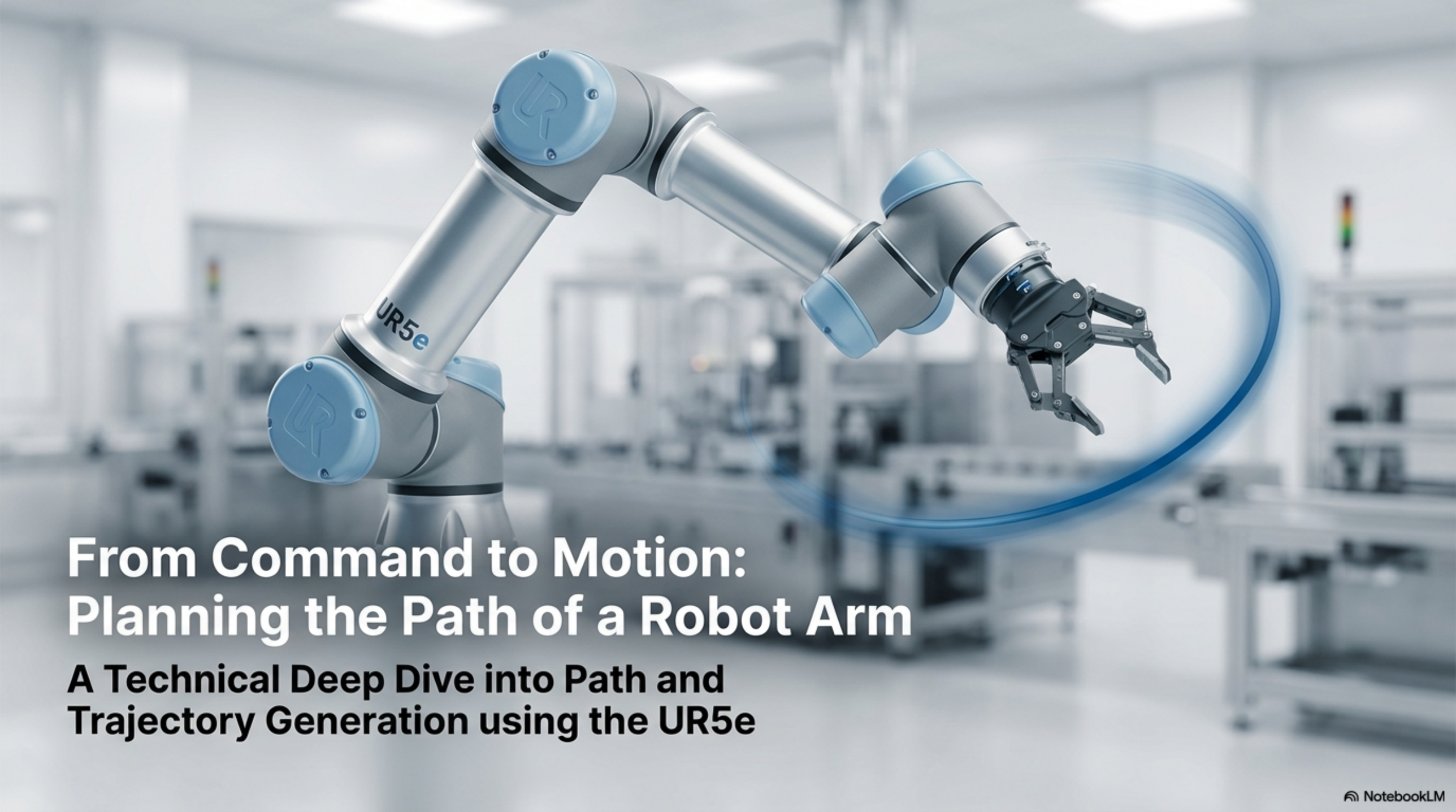
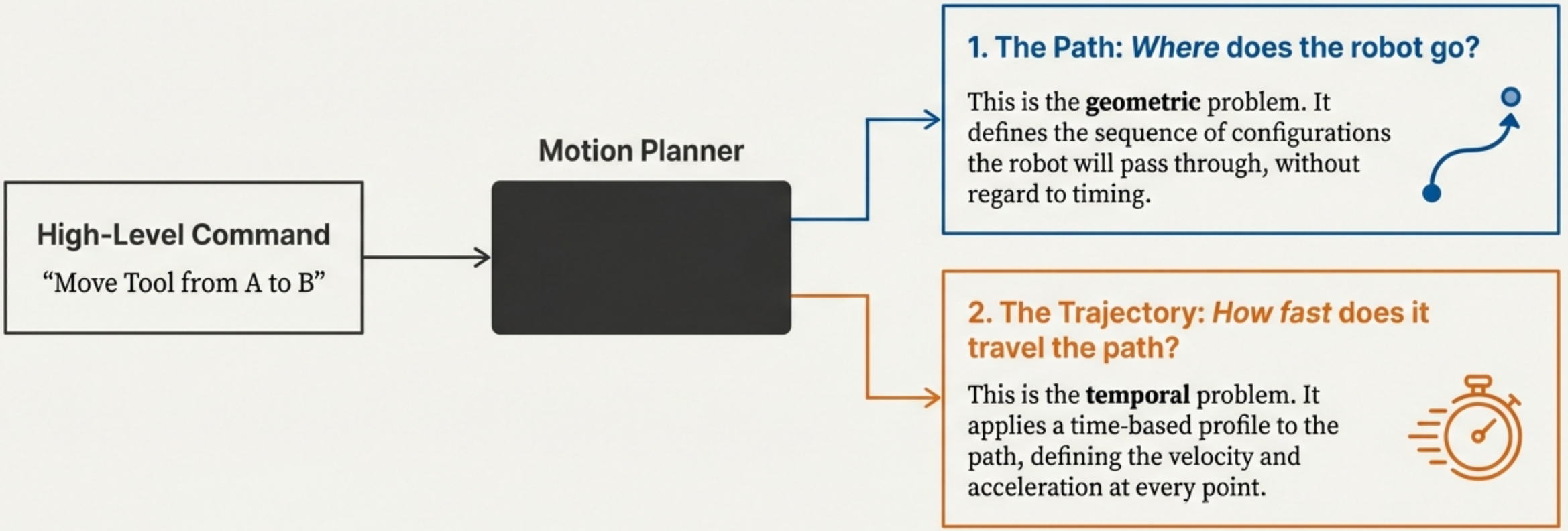


