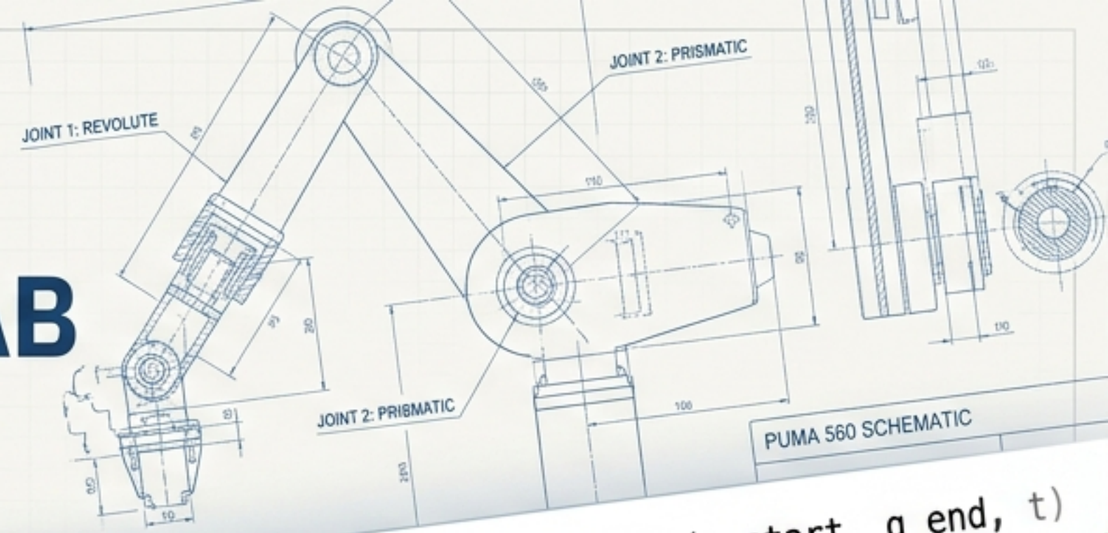
De la Teoría al Movimiento: Simulando Robots en MATLAB

Taller Práctico con el Toolbox de Peter Corke (Robotics Toolbox for MATLAB)



01
Instalación

02
Cinemática

03
Trayectorias

04
Proyecto Final: PUMA 560

```
function [q_d] = compute_trajectory(q_start, q_end, t)
    % Use robotics toolbox function for quintic polynomia
    q_d = jtraj(q_start, q_end, t);
end


puma560; % Load PUMA 560 robot model
q_home = puma.qz; % Home position
T_home = puma.fkine(q_home); % Forward kinematics
puma.plot(q_home, 'view', '3d'); % Visualize in 3D

% Define workspace
W = [-1 1 -1 1 -1 1];
puma.teach(W); % Interactive teach pendant
```

La Metodología: Del Concepto Físico a la Ejecución

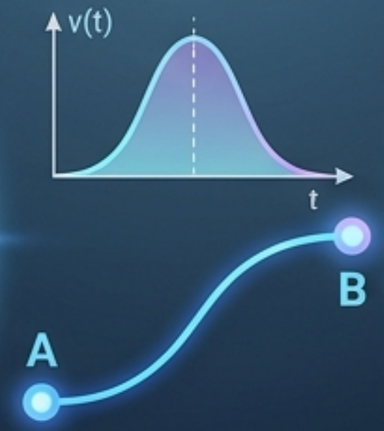
El Laboratorio

Configurar el entorno (MATLAB Desktop o Cloud).

¿Dónde Estamos?

Comprender el Espacio de Juntas vs. Espacio Cartesiano.

¿Hacia Dónde Vamos y a Qué Ritmo?



Cinemática y Jacobianos para trayectorias punto a punto.

