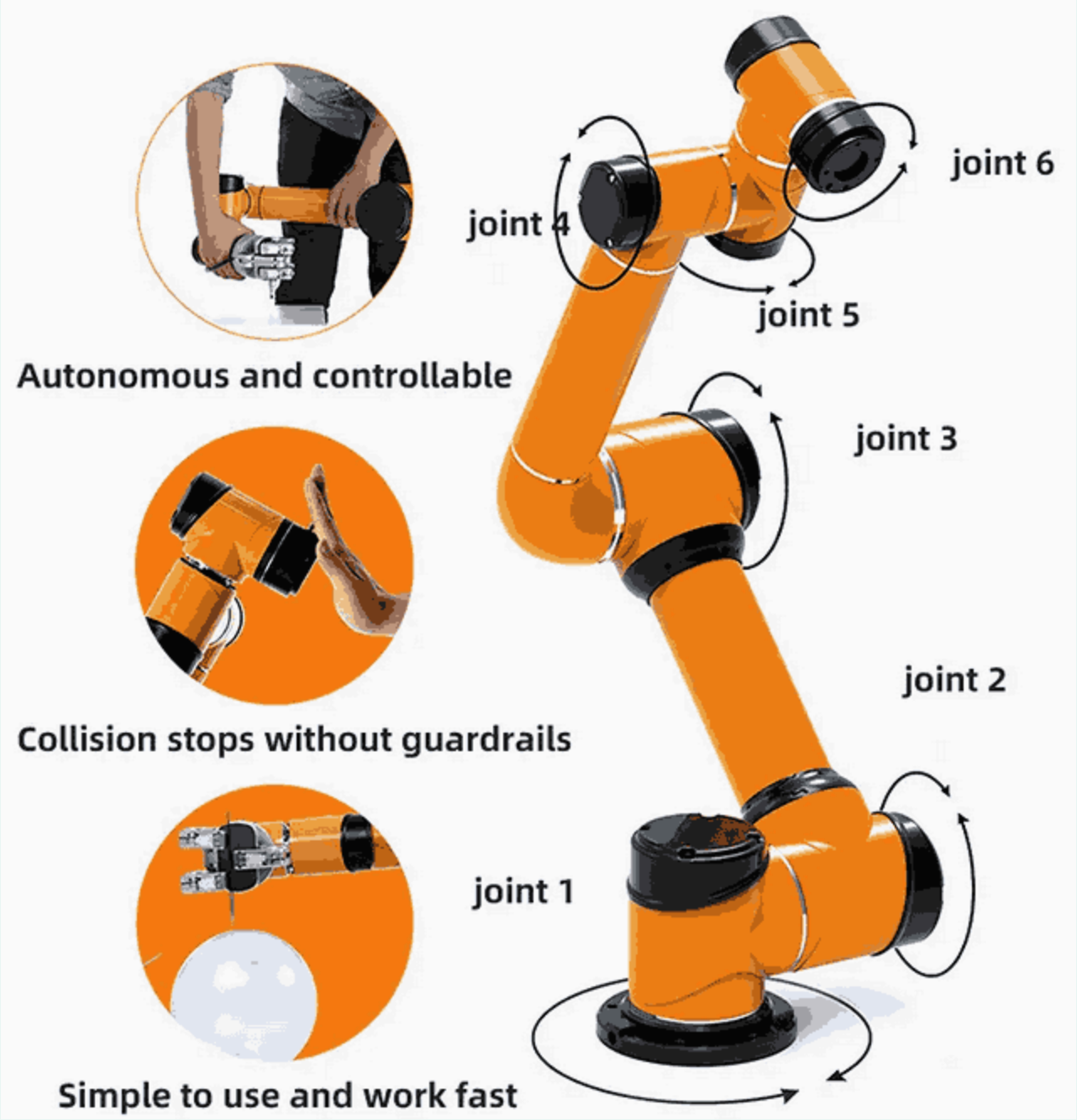
Application 1: Vision-Guided Robotic Pick-and-Place System (Manufacturing Automation)



A vision-guided robotic system uses a camera to locate workpieces for automated pick-and-place operations in a manufacturing cell.