From Command to Motion: Planning the Path of a Robot Arm

A Technical Deep Dive into Path and Trajectory Generation using the UR5e

The Central Challenge: Translating ‘Go from A to B’ into Physical Reality

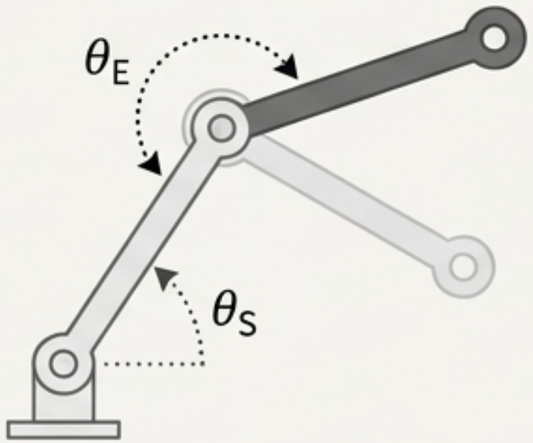
At its core, robot motion planning is the bridge between a high-level task and the low-level joint commands that execute it. It’s the process of generating a smooth, controlled, and physically achievable motion for the manipulator. This process breaks down into two fundamental questions:



Part 1: Defining the Path – The Geometry of Motion

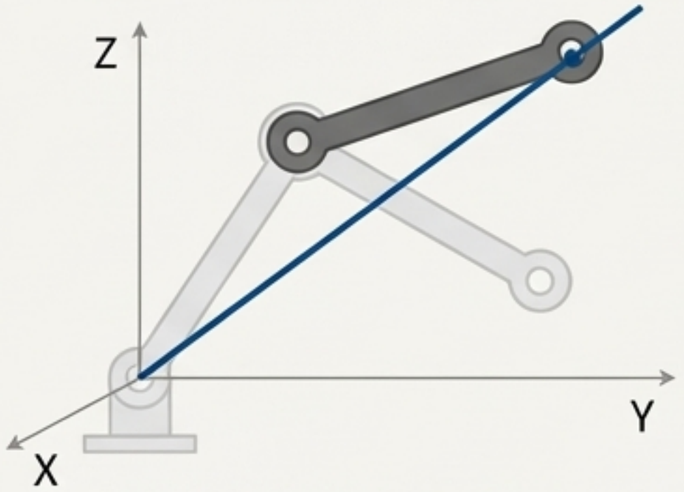
Before a robot can move, we must first define the geometric path its end-effector will follow. This path can be planned in one of two distinct spaces, each with significant trade-offs in terms of computational complexity and path predictability.

Joint Space



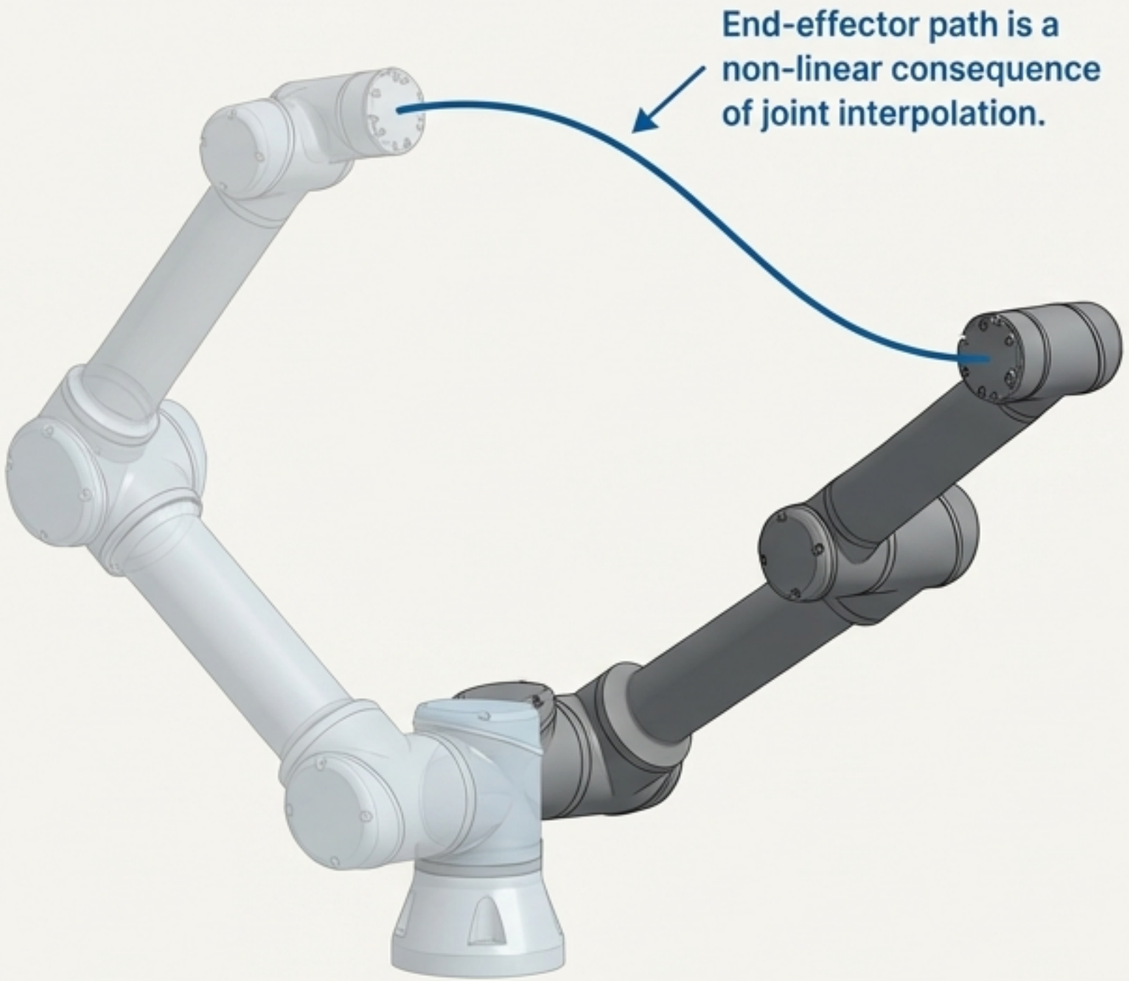
The path is defined by interpolating the angles of each individual robot joint. This is the most direct and computationally simple method.