La Obra Maestra

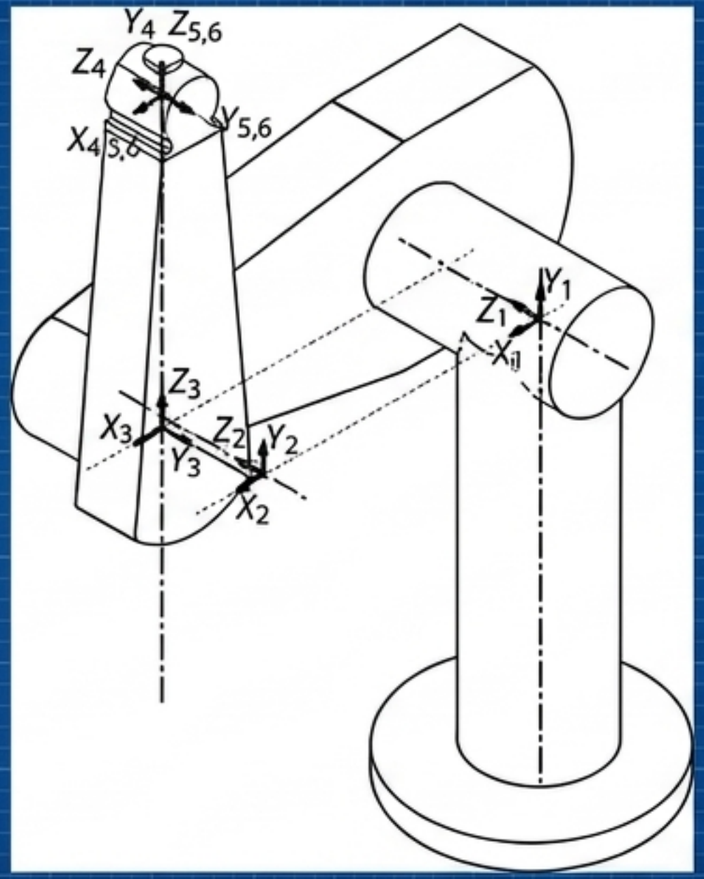
Integración en un proyecto real con el PUMA 560.

Fase 1: Preparando el Laboratorio de Código

Basado en la Edición 2 (RVC 2e) - Herramientas esenciales para cinemática y generación de trayectorias.

Fase 2: El Despertar del Robot (PUMA 560)

El Concepto

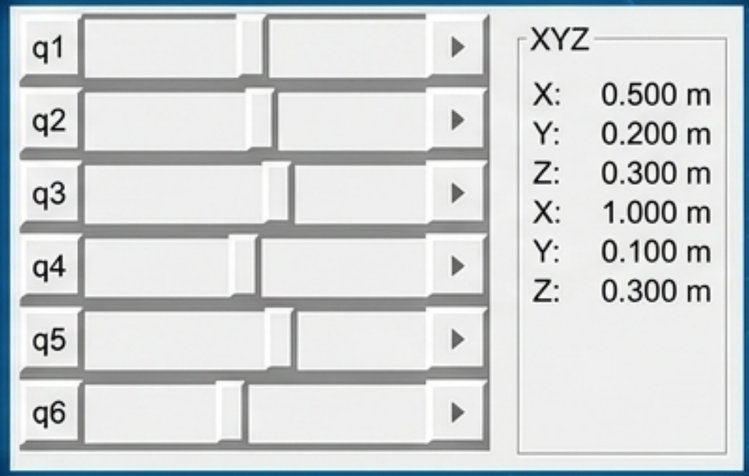


El PUMA 560 es un manipulador clásico de 6 grados de libertad (6 DoF). El toolbox ya contiene su modelo cinemático completo cargado con los parámetros de Denavit-Hartenberg.

El Código

```
% Instanciar el modelo del robot
mdl_puma560

% Abrir la interfaz manual interactiva
p560.teach()
```



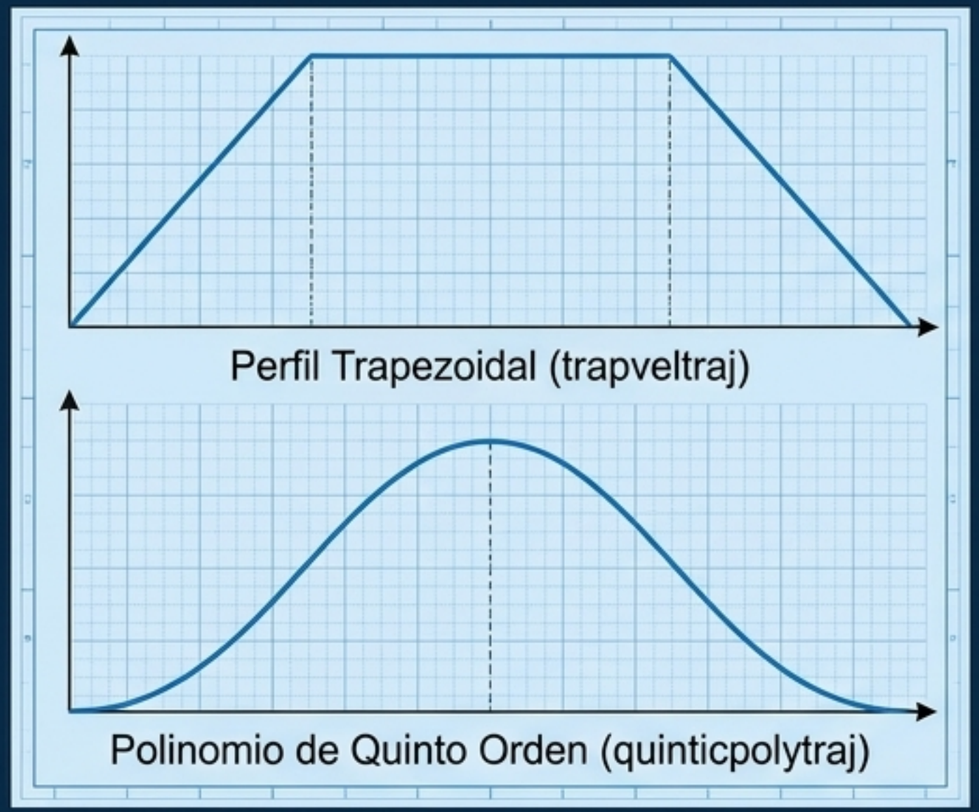
`teach()` permite interactuar manualmente con cada junta (movimiento Joint Jog) y observar el cambio inmediato en la posición y orientación del efector final.