System Problem Statement

- **Configuration:** 6-DOF robot with overhead camera system
- **Task:** Transform workpiece coordinates from camera frame to robot base frame for grasping



What we need to calculate:

1. **Inverse transformation** of camera-to-robot base relationship
2. **Workpiece position** in robot base coordinates using transformation composition

3. **Camera recalibration** effect when camera orientation changes
4. **Point transformation** through multiple coordinate frames

Key Question: How do we use homogeneous transformations to convert visual detection data into robot grasping commands?

Equivalent System Model



Given transformation matrices:

H₁ = Workpiece position relative to camera:

$$H_1 = H_{workpiece}^{camera} = \begin{bmatrix} 0 & -1 & 0 & 3 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

H₂ = Robot base position relative to camera:

$$H_2 = H_{robot}^{camera} = \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & -1 & 0 & 10 \\ 0 & 0 & -1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 1: Calculate Inverse Transformation H_2^{-1}

> **Click to reveal inverse transformation calculation**

Step 2: Calculate Workpiece Position Relative to Robot Base

> **Click to reveal workpiece coordinate transformation**

Step 3: Camera Recalibration Analysis

> **Click to reveal camera rotation effect**

Step 4: Point Transformation Through Frames

> **Click to reveal 3D point transformation example**

Engineering Insight

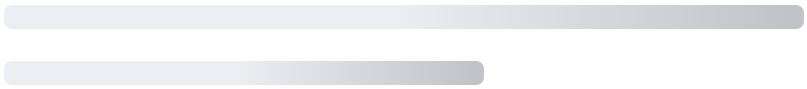
This application demonstrates **coordinate frame transformation chains** in vision-guided robotics:

1. **Matrix inversion** allows us to reverse transformation relationships (camera→base becomes base→camera)
2. **Transformation composition** systematically relates multiple coordinate frames (workpiece→camera→base)
3. **Homogeneous matrices** unify rotation and translation in a single 4×4 operation
4. **Camera calibration** is critical - any camera movement requires recalculation of all transformations

Practical Implementation: Industrial vision systems use **eye-in-hand** (camera on robot) or **eye-to-hand** (camera fixed) configurations. This example shows eye-to-hand, where camera calibration (H_2) must be precise and stable. Camera movement requires **full recalibration** of the transformation chain.

Real-world consideration: Vision systems use **feature detection** (edges, corners, patterns) to determine workpiece pose. The transformation mathematics shown here is applied after visual detection to convert pixel coordinates to robot coordinates for grasping.

Application 2: 3-DOF Robot Manipulator Composite Transformations (Assembly Automation)



A 3-DOF robot manipulator performs a pick-and-place operation requiring composite rotations about multiple axes followed by translation.

System Problem Statement

- **Configuration:** 3-DOF articulated robot with revolute joints
- **Task:** Execute precise sequential rotations and translation to position sensor payload



What we need to calculate:

1. **Individual rotation matrices** for each axis (Z, Y, X) with verification
2. **Composite rotation matrix** from sequential multiplication
3. **Homogeneous transformation** incorporating translation
4. **Point transformation** to determine final sensor position
5. **Distance verification** to confirm rotation properties

> **Key Question:** How do we systematically compose multiple rotations and verify the mathematical properties of transformation matrices?