Cartesian Space



The path is defined by a geometric shape (e.g., a straight line) for the end-effector in 3D space. This is more intuitive for many tasks but requires more complex calculations.

Joint Space Motion: The Path of Least Resistance



In Joint Space motion (or `moveJ`), the path is generated by independently interpolating each joint from its start angle to its end angle. The controller ensures all joints start and stop simultaneously. The resulting end-effector path is a consequence of this joint-level interpolation.

Characteristics

- The end-effector follows a curved, often non-intuitive path through space.
- This represents the “path of least resistance” for the robot’s motors.

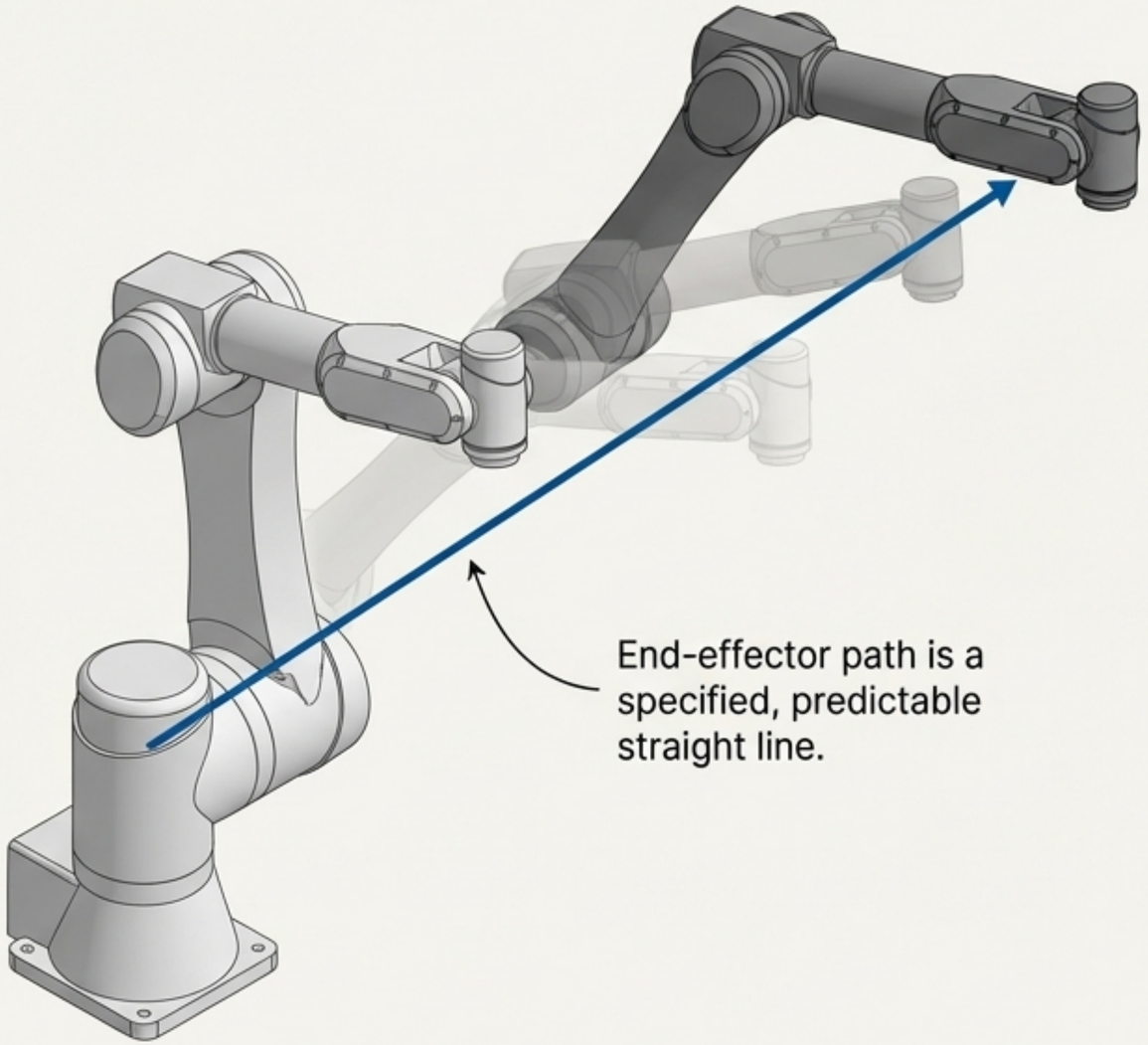
✅ Pros

- ✓ **Computationally Fast:** Requires only one inverse kinematics (IK) calculation for the endpoint.
- ✓ **Robust:** Less likely to encounter singularities mid-path compared to Cartesian moves.