Fase 3: ¿Dónde Estamos? El Problema de la Cinemática

	Cinemática Directa (Forward Kinematics)	Cinemática Inversa (Inversa Kinematics)
Flujo de Datos	$\begin{bmatrix} \theta1, \\ \theta2, \\ \dots \\ \theta6 \end{bmatrix}$ →  → $\begin{bmatrix} X, Y \\ Z, \\ R, \\ P, Y \end{bmatrix}$	$\begin{bmatrix} X, Y \\ Z, \\ R, \\ P, Y \end{bmatrix}$ →  → $\begin{bmatrix} \theta1, \\ \theta2, \\ \dots \\ \theta6 \end{bmatrix}$
Función en el Toolbox	`T = p560.fkine(q);` (Retorna una matriz de transformación homogénea SE(3)).	`q = p560.ikine6s(T);` (Calcula la solución analítica para un robot con muñeca esférica).
Naturaleza del Cálculo	Directo y Único. Siempre hay una sola posición espacial para un conjunto de ángulos.	Complejo y Múltiple. Pueden existir múltiples configuraciones de juntas (ej. codo arriba / codo abajo) para llegar al mismo punto.

Fase 4: Hacia Dónde Vamos (Generación de Trayectorias)

Perfiles de Velocidad



Un robot no salta mágicamente de un punto a otro. Necesita una interpolación suave para evitar aceleraciones infinitas que dañarían los motores.

Código de Movimiento

```
% Generar trayectoria entre dos configuraciones articulares
% q1 = inicio, q2 = destino, 50 = número de pasos
qt = jtraj(q1, q2, 50);

% Animar el movimiento en 3D
p560.plot3d(qt)
```

La función `jtraj` utiliza un polinomio de quinto orden por defecto, asegurando que la velocidad y aceleración sean cero al inicio y al final del movimiento.

Fase 5: A Qué Ritmo (La “Caja de Cambios” Jacobiana)

ν (Vector Cartesiano):
Velocidad lineal y angular del efector final $[v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$.