🛠️ Equivalent System Model



Given transformation sequence:

1. **Initial rotation:** 45° about fixed Z-axis
2. **Second rotation:** 60° about fixed Y-axis
3. **Third rotation:** 30° about fixed X-axis
4. **Translation:** (2, 3, 1) units in final coordinate frame
5. **Sensor point:** P(1, 1, 0) to be transformed

Step 1: Derive Individual Rotation Matrices

> **Click to reveal rotation matrix derivations**

Step 2: Verify Orthogonality of Rotation Matrices

> **Click to reveal orthogonality verification**

Step 3: Calculate Composite Rotation Matrix

> **Click to reveal composite rotation calculation**

Step 4: Form Homogeneous Transformation Matrix

> **Click to reveal homogeneous transformation**

Step 5: Transform Sensor Point and Verify Distance

> **Click to reveal point transformation and verification**

Engineering Insight

This application demonstrates **composite 3D rotations** for robot manipulator control:

1. **Sequential rotations** are applied right-to-left in matrix multiplication: $R_x \cdot R_y \cdot R_z$ applies Z first
2. **Orthogonality verification** ($R^\top R = I$) confirms valid rotation matrices before use
3. **Homogeneous transformations** unify rotation and translation in a single 4×4 matrix operation
4. **Distance preservation** is a fundamental property of rotations (orthogonal matrices)

Matrix Multiplication Order: The order $R_x(30^\circ) \cdot R_y(60^\circ) \cdot R_z(45^\circ)$ means:

- **First:** Rotate 45° about the fixed Z-axis
- **Second:** Rotate 60° about the fixed Y-axis
- **Third:** Rotate 30° about the fixed X-axis

This is the **extrinsic (fixed-frame) convention** where all rotations are about the original coordinate axes, and the matrix multiplication order is the reverse of the physical rotation sequence.

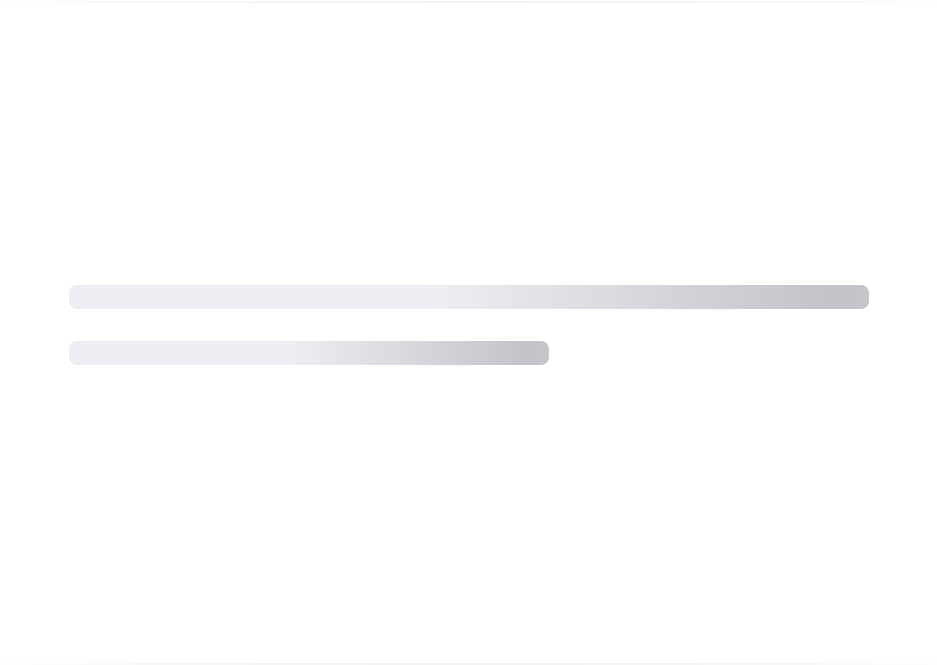
Practical Implementation: Industrial robots use **forward kinematics** to compute end-effector position from joint angles. The mathematics here shows how to:

- Build individual rotation matrices from joint angles
- Compose them systematically
- Add translation offsets
- Verify mathematical correctness through orthogonality checks

Real-world consideration: Actual robot control systems use **quaternions** for interpolation (avoiding gimbal lock) but convert to rotation matrices for final

position calculation. The systematic matrix approach shown here is fundamental to all robot motion planning algorithms.