❌ Cons

- ❌ **Unpredictable Path:** The end-effector path is not a straight line and can be difficult to visualize.
- ❌ **Collision Risk:** The non-intuitive path may lead to collisions with objects in the workspace.

Cartesian Space Motion: The Predictable Straight Line



In Cartesian Space motion (or `moveL`), the path for the tool's center point (TCP) is specified as a straight line between the start and end positions. To achieve this, the controller must continuously calculate the required joint angles at a high frequency throughout the motion.

Characteristics

- The end-effector moves in a predictable, straight line.
- Orientation can also be interpolated smoothly from start to end.

Pros

✅ **Intuitive & Task-Oriented**: Ideal for tasks like welding, sealing, or inserting objects where a straight-line path is critical.

✅ **Predictable**: The path is easily visualized and avoids unexpected movements.

Cons

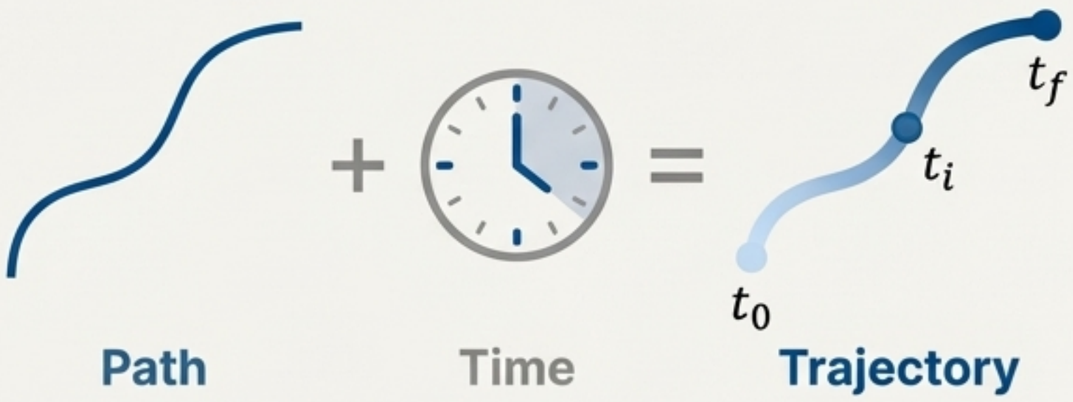
❌ **Computationally Intensive:** Requires continuous inverse kinematics calculations.

❌ **Singularity Risk**: The path may pass through or near a kinematic singularity, causing joint velocities to approach infinity.

Part 2: Planning the Trajectory – The Profile of Motion Over Time

Once a geometric path is defined (in either Joint or Cartesian space), we must determine *how* the robot moves along it over time. This is trajectory planning.

A simple 'bang-bang' control (instantaneous acceleration/deceleration) is physically impossible and would cause extreme wear on the robot. Instead, we need a smooth function that defines position, velocity, and acceleration over the duration of the move.



Key Constraints for any Trajectory

- Start Position: $\theta(0) = \theta_0$
- End Position: $\theta(t_f) = \theta_f$
- Start Velocity: $\dot{\theta}(0) = \dot{\theta}_0$ (usually 0)
- End Velocity: $\dot{\theta}(t_f) = \dot{\theta}_f$ (usually 0)

Trajectory Method 1: Cubic Polynomials

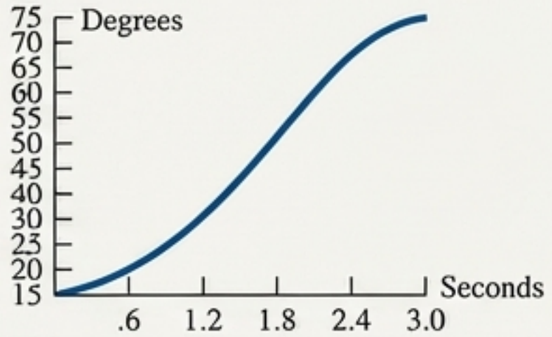
A third-order (cubic) polynomial is the simplest function that can satisfy the four boundary conditions for position and velocity.

Position: $\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

Velocity: $\dot{\theta}(t) = a_1 + 2 a_2 t + 3 a_3 t^2$

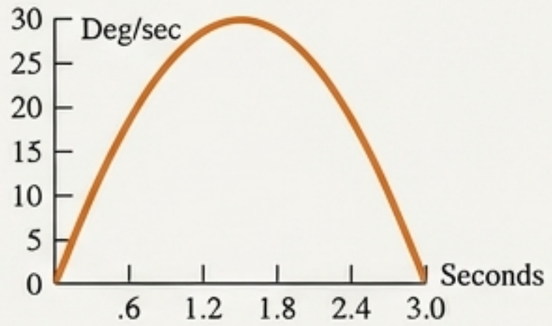
Acceleration: $\ddot{\theta}(t) = 2 a_2 + 6 a_3 t$

The coefficients (a_0, a_1, a_2, a_3) are calculated based on the start/end positions and velocities. This method guarantees continuous velocity, but acceleration changes linearly, which means jerk (the derivative of acceleration) is not continuous.