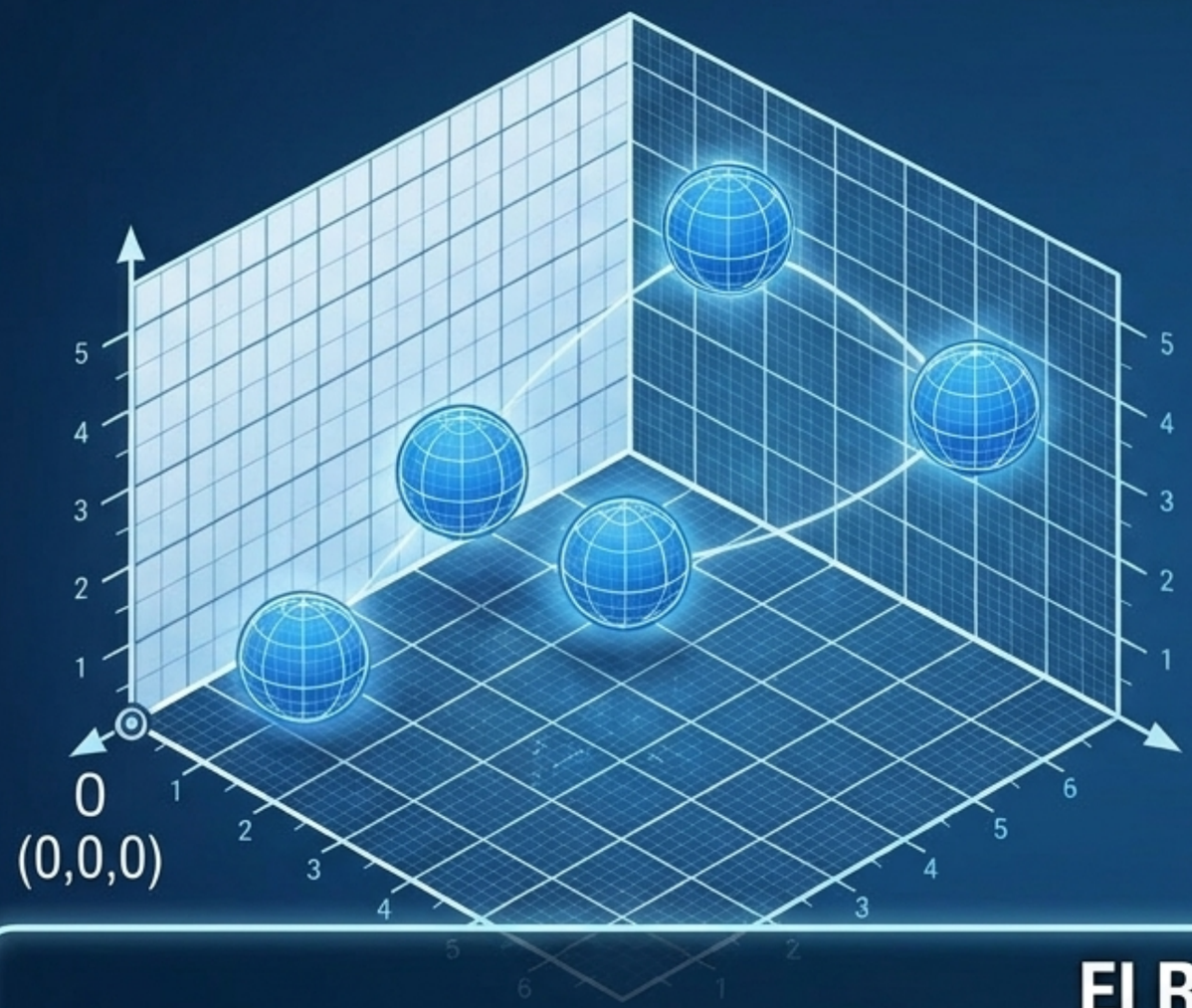
$$\nu = J(q) \cdot \dot{q}$$

$J(q)$ (La Matriz Jacobiana):
La caja de cambios dinámica. Cambia dependiendo de la pose actual del robot.
Si el robot se acerca a una singularidad (brazo totalmente estirado), los engranajes se bloquean. ⚠

$\dot{q}$ (Vector Articular):
Velocidad de rotación individual de cada motor $[\dot{q}_1, \ldots, \dot{q}_6]^T$.

```
% Calcular el Jacobiano geométrico en la pose 'q'
J = p560.jacob0(q);
```

Fase 6: Obra Maestra (Planteamiento del Proyecto)



La Misión

Programar el PUMA 560 para que su efector final visite secuencialmente cuatro coordenadas esféricas en el espacio 3D, partiendo desde una posición de reposo específica.

Coordenadas del Sistema

- **Pose Inicial (q0)**: [-1.5708, 1.5708, -1.5708, 0, 0, 0] radianes.
- **Esfera 1**: [1, 0, 0]
- **Esfera 2**: [0, 1, 0]
- **Esfera 3**: [0, 0, 1]
- **Esfera 4**: [-0.0203, 0.15, 1.064]

El Reto

Convertir coordenadas X,Y,Z a ángulos de motor (Cinemática Inversa) y enlazar los movimientos sin colisiones bruscas (Generación de Trayectorias).

Arquitectura de la Solución: El Pipeline de Datos

Paso 1: Transformación Espacial

Convertir la coordenada [x,y,z] en una Matriz de Transformación Homogénea.

```
T_target = transl(x, y, z) * troty(90);
```

Paso 2: Cinemática Inversa

Encontrar los ángulos de las juntas para alcanzar T_target.

```
q_target = p560.ikine6s(T_target, 'ld');
```

Paso 3: Interpolación de Trayectoria

Crear 50 pasos suaves entre la pose actual y la meta.

```
qt = jtraj(q_current, q_target, 50);
```

Paso 4: Ejecución y Trazado

Actualizar el estado y dibujar el robot en 3D.