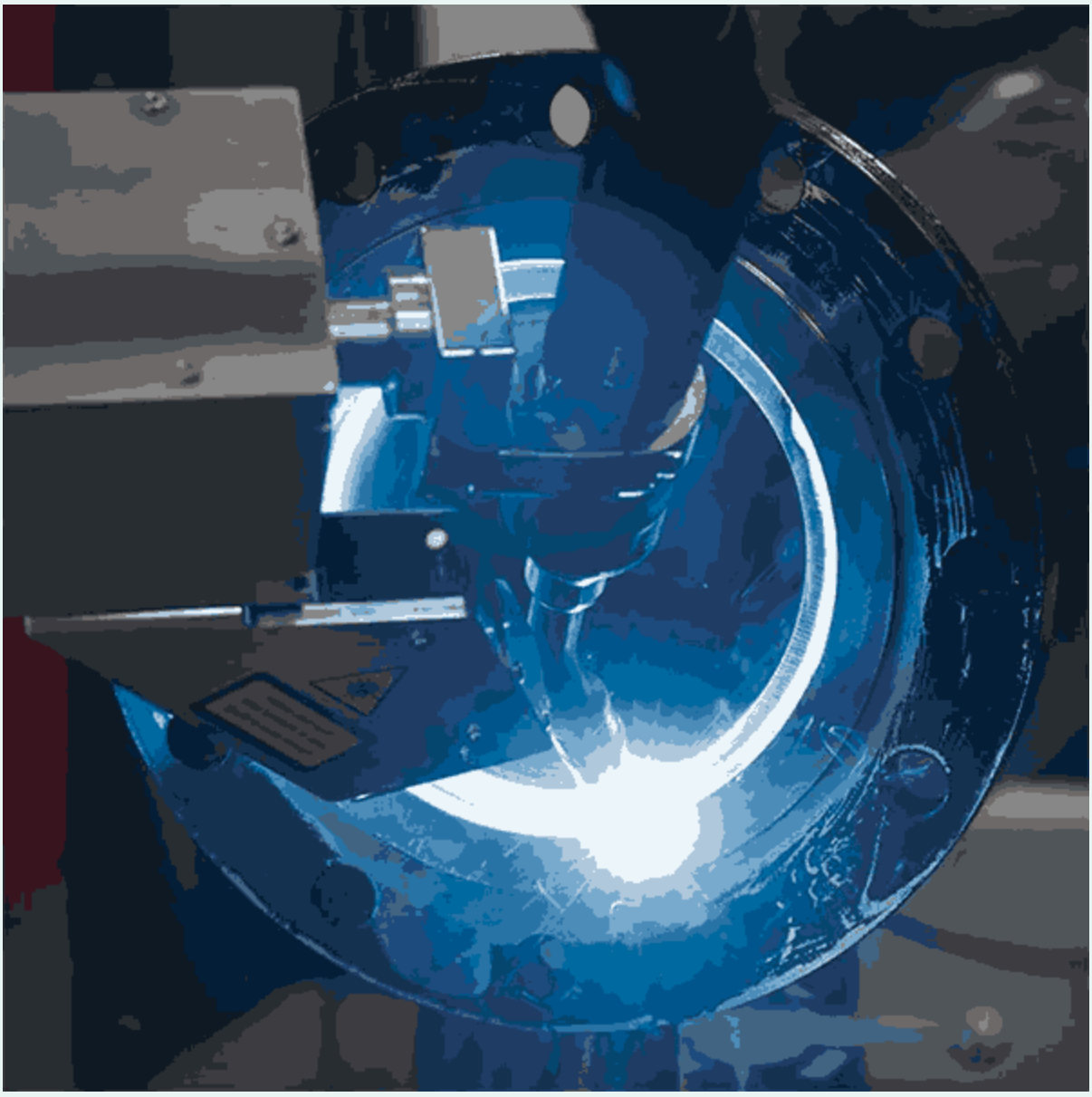
Application 3: Robotic Welding of Pipe Joints (Manufacturing)



A 3-axis welding robot must perform orbital welding around a horizontal pipe with the torch perpendicular to the pipe surface at key positions around the weld.