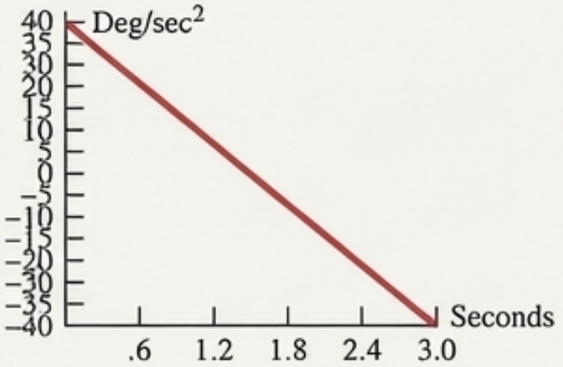
Position

A smooth 'S' curve.



Velocity

A parabolic curve, starting and ending at zero.



Acceleration

A straight line, indicating a non-zero jerk at the start and end.

Trajectory Method 2: Linear Segments with Parabolic Blends (LSPB)

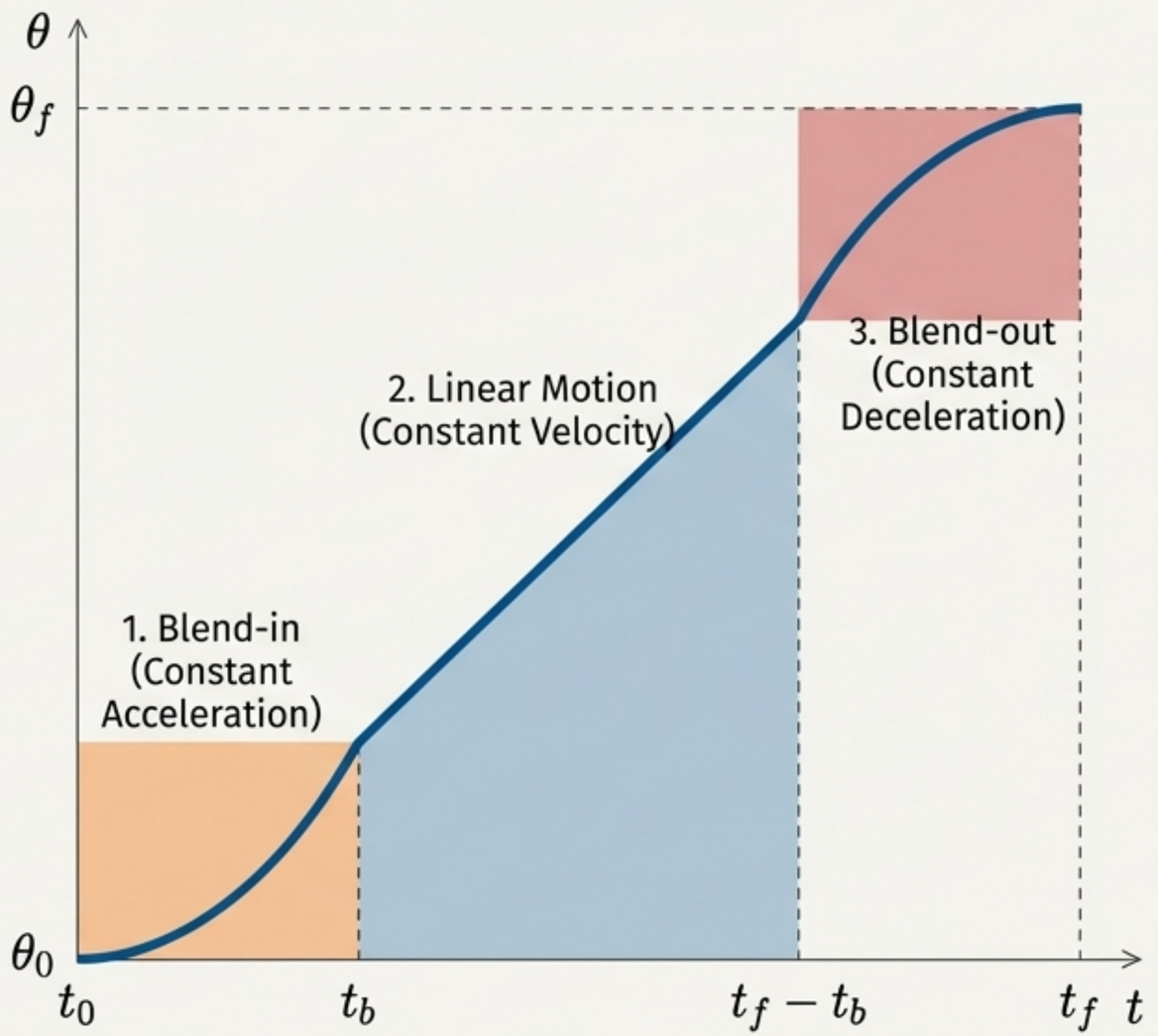
LSPB, also known as a trapezoidal velocity profile, offers more direct control over motion parameters. The trajectory is divided into three segments:

1. **Parabolic Blend (Acceleration):** The robot accelerates at a constant rate until it reaches a desired cruising velocity.
2. **Linear Segment (Cruise):** The robot moves at a constant velocity.
3. **Parabolic Blend (Deceleration):** The robot decelerates at a constant rate to a stop.

This method is highly intuitive because it allows the programmer to specify a maximum velocity and acceleration for the move.