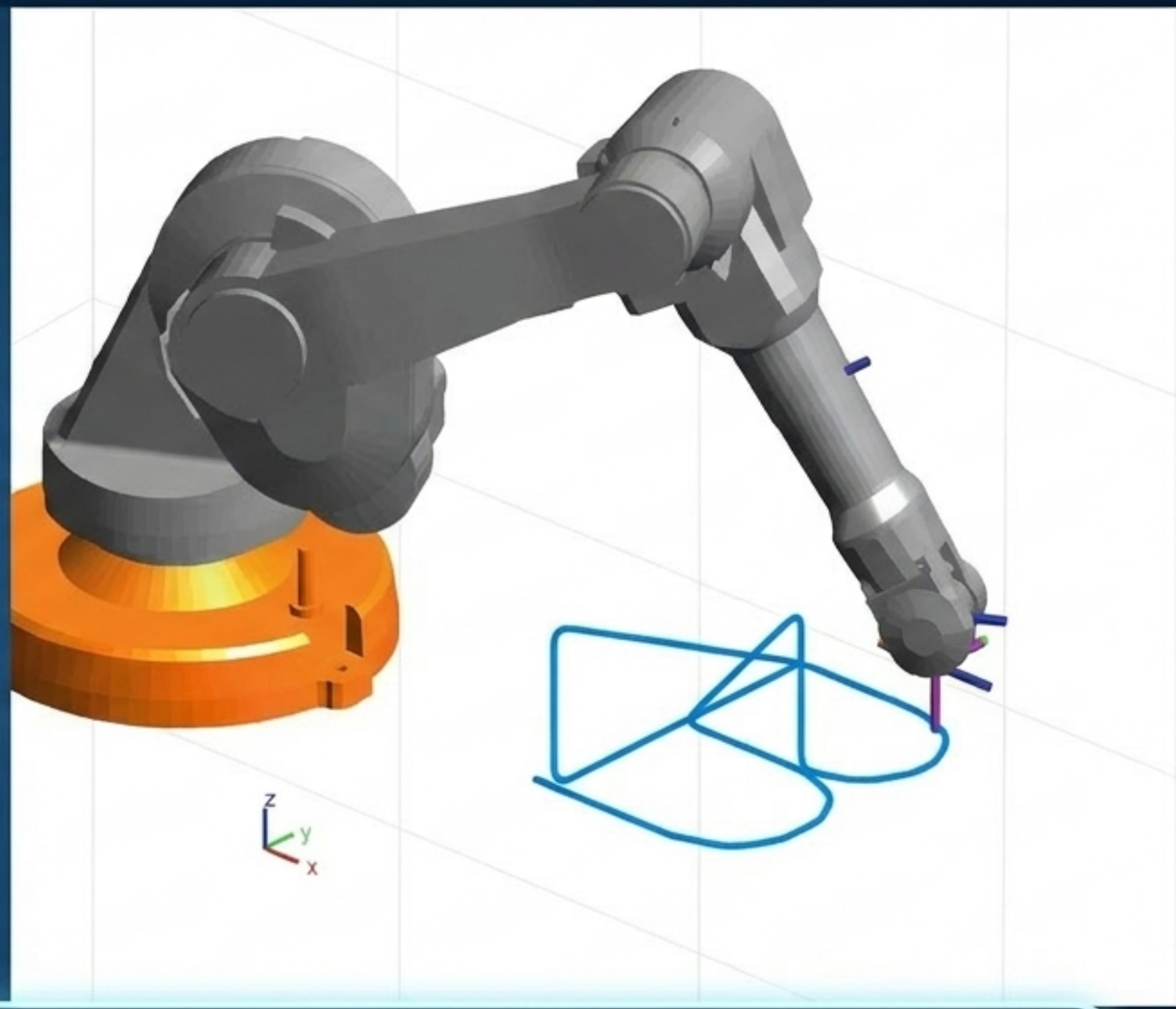
```
p560.plot3d(qt);
```

Ejecución: Uniendo el Espacio y el Tiempo

Código de Ejecución Final

Código de Ejecución Final

```
% Bucle sobre cada Esfera
for i = 1:size(esferas, 1)
    % 1. Matriz T
    T_goal = transl(esferas(i, :)) * troty(90);
    % 2. IK
    q_goal = p560.ikine6s(T_goal, 'ld');
    % 3. Trayectoria
    qt = jtraj(q_actual, q_goal, 50);
    % 4. Animación
    p560.plot3d(qt);
    % Actualizar posición
    q_actual = q_goal;
end
```



Al dominar ikine y jtraj, hemos cerrado el puente entre la teoría matemática y la simulación robótica funcional.