System Problem Statement

- **Configuration:** 3-axis welding robot with welding torch
- **Workpiece:** 200mm diameter horizontal steel pipe (aligned with X-axis)
- **Task:** Calculate torch orientation at 4 key waypoints (top, side, bottom, side) around pipe circumference



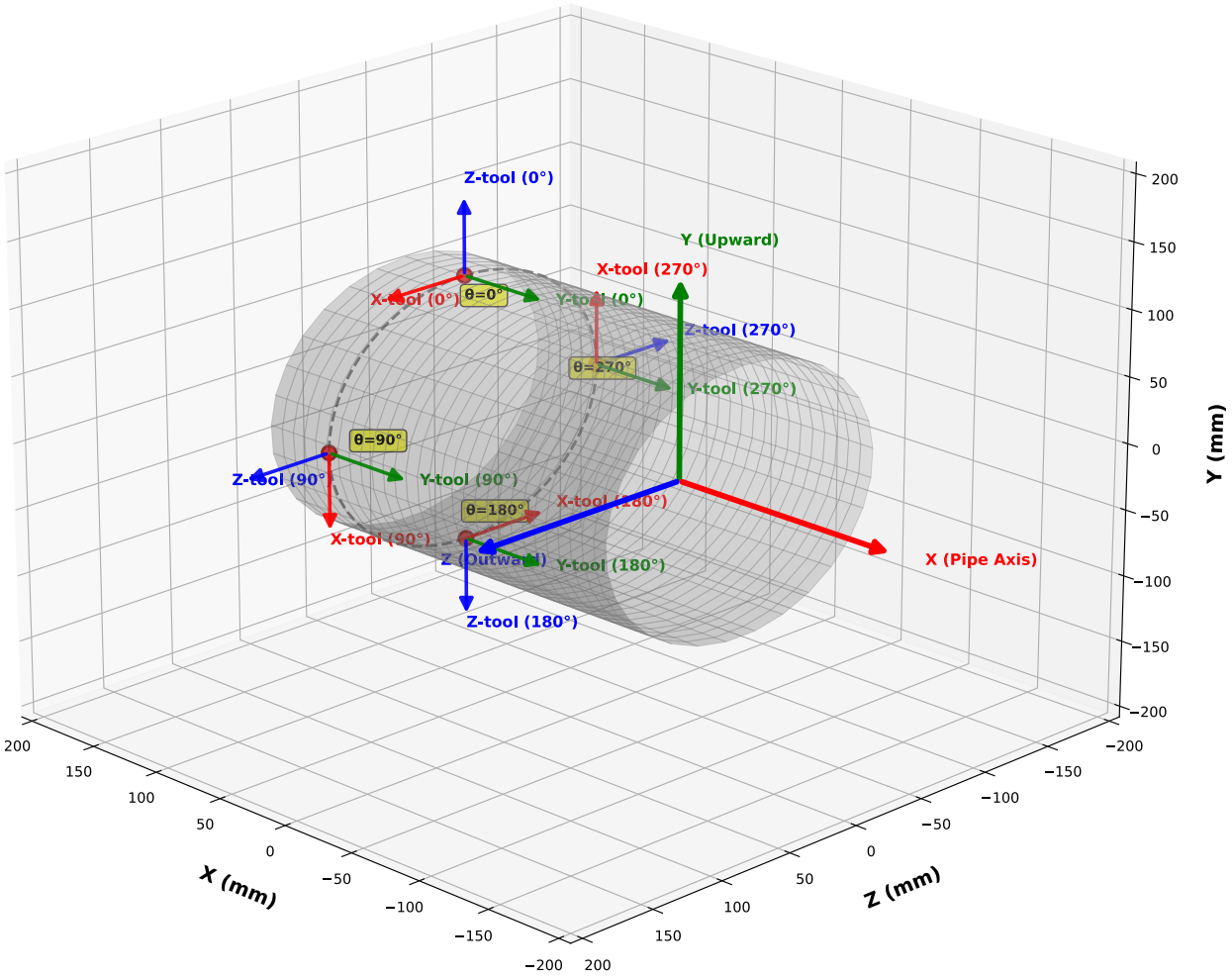
What we need to calculate:

1. **Surface normal vectors** at 4 waypoints around pipe circumference (0°, 90°, 180°, 270°)
2. **Tool orientation matrices** ensuring torch perpendicularity to surface
3. **Euler angle sequences** (ZYX convention) for robot controller

4. **Verification** that torch remains perpendicular to pipe surface

Key Question: How do we systematically calculate 3D rotation matrices to maintain torch perpendicularity at different positions around a cylindrical surface?

Equivalent System Model



Given:

- **Pipe parameters:** Radius R = 100 mm, horizontal pipe aligned with X-axis
- **Pipe centerline direction:** $\hat{p} = (1, 0, 0)$ (along X-axis)

- **Starting point:** θ = 0° at top of pipe (pointing in +Z direction)
- **Torch requirements:** Torch Z-axis must align with surface normal (pointing radially outward from pipe)
- **Waypoints:** 4 cardinal points (θ = 0°, 90°, 180°, 270°)
- **Coordinate system:** YZ-plane is perpendicular to pipe axis

Step 1: Surface Normal Calculation at Waypoints

> **Click to reveal surface normal calculations**

Step 2: Tool Orientation Matrix Construction

> **Click to reveal orientation matrix calculations**

Step 3: Euler Angle Extraction (ZYX Convention)

> **Click to reveal Euler angle extraction**

Step 4: Verification - Torch Perpendicularity Check

> **Click to reveal verification calculations**

Application 4: Drone Camera Gimbal Stabilization (Aerial Photography)

A 2-axis camera gimbal must compensate for drone tilting to keep the camera pointing straight down at the ground.

System Problem Statement

- **Configuration:** 2-axis gimbal (roll, pitch) mounted on quadcopter drone
- **Drone tilt:** Drone is tilted with roll = 10° and pitch = 20° (from wind or forward flight)
- **Task:** Calculate gimbal compensation angles to keep camera pointing straight down



What we need to calculate:

1. **Drone orientation matrix** from Euler angles (roll and pitch only, no yaw)
2. **Desired camera orientation** (pointing straight down in -Y direction)
3. **Gimbal compensation matrix** $R_{gimbal} = R_{drone}^{-1} \cdot R_{desired}$
4. **Gimbal Euler angles** (roll and pitch) extracted from compensation matrix

5. **Verification** that camera points straight down after compensation

Key Question: How do we use rotation matrix inversion to calculate gimbal stabilization commands that cancel out drone motion?

🛠️ Equivalent System Model



Given:

- **Drone attitude:** Roll $\varphi = 10°$, Pitch $\theta = 20°$, Yaw $\psi = 0°$ (ignore yaw for simplicity)
- **Camera desired orientation:** Pointing straight down (-Y direction)
- **Coordinate system:** X = forward, Y = up, Z = right
- **Gimbal convention:** Roll about X-axis, then Pitch about Y-axis

Step 1: Drone Orientation Matrix from Euler Angles

> **Click to reveal drone orientation calculations**

Step 2: Desired Camera Orientation Matrix

> **Click to reveal desired orientation calculations**

Step 3: Gimbal Compensation Calculation

> **Click to reveal gimbal compensation calculations**

Step 4: Extract Gimbal Euler Angles (Roll and Pitch)

> **Click to reveal Euler angle extraction**

Step 5: Verification - Does the Camera Point Straight Down?