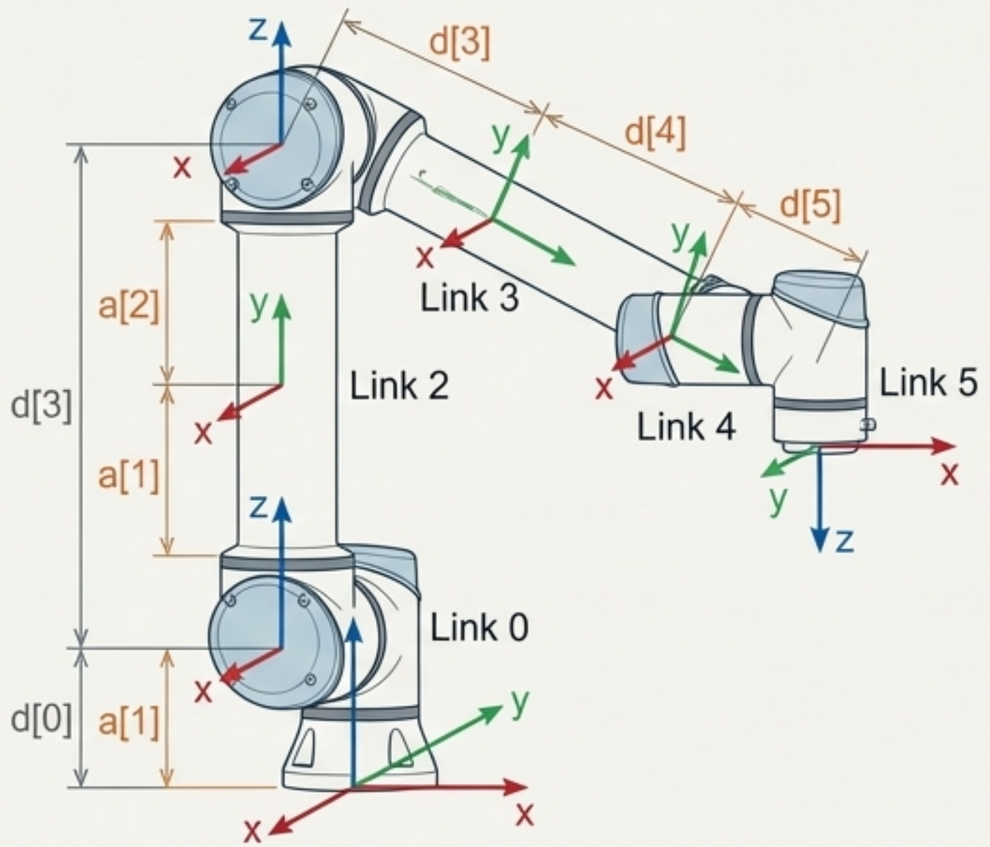
Characteristic Profiles

- **Position:** A curve with a linear mid-section.
- **Velocity**: A trapezoidal shape.
- **Acceleration:** Piecewise constant (bang-bang jerk), which is simple but can still induce vibrations.



Application & Synthesis: Motion Planning for the UR5e

Let's apply these concepts to a real-world example: the Universal Robots UR5e, a 6-axis collaborative robot arm. To plan its motion, we must first understand its fundamental kinematic structure. This structure is formally described using Denavit-Hartenberg (DH) parameters.



The DH Parameter Table

The DH parameters provide the "kinematic DNA" of a serial manipulator. For each link, four parameters define the geometric relationship to the next:

- a_{i-1}: link length
- α_{i-1}: link twist
- d_i: link offset
- θ_i: joint angle (variable for revolute joints)

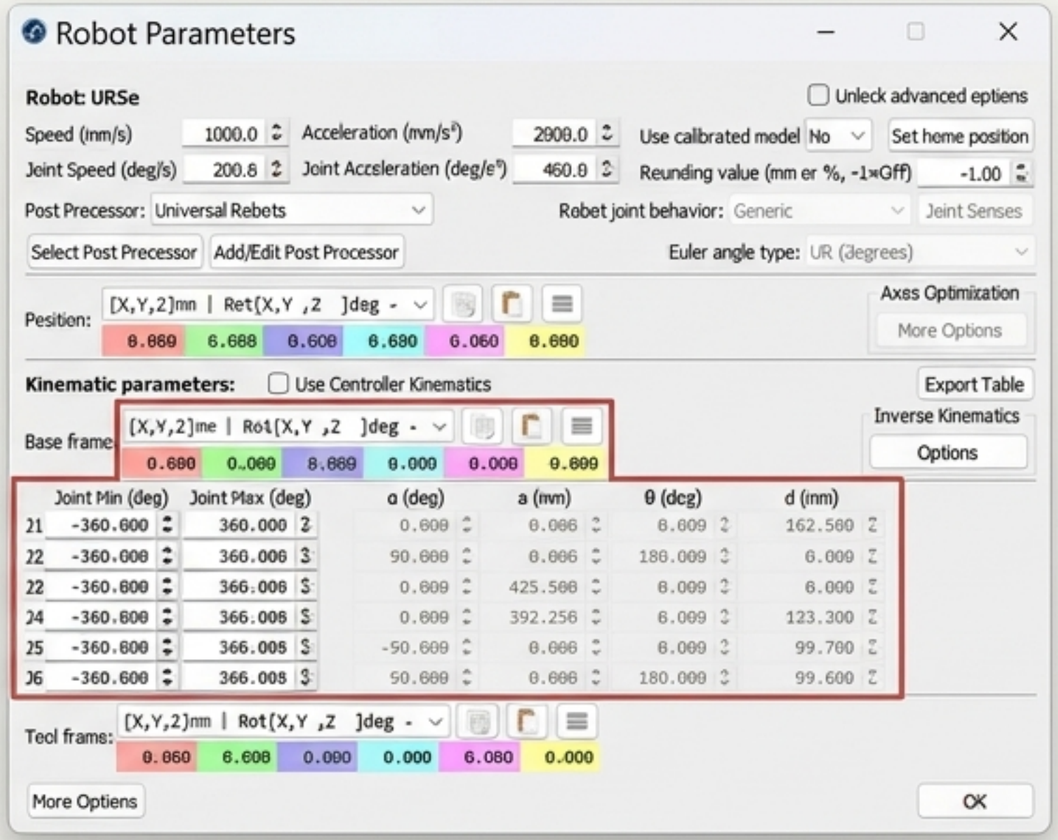
The Kinematic DNA of the UR5e

The Denavit-Hartenberg parameters for the UR5e define the precise geometry of its six-link chain. This table is the input for all forward and inverse kinematic calculations.

Official 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

*Note: Software like RoboDK may use a 'Modified DH' convention, which can alter the parameter values but describes the same physical robot.



From Joint Angles to Cartesian Pose: Forward Kinematics

With the DH parameters, we can systematically compute the end-effector's position and orientation for any given set of joint angles $(\theta_1, \ldots, \theta_6)$.

1. Link Transformations

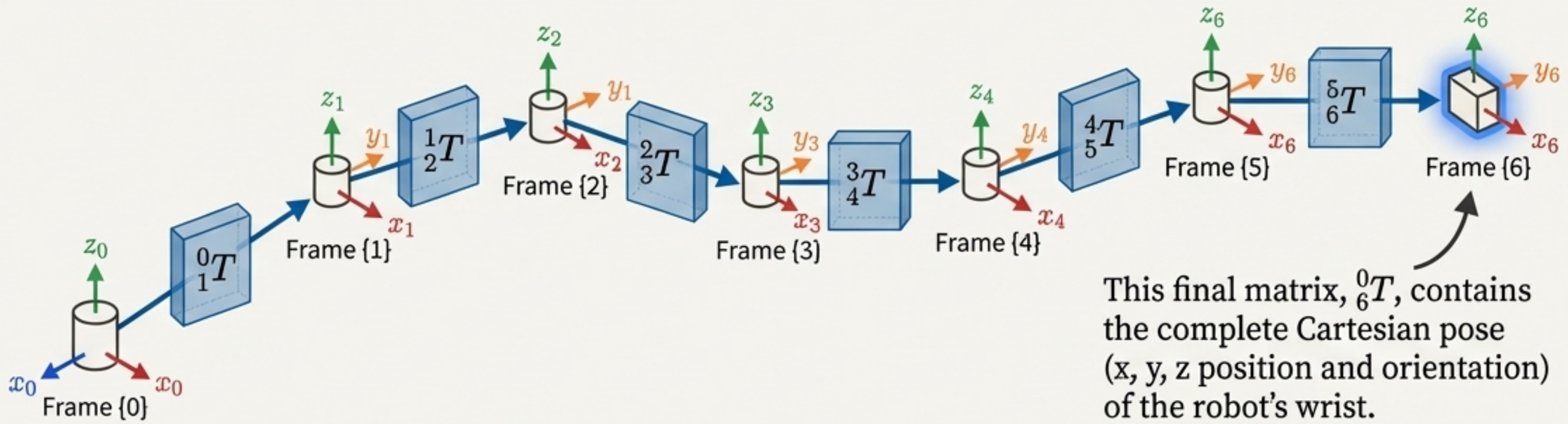
For each link i, the DH parameters are used to construct a 4×4 homogeneous transformation matrix, ${}^{i-1}_{i}T$, which describes the position and orientation of frame $\{i\}$ relative to frame $\{i-1\}$.

$$ {}^{i-1}_{i}T = R_z(\theta_i) D_z(d_i) D_x(a_{i-1}) R_x(\alpha_{i-1}) $$

2. Chain Multiplication

These matrices are multiplied together to find the final transformation from the base frame $\{0\}$ to the end-effector (wrist) frame $\{6\}$.

$$ {}^{0}_{6}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T \cdot {}^{4}_{5}T \cdot {}^{5}_{6}T $$



From Cartesian Pose to Joint Angles: The Inverse Kinematics Problem

In most applications, we know the desired Cartesian pose of the tool and need to find the joint angles to achieve it. This is the inverse kinematics (IK) problem.

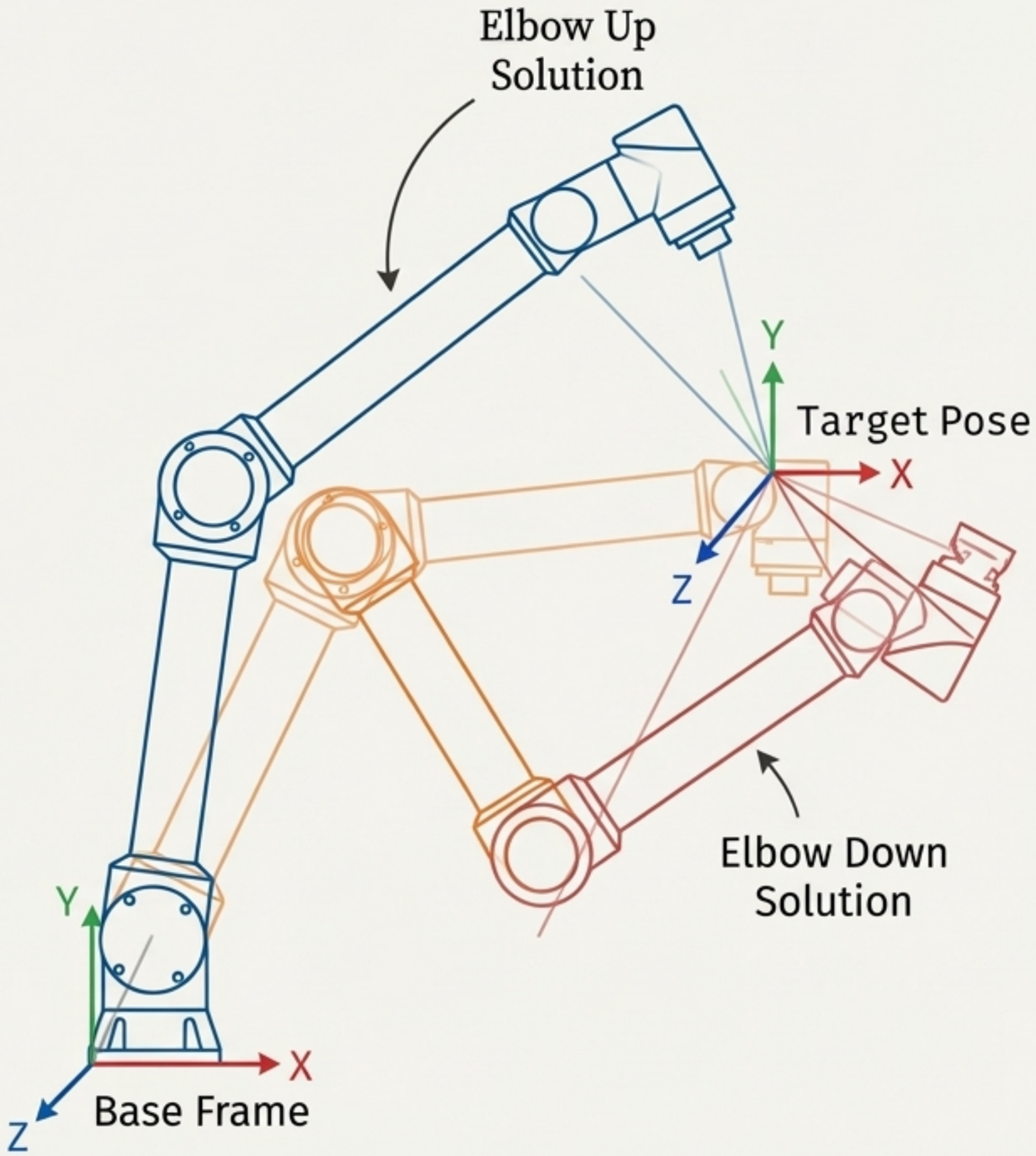
Given: A desired pose matrix ${}^{0}_{6}T$.

Find: The set of joint angles $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$.

Unlike forward kinematics, the inverse problem is a set of non-linear, transcendental equations. For a 6-axis arm like the UR5e... there can be up to **8 unique joint configurations** that result in the exact same end-effector pose.

The controller must choose one solution, often based on:

- **Proximity:** The solution requiring the smallest total joint movement.
- **Configuration:** Avoiding certain poses (e.g., 'elbow up' vs. 'elbow down').
- **Joint Limits:** Discarding solutions that violate joint range constraints.

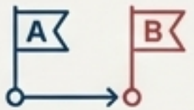


The Full Picture: Executing a ‘moveL’ Command on the UR5e

Let’s trace the complete process for a linear Cartesian move from pose A to pose B.

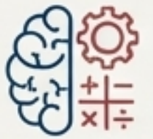
1 Define Endpoints

The system starts with the Cartesian poses for A (${}^0T_A$) and B (${}^0T_B$).



2 Solve IK for Endpoints

$q_A = \text{IK}({}^0T_A)$
$q_B = \text{IK}({}^0T_B)$
The controller chooses the most appropriate solution for q_B (e.g., the one ‘closest’ to q_A).



3 Path Generation (Geometric)

The path is defined as a straight line in Cartesian space from A to B.
The path is discretized into many intermediate points (${}^0T_i$).
IK is solved for each point i to get a sequence of joint configurations q_i.

4 Trajectory Generation (Temporal)

A trajectory profile (e.g., LSPB) is applied to the sequence of joint configurations q_i.
This generates the time-stamped position, velocity, and acceleration commands for each joint motor, respecting the robot's physical limits.

Key Decisions in Motion Planning

Effective robot motion planning hinges on understanding and managing fundamental trade-offs. The choice of how to move a robot from A to B is a strategic decision based on the requirements of the task.

PATH (The *Where*)

The geometric problem.

Joint Space

Computationally simple, fast, robust against singularities. Use when the exact path doesn't matter.

Cartesian Space

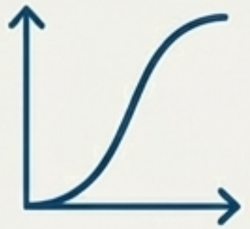
Predictable, intuitive straight-line motion. Use when the path is critical to the task (e.g., welding, assembly).

TRAJECTORY (The *How Fast*)

The temporal problem.

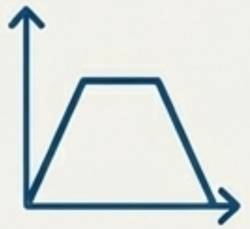
Cubic Polynomials

Ensures continuous velocity. A classic, mathematically simple approach.



LSPB (Trapezoidal)

Provides direct control over cruising velocity and constant acceleration. More intuitive for tuning motion profiles.



Mastering these concepts allows a programmer to move beyond simple 'teach-and-playback' and unlock the full potential of a robot arm's flexibility and precision.