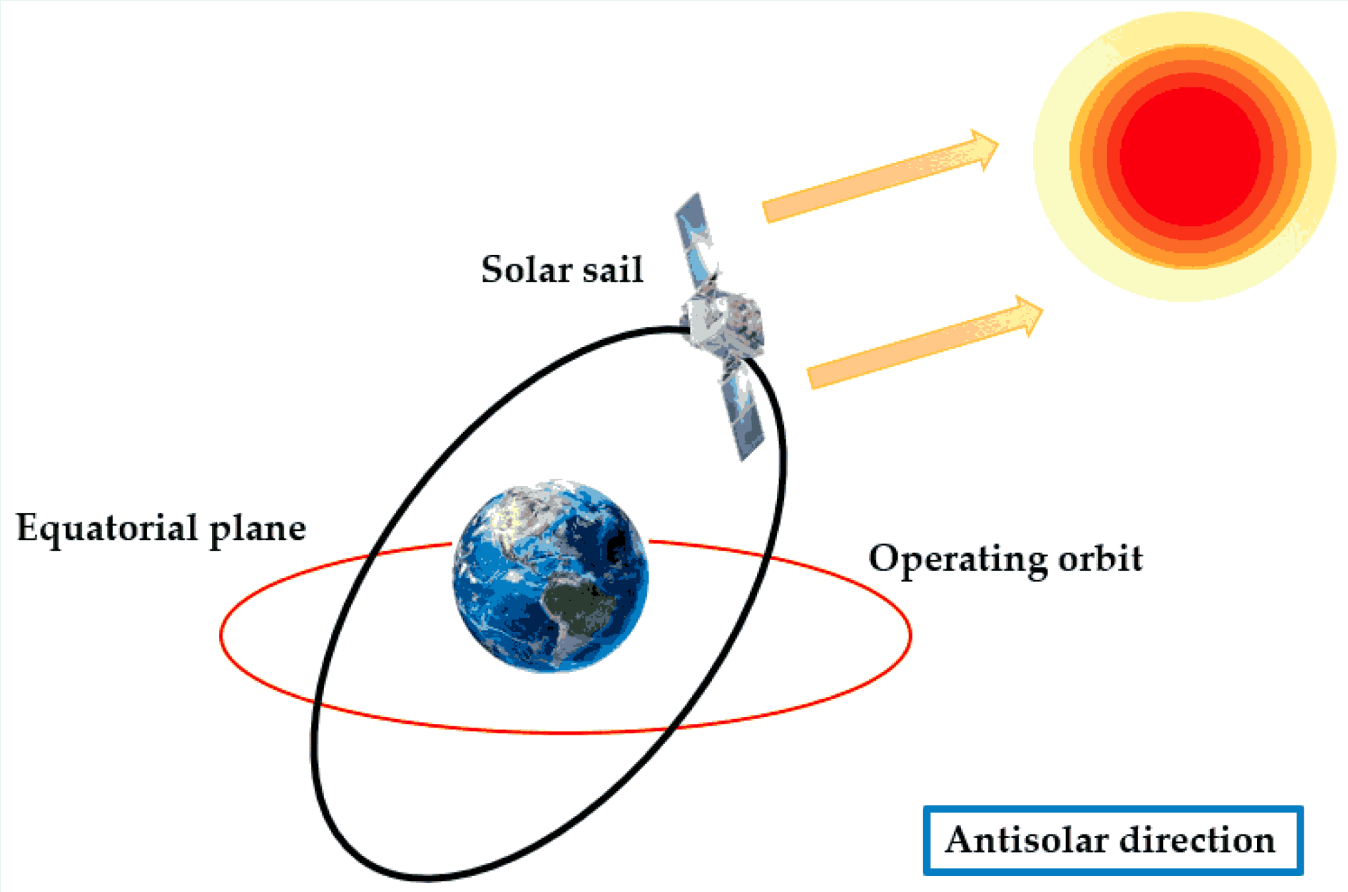
> **Click to reveal verification calculations**

Application 5: Satellite Attitude Control for Solar Tracking (Space Systems)

A satellite needs to reorient its solar panels to point toward the sun using reaction wheel actuation.

System Problem Statement

- **Configuration:** Spacecraft with 3-axis attitude control using reaction wheels
- **Objective:** Rotate solar panel normal from current orientation to point at sun
- **Task:** Calculate required spacecraft rotation using arbitrary axis rotation (Rodrigues formula)



What we need to calculate:

1. **Sun vector** in spacecraft body frame
2. **Rotation axis** (perpendicular to current and desired panel orientations)
3. **Rotation angle** (how much to rotate)
4. **Rotation matrix** using Rodrigues formula
5. **Verification** that panel normal aligns with sun vector

Key Question: How do we use arbitrary axis rotation to reorient the spacecraft solar panels toward the sun?

🛠️ Equivalent System Model



Given:

- **Spacecraft body frame:** X = velocity direction, Y = radial (toward Earth), Z = orbit normal
- **Solar panel normal (current):** Along +Z body axis = [0, 0, 1]
- **Sun direction (in body frame):** [0.6, 0, 0.8] (normalized)
- **Solar panel area:** 2.5 m², efficiency 30%
- **Task:** Find rotation matrix R that aligns panel normal with sun direction

What makes this interesting:

- We cannot decompose this into simple X, Y, Z rotations easily
- Instead, we use **Rodrigues formula** for arbitrary axis rotation
- This is how real spacecraft attitude control systems work

Step 1: Identify Current and Desired Panel Orientations

> **Click to reveal vector setup**

Step 2: Calculate Rotation Axis and Angle

> **Click to reveal rotation axis and angle calculations**

Step 3: Build Rotation Matrix Using Rodrigues Formula

> **Click to reveal Rodrigues rotation matrix construction**

Step 4: Verify the Rotation Works

> **Click to reveal verification**

> **Instrument it**
>
> The rotation matrices and quaternions here are exactly what an IMU reports. A 9-axis IMU (accelerometer, gyroscope, and magnetometer, such as the BNO055 or ICM-20948) fuses its readings into the same orientation you compute by hand, so you can check your transforms against a real board you tilt in your fingers. Wire it to an ESP32 or STM32 over I2C, and you can watch the live orientation on siliconwit.io and see the gyro drift that makes the magnetometer correction necessary.

Summary and Next Steps



In this lesson, you learned 3D rotation and transformation mathematics through theory and applications.

Key Skills Developed:

1. **Construct** rotation matrices for X, Y, Z coordinate axes using trigonometric relationships
2. **Apply** Euler angle sequences (ZYX, ZYZ) while managing singularities and gimbal lock

3. **Calculate** arbitrary axis rotations using decomposition method or Rodrigues formula
4. **Compose** 4×4 homogeneous transformation matrices for unified rotation and translation

Two Methods for Arbitrary Axis Rotations:

- **Decomposition approach:** Intuitive 5-step geometric method (best for education and analysis)
- **Rodrigues formula:** $R = I + \sin\theta\, W + (1 - \cos\theta) W^2$ (best for implementation)

Applications Covered:

- **Robotic welding** - Surface normals, tool orientation matrices, SLERP interpolation
- **Drone gimbal stabilization** - Matrix inversion, real-time compensation at 200Hz
- **Satellite attitude control** - Rodrigues formula for sun tracking with dual constraints

Critical Properties:

- Rotation matrices are orthogonal: $R^{-1} = R^T$
- Euler angles have gimbal lock at pitch = ±90°
- Matrix multiplication order matters: $R_3R_2R_1$ applies R_1 first

Coming Next: Lesson 4 develops systematic kinematic modeling using DH parameters for Stewart Platform analysis.

Comments

Loading comments...

© 2021-2026 SiliconWit®. All rights reserved.

Previous
Lesson 2: Planar Transformations and Foundations

Next
Lesson 4: Matrix Methods and Link Modeling

Privacy · Cookie